A Theory Parts

A.1 Proof of Theorem 3.1

The proof of Theorem 3.1 is based on the so-called Score-projection identity which was first found in Vincent [60] to bridge denoising score matching and denoising auto-encoders. Later the identity is applied by Zhou et al. [79] for deriving distillation methods based on Fisher divergences. We appreciate the efforts of Zhou et al. [79] and re-write the score-projection identity here without proof. Readers can check Zhou et al. [79] for a complete proof of score-projection identity.

Theorem A.1. Let $\boldsymbol{u}(\cdot)$ be a vector-valued function, using the notations of Theorem 3.1, under mild conditions, the identity holds:

$$\mathbb{E}_{\boldsymbol{x}_t\sim p_{\theta,t}}\boldsymbol{u}(\boldsymbol{x}_t)^T\Big\{\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)-\nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)\Big\}=0,\quad\forall\theta.$$

Next, we turn to prove the Theorem 3.1.

Proof. We prove a more general result. Let $\boldsymbol{u}(\cdot)$ be a vector-valued function, the so-called score-projection identity [79, 60] holds,

$$\mathbb{E}_{\boldsymbol{x}_t\sim p_{\theta,t}}\boldsymbol{u}(\boldsymbol{x}_t)^T\Big\{\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)-\nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)\Big\}=0,\quad\forall\theta. \tag{A.1}$$

Notice that for most commonly used forward diffusion processes such as VP and VE process [57], the term $\nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)$ turns out to be a scale of the difference of an added Gaussian noise ϵ, therefore the θ gradient for $\nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)$ will vanish. Taking θ gradient on both sides of identity (A.1), we have

$$0=\mathbb{E}_{\boldsymbol{x}_t\sim p_{\theta,t}}\frac{\partial}{\partial\boldsymbol{x}_t}\Big\{\boldsymbol{u}(\boldsymbol{x}_t)^T\big\{\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)-\nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)\big\}\Big\}\frac{\partial\boldsymbol{x}_t}{\partial\theta}+\mathbb{E}_{\boldsymbol{x}_t\sim p_{\theta,t}}\boldsymbol{u}(\boldsymbol{x}_t)^T\frac{\partial}{\partial\theta}\Big\{\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)\Big\}$$

So we have an identity

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}_t\sim p_{\theta,t}}\boldsymbol{u}(\boldsymbol{x}_t)^T\frac{\partial}{\partial\theta}\Big\{\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)\Big\}&=-\mathbb{E}_{\boldsymbol{x}_t\sim p_{\theta,t}}\frac{\partial}{\partial\boldsymbol{x}_t}\Big\{\boldsymbol{u}(\boldsymbol{x}_t)^T\big\{\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)-\nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)\big\}\Big\}\frac{\partial\boldsymbol{x}_t}{\partial\theta}\\&=-\frac{\partial}{\partial\theta}\mathbb{E}_{\boldsymbol{x}_t\sim p_{\theta,t}}\Big\{\boldsymbol{u}(\boldsymbol{x}_t)\big\{\boldsymbol{s}_{p_{\mathrm{sg}[\theta],t}}(\boldsymbol{x}_t)-\nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)\big\}\Big\}\end{aligned} \tag{A.2}$$

Notice that the left-hand side of equation (A.2) can be interpreted as the gradient of the loss function when the parameter dependency of the sampling distribution is cut off, i.e.

$$\mathbb{E}_{\boldsymbol{x}_t\sim p_{\theta,t}}\boldsymbol{u}(\boldsymbol{x}_t)^T\frac{\partial}{\partial\theta}\Big\{\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)\Big\}=\frac{\partial}{\partial\theta}\mathbb{E}_{\boldsymbol{x}_t\sim p_{\mathrm{sg}[\theta],t}}\Big\{\boldsymbol{u}(\boldsymbol{x}_t)^T\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)\Big\} \tag{A.3}$$

Therefore we have the final equation

$$\frac{\partial}{\partial\theta}\mathbb{E}_{\boldsymbol{x}_t\sim p_{\mathrm{sg}[\theta],t}}\Big\{\boldsymbol{u}(\boldsymbol{x}_t)^T\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)\Big\}=-\frac{\partial}{\partial\theta}\mathbb{E}_{\boldsymbol{x}_t\sim p_{\theta,t}}\Big\{\boldsymbol{u}(\boldsymbol{x}_t)\big\{\boldsymbol{s}_{p_{\mathrm{sg}[\theta],t}}(\boldsymbol{x}_t)-\nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)\big\}\Big\} \tag{A.4}$$

which holds for arbitrary function $\boldsymbol{u}(\cdot)$ and parameter θ. If we set

$$\begin{aligned}\boldsymbol{u}(\boldsymbol{x}_t)&=\mathbf{d}'(\boldsymbol{y}_t)\\\boldsymbol{y}_t&=\boldsymbol{s}_{p_{\mathrm{sg}[\theta],t}}(\boldsymbol{x}_t)-\boldsymbol{s}_{q_t}(\boldsymbol{x}_t)\end{aligned}$$

Then we formally have

$$\begin{aligned}&\frac{\partial}{\partial\theta}\mathbb{E}_{\boldsymbol{x}_t\sim p_{\mathrm{sg}[\theta],t}}\Big\{\mathbf{d}'(\boldsymbol{y}_t)\Big\}^T\Big\{\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)\Big\}\\&=\frac{\partial}{\partial\theta}\mathbb{E}_{\substack{\boldsymbol{x}_0\sim p_{\theta,0},\\\boldsymbol{x}_t|\boldsymbol{x}_0\sim q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)}}\Big\{-\mathbf{d}'(\boldsymbol{y}_t)\Big\}^T\Big\{\boldsymbol{s}_{p_{\theta,t}}(\boldsymbol{x}_t)-\nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t|\boldsymbol{x}_0)\Big\}\end{aligned} \tag{A.5}$$

$\square$

A.2 Pytorch style pseudo-code of Score Implicit Matching

In this section, we give a PyTorch style pseudo-code for algorithm 1, with the Pseudo-Huber distance function. For a detailed algorithm on CIFAR10 with EDM model, please check Algorithm 2.

```
import torch
import torch.nn as nn
import torch.optim as optim

# Initialize generator G
G = Generator()

## load teacher DM
Sd = DiffusionModel().load('/path_to_ckpt').eval().requires_grad_(False)
Sg = copy.deepcopy(Sd) ## initialize online DM with teacher DM

# Define optimizers
opt_G = optim.Adam(G.parameters(), lr=0.001, betas=(0.0, 0.999))
opt_Sg = optim.Adam(Sg.parameters(), lr=0.001, betas=(0.0, 0.999))

# Training loop
while True:
    ## update Sg
    Sg.train().requires_grad_(True)
    G.eval().requires_grad_(False)

    # loop for 2 times to update Sg
    for _ in range(2):
      z = torch.randn((2000, 2)).to(device)
      with torch.no_grad():
        fake_x = G(z)

      t = torch.from_numpy(np.random.choice(np.arange(1,Sd.T), size=fake_x.shape[0], replace=True)).to(device).long()
      fake_xt, t, noise, sigma_t, g2_t = Sd(fake_x, t=t, return_t=True)
      sigma_t = sigma_t.view(-1,1).to(device)
      g2_t = g2_t.to(device)
      score = Sg(torch.cat([fake_xt,t.view(-1,1)/Sd.T],-1))/sigma_t

      batch_sg_loss = score + noise/sigma_t
      batch_sg_loss = (g2_t*batch_sg_loss.square().sum(-1)).mean()*Sd.T

      optimizer_Sg.zero_grad()
      batch_sg_loss.backward()
      optimizer_Sg.step()


    ## update G
    Sg.eval().requires_grad_(False)
    G.train().requires_grad_(True)

    z = torch.randn((2000, 2)).to(device)
    fake_x = G(z)

    t = torch.from_numpy(np.random.choice(np.arange(1,diffusion.T), size=fake_x.shape[0], replace=True)).to(device).long()
    fake_xt, t, noise, sigma_t, g2_t = diffusion(fake_x, t=t, return_t=True)
    sigma_t = sigma_t.view(-1,1).to(device)
    g2_t = g2_t.to(device)

    score_true = Sd(torch.cat([fake_xt,t.view(-1,1)/diffusion.T],-1))/sigma_t
    score_fake = Sg(torch.cat([fake_xt,t.view(-1,1)/diffusion.T],-1))/sigma_t
```

```python
    score_diff = score_true - score_fake

    offset_coeff = denoise_diff / torch.sqrt(denoise_diff.square().sum
    ([1,2,3], keepdims=True) + self.phuber_c**2)
    weight = 1.0

    batch_g_loss = weight * offset_coeff * (fake_denoise - images)
    batch_g_loss = batch_g_loss.sum([1,2,3]).mean()

    optimizer_G.zero_grad()
    batch_g_loss.backward()
    optimizer_G.step()
```

Listing 1: Pytorch Style Pseudo-code of SIM

A.3 Instances of SIM with different distance functions

In section 3.3, we have discussed the powered normed as distance functions. Other choices, such as the Huber distance, which is defined as

$$\forall 1 \leq d \leq D, \;\; L_\delta(\boldsymbol{y})_d \coloneqq \begin{cases} y_d^2/2 & \text{for } y_d \geq \delta \\ \delta(|y_d| - \delta/2) & \text{otherwise} \end{cases}$$

For other choices of distance functions, such as $L1$ norm and exponential with powered norms, we put them in Table 4.

Table 4: Instances of Score Implicit Matching loss with different distance functions. The notations are aligned with the Algorithm 1.

Choice of $\mathbf{d}(.)$	$\mathbf{d}'(\boldsymbol{y}_t)$	Loss function
$\Vert\boldsymbol{y}_t\Vert_2^2$	$2\boldsymbol{y}_t$	$-2\boldsymbol{y}_t^T\Big\{\boldsymbol{s}_\psi(\boldsymbol{x}_t,t) - \nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t\vert\boldsymbol{x}_0)\Big\}$
$\Vert\boldsymbol{y}_t\Vert_\alpha^\alpha,\ \alpha\geq 1,\ \alpha\ even$	$\alpha\boldsymbol{y}_t^{(\alpha-1)}$	$-\alpha\Big\{\boldsymbol{y}_t^{(\alpha-1)}\Big\}^T\Big\{\boldsymbol{s}_\psi(\boldsymbol{x}_t,t) - \nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t\vert\boldsymbol{x}_0)\Big\}$
$\exp(\beta\Vert\boldsymbol{y}_t\Vert_\alpha^\alpha)-1,\ \alpha\geq 1,\ \alpha\ even$	$\alpha\exp(\beta\Vert\boldsymbol{y}_t\Vert_\alpha^\alpha)\boldsymbol{y}_t^{(\alpha-1)}$	$-\alpha\exp(\beta\Vert\boldsymbol{y}_t\Vert_\alpha^\alpha)\Big\{\boldsymbol{y}_t^{(\alpha-1)}\Big\}\Big\{\boldsymbol{s}_\psi(\boldsymbol{x}_t,t) - \nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t\vert\boldsymbol{x}_0)\Big\}$
$\Vert\boldsymbol{y}_t\Vert_1$	$\operatorname{sign}(\boldsymbol{y}_t)$	$-\operatorname{sign}(\boldsymbol{y}_t)^T\Big\{\boldsymbol{s}_\psi(\boldsymbol{x}_t,t) - \nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t\vert\boldsymbol{x}_0)\Big\}$
$L_\delta(\boldsymbol{y}_t)$, $L_\delta(.)$ Huber Loss	$\frac{\partial}{\partial\boldsymbol{y}_t}L_\delta(\boldsymbol{y}_t)$	$-\frac{\partial}{\partial\boldsymbol{y}_t}L_\delta(\boldsymbol{y}_t)^T\Big\{\boldsymbol{s}_\psi(\boldsymbol{x}_t,t) - \nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t\vert\boldsymbol{x}_0)\Big\}$
$\sqrt{\Vert\boldsymbol{y}_t\Vert_2^2 + c^2} - c$	$2\frac{\boldsymbol{y}_t}{\sqrt{\Vert\boldsymbol{y}_t\Vert_2^2+c^2}}$	$-2\Big\{2\frac{\boldsymbol{y}_t}{\sqrt{\Vert\boldsymbol{y}_t\Vert_2^2+c^2}}\Big\}^T\Big\{\boldsymbol{s}_\psi(\boldsymbol{x}_t,t) - \nabla_{\boldsymbol{x}_t}\log q_t(\boldsymbol{x}_t\vert\boldsymbol{x}_0)\Big\}$

B Empirical Parts

B.1 Answer for the human preference study

The answer to the human preference study in Figure 1 is

- the middle image of the first row is generated by one-step SIM-DiT-600M;
- the leftmost image of the second row is generated by one step SIM-DiT-600M;
- the leftmost image of the third row is generated by one-step SIM-DiT-600M.

B.2 Experiment details on CIFAR10 dataset

We follow the experiment setting of SiD and DI on CIFAR10. We start with a brief introduction to the EDM model [21].

The EDM model depends on the diffusion process

$$\mathrm{d}\boldsymbol{x}_t = t\mathrm{d}\boldsymbol{w}_t, t \in [0, T]. \tag{B.1}$$

Samples from the forward process (B.1) can be generated by adding random noise to the output of the generator function, i.e., $\boldsymbol{x}_t = \boldsymbol{x}_0 + t\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is a Gaussian vector. The EDM model also reformulates the diffusion model's score matching objective as a denoising regression objective, which writes,

$$\mathcal{L}(\psi) = \int_{t=0}^{T} \lambda(t)\mathbb{E}_{\boldsymbol{x}_0 \sim p_0, \boldsymbol{x}_t|\boldsymbol{x}_0 \sim p_t(\boldsymbol{x}_t|\boldsymbol{x}_0)} \|\boldsymbol{d}_\psi(\boldsymbol{x}_t, t) - \boldsymbol{x}_0\|_2^2 \mathrm{d}t. \tag{B.2}$$

Where $\boldsymbol{d}_\psi(\cdot)$ is a denoiser network that tries to predict the clean sample by taking noisy samples as inputs. Minimizing the loss (B.2) leads to a trained denoiser, which has a simple relation to the marginal score functions as:

$$\boldsymbol{s}_\psi(\boldsymbol{x}_t, t) = \frac{\boldsymbol{d}_\psi(\boldsymbol{x}_t, t) - \boldsymbol{x}_t}{t^2} \tag{B.3}$$

Under such a formulation, we actually have pre-trained denoiser models for experiments. Therefore, we use the EDM notations in later parts.

Construction of the one-step generator. Let $\boldsymbol{d}_\theta(\cdot)$ be pretrained EDM denoiser models. Owing to the denoiser formulation of the EDM model, we construct the generator to have the same architecture as the pre-trained EDM denoiser with a pre-selected index t^*, which writes

$$\boldsymbol{x}_0 = g_\theta(\boldsymbol{z}) \coloneqq \boldsymbol{d}(\boldsymbol{z}, t^*), \;\; \boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, (t^*)^2\mathbf{I}). \tag{B.4}$$

We initialize the generator with the same parameter as the teacher EDM denoiser model.

Time index distribution. When training both the EDM diffusion model and the generator, we need to randomly select a time t in order to approximate the integral of the loss function (B.2). The EDM model has a default choice of t distribution as log-normal when training the diffusion (denoiser) model, i.e.

$$t \sim p_{EDM}(t): \;\; t = \exp(s) \tag{B.5}$$

$$s \sim \mathcal{N}(P_{mean}, P_{std}^2), \;\; P_{mean} = -1.2, P_{std} = 1.2. \tag{B.6}$$

And a weighting function

$$\lambda_{EDM}(t) = \frac{(t^2 + \sigma_{data}^2)}{(t \times \sigma_{data})^2}. \tag{B.7}$$

In our algorithm, we follow the same setting as the EDM model when updating the online diffusion (denoiser) model.

In SiD, they propose to use a special discrete time distribution, which writes

$$\sigma_k = (\sigma_{max}^{\frac{1}{\rho}} \frac{i}{K-1}(\sigma_{min}^{\frac{1}{\rho}} - \sigma_{max}^{\frac{1}{\rho}}))^\rho,$$

$$\sigma_{max} = 80.0, \sigma_{min} = 0.002, \rho = 7.0, K = 1000$$

They proposed to choose t uniformly from

$$t \sim p_{SiD}(t): \;\; k \sim Unif[0, 800], t = \sigma_k; \tag{B.8}$$

We name such a time distribution the $Karr$ distribution in Figure 2 because such a schedule was originally proposed in Karras' EDM work for sampling.

However, in practice, we find that $Karr$ distribution (B.8) empirically does not work well. Instead, we find that a modified log-normal time distribution when updating the generation with SIM works better than $Karr$ distribution. Our SIM time distribution writes:

$$t \sim p_{SIM}(t): \;\; t = \exp(s) \tag{B.9}$$

$$s \sim \mathcal{N}(P_{mean}, P_{std}^2), \;\; P_{mean} = -3.5, P_{std} = 2.5. \tag{B.10}$$

Table 5: Hyperparameters used for SIM on CIFAR10 EDM Distillation

Hyperparameter	CIFAR-10 (Uncond)		CIFAR-10 (Cond)	
	DM $\boldsymbol{s}_\psi$	Generator g_θ	DM $\boldsymbol{s}_\psi$	Generator g_θ
Learning rate	1e-5	1e-5	1e-5	1e-5
Batch size	256	256	256	256
$\sigma(t^*)$	2.5	2.5	2.5	2.5
$Adam\ \beta_0$	0.0	0.0	0.0	0.0
$Adam\ \beta_1$	0.999	0.999	0.999	0.999
Time Distribution	$p_{EDM}(t)$(B.5)	$p_{SIM}(t)$(B.9)	$p_{EDM}(t)$(B.5)	$p_{SIM}(t)$(B.9)
Weighting	$\lambda_{EDM}(t)$(B.7)	1	$\lambda_{EDM}(t)$(B.7)	1
Loss function	(B.2)	(**??**)	(B.2)	
Number of GPUs	4×A100-40G	4×A100-40G	4×A100-40G	4×A100-40G

Algorithm 2: SIM with Pseudo-Huber distance for distilling EDM teacher [Pytorch Style].

Input: pre-trained EDM denoiser $\boldsymbol{d}_{q_t}(.)$, generator g_θ, prior distribution p_z, online EDM denoiser $\boldsymbol{d}_\psi(.)$; differentiable distance function $\mathbf{d}(.)$, and forward diffusion (2.1).

while *not converge* **do**

// *freeze θ, update ψ:*

$\boldsymbol{x}_0 = g_\theta(\boldsymbol{z}).detach(),\ \ \boldsymbol{z} \sim p_z$

$t \sim p_{EDM}(t),\ \ \boldsymbol{x}_t = \boldsymbol{x}_0 + t\epsilon,\ \ \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$\mathcal{L}(\psi) = \lambda_{EDM}(t) \times \|\boldsymbol{d}_\psi(\boldsymbol{x}_t, t) - \boldsymbol{x}_0\|_2^2$

$\mathcal{L}(\psi).backward()$; update ψ

// *freeze ψ, update θ:*

$\boldsymbol{x}_0 = g_\theta(\boldsymbol{z}),\ \ \boldsymbol{z} \sim p_z$

$t \sim p_{SIM}(t),\ \ \boldsymbol{x}_t = \boldsymbol{x}_0 + t\epsilon,\ \ \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$\mathcal{L}(\theta) = -\left\{\frac{\boldsymbol{y}_t}{\sqrt{\|\boldsymbol{y}_t\|_2^2 + c^2}}\right\}^T \left\{\boldsymbol{d}_\psi(\boldsymbol{x}_t, t) - \boldsymbol{x}_0\right\}$, where $\boldsymbol{y}_t := \boldsymbol{d}_\psi(\boldsymbol{x}_t, t) - \boldsymbol{d}_{q_t}(\boldsymbol{x}_t)$

$\mathcal{L}(\theta).backward()$; update θ

end

return θ, ψ.

Weighting function. As we have said, we use the same $\lambda_{EDM}(t)$ (B.7) weighting function as EDM when updating the denoiser model. When updating the generator, SiD uses a specially designed weighting function, which writes:

$$w_{SiD}(t) = \frac{C \times t^4}{\|\boldsymbol{x}_0 - \boldsymbol{d}_{q_t}(\boldsymbol{x}_t)\|_{1,\mathrm{sg}}} \tag{B.11}$$

$$\boldsymbol{x}_t = \boldsymbol{x}_0 + t\epsilon,\ \ \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{B.12}$$

The notation sg means stop-gradient, and C is the data dimensions. They claim such a weighting function helps to stabilize the training. However, in our experiments, since the SIM itself has normalized the loss (see section 4), we do not use such ad-hoc weighting functions. Instead, we just set the weighting function to be 1 for all time. We call the SiD's weighting function the $sidwgt$ in Figure 2, and our weighting the $nowgt$ in Figure 2.

In Figure 2, we compare the SiD and SIM with different time distribution and weighting functions. We find that SIM+nowgt+lognormal time distribution gives the best performances significantly, therefore our final experiment tasks such a configuration. Table 5 records the detailed configurations we use for SIM on CIFAR10 EDM distillation.

With the optimal setting and EDM formulation, we can rewrite our algorithm in an EDM style in Algorithm 2.

B.3 Experiment details on Text-to-Image Distillation

In the Text-to-Image distillation part, in order to align our experiment with that on CIFAR10, we rewrite the PixArt-α model in EDM formulation:

$$D_\theta(\mathbf{x}; \sigma) = \mathbf{x} - \sigma F_\theta \tag{B.13}$$

Here, following the iDDPM+DDIM preconditioning in EDM, PixArt-α is denoted by F_θ, $\mathbf{x}$ is the image data plus noise with a standard deviation of σ, for the remaining parameters such as C_1 and C_2, we kept the other parameters unchanged to match those defined in EDM. Unlike the original model, we only retained the image channels for the output of this model. Since we employed the preconditioning of iDDPM+DDIM in the EDM, each σ value is rounded to the nearest 1000 bins after being passed into the model. For the actual values used in PixArt-α, beta_start is set to 0.0001, and beta_end is set to 0.02. Therefore, according to the formulation of EDM, the range of our noise distribution is [0.01, 156.6155], which will be used to truncate our sampled σ. For our one-step generator, it is formulated as:

$$g_\theta(\mathbf{x};\sigma_{\text{init}}) = \mathbf{x} - \sigma_{\text{init}} F_\theta \tag{B.14}$$

Here following SiD $\sigma_{\text{init}} = 2.5$ and $\mathbf{x} \sim \mathcal{N}(0, \sigma_{\text{init}}\mathbf{I})$, we observed in practice that larger values of σ_{init} lead to faster convergence of the model, but the difference in convergence speed is negligible for the complete model training process and has minimal impact on the final results.

We utilized the SAM-LLaVA-Caption10M dataset, which comprises prompts generated by the LLaVA model on the SAM dataset. These prompts provide detailed descriptions for the images, thereby offering us a challenging set of samples for our distillation experiments.

All experiments in this section were conducted on 4 A100-40G GPUs with bfloat16 precision, using the PixArt-XL-2-512x512 model version, employing the same hyperparameters. For both optimizers, we utilized Adam with a learning rate of 5e-6 and betas=[0, 0.999]. Additionally, to enable a batch size of 1024, we employed gradient checkpointing and set the gradient accumulation to 8. Finally, regarding the training noise distribution, instead of adhering to the original iDDPM schedule, we sample the σ from a log-normal distribution with a mean of -2.0 and a standard deviation of 2.0, we use the same noise distribution for both optimization step and set the two loss weighting to constant 1. Our best model was trained on the SAM Caption dataset for approximately 16k iterations, which is equivalent to less than 2 epochs. This training process took about 2 days on 4 A100-40G GPUs.

We also tested the impact of different noise distributions on the distillation process. When the noise distribution is highly concentrated around smaller values, we observed a phenomenon where the generated samples appear excessively dark. On the other hand, when we used slightly larger noise distributions, we found that the structure of the generated samples tended to be unstable.

B.4 Instruction for Human Preference Study

Our user study primarily focuses on comparing the outputs of the distilled model and the teacher model. Each image has undergone rigorous manual review to ensure the safety of survey participants. We conducted the study using questionnaires, where users were presented with two randomly ordered images generated by the distilled model and teacher model and asked to select the sample that best matched the text description and had higher image quality. Finally, we used the collected votes for the distilled model and the teacher model as indicators of user preference. The questionnaire website used for conducting these evaluations are shown in Figure 4.

To be more specific, we randomly selected 17 prompt words and generated images of resolution 512x512 using both the student model and the teacher model. To facilitate comparison, we presented the two images side by side in random order. In the questionnaire, we provided the complete prompt words for reference in addition to the generated images. In the end, we collected approximately 30 survey responses in total.

B.5 Generated Samples on CIFAR10

B.6 Prompts for Figure 3

- prompt for first row of Figure 3: *A small cactus with a happy face in the Sahara desert.*
- prompt for second row of Figure 3: *An image of a jade green and gold coloured Fabergé egg, 16k resolution, highly detailed, product photography, trending on artstation, sharp focus, studio photo, intricate details, fairly dark background, perfect lighting, perfect composition, sharp features, Miki Asai Macro photography, close-up, hyper detailed, trending on artstation, sharp focus, studio photo, intricate details, highly detailed, by greg rutkowski.*
- prompt for third row of Figure 3: *Baby playing with toys in the snow.*

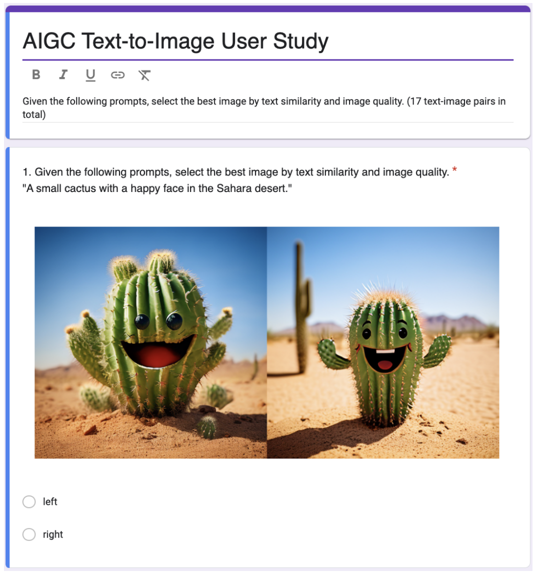

Figure 4: Demonstration of our human preference user study interface.

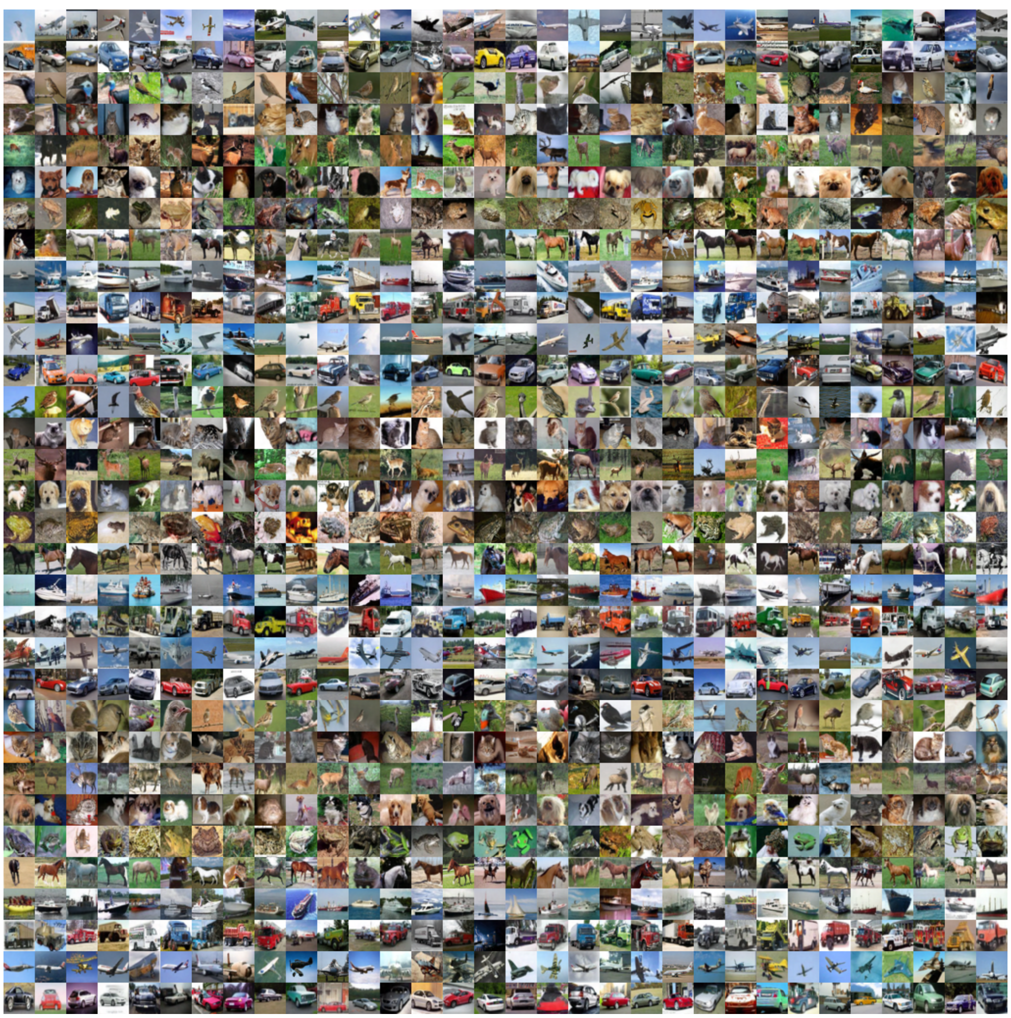

Figure 5: One-step SIM model on CIFAR10-conditional. FID=1.96.

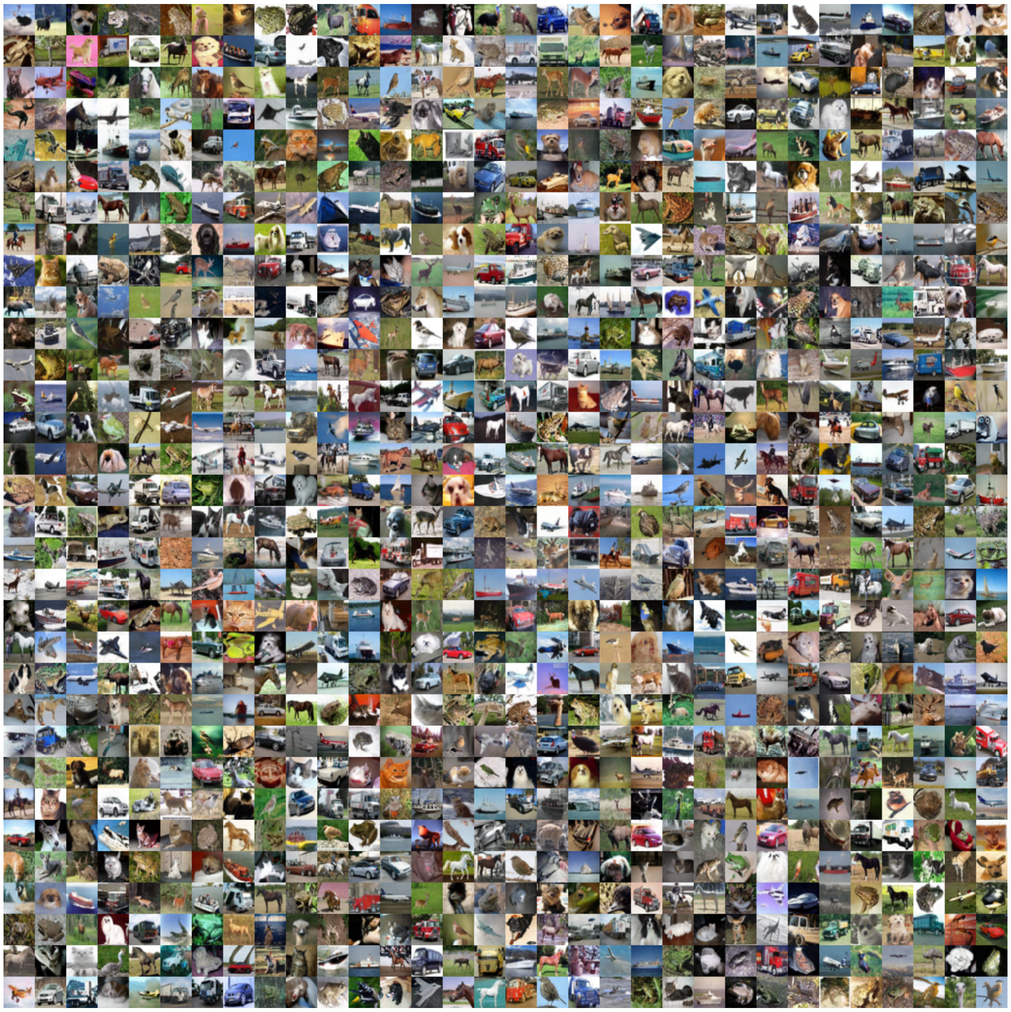

Figure 6: One-step SIM model on CIFAR10-unconditional. FID=2.17.