SD-DiT: Unleashing the Power of Self-supervised Discrimination in Diffusion Transformer – Supplementary Material

Rui Zhu[1], Yingwei Pan[2], Yehao Li[2], Ting Yao[2], Zhenglong Sun[1], Tao Mei[2], Chang Wen Chen[3]
[1] The Chinese University of HongKong, Shenzhen [2] HiDream.ai Inc. [3] The Hong Kong Polytechnic University
ruizhu@link.cuhk.edu.cn, {pandy, liyehao, tiyao}@hidream.ai, sunzhenglong@cuhk.edu.cn
tmei@hidream.ai, changwen.chen@polyu.edu.hk

The supplementary material contains: 1) more implementation details about our proposed Diffusion Transformer with Self-supervised Discrimination (SD-DiT) in Sec. 1; 2) the pseudocode of SD-DiT in Algorithm A1; 3) the qualitative visualization results of SD-DiT-XL/2 in Fig. A1.

1. Implementation Details

In contrast to DiT [4] and MDT [2] whose settings are derived from the ADM formulation [1], our SD-DiT employs the formulation of EDM [3] in order to construct the discriminative pairs according to the theory of the *consistency function* (Eq.(7) in main paper) [6] based on the PF-ODE (Eq. (4) in main paper) of EDM. Specifically, we adopt the EDM preconditioning parameterization by using a σ-dependent skip connection[1]:

$$D_\theta(\boldsymbol{x};\sigma) = \boldsymbol{c}_{\text{skip}}(\sigma)\,\boldsymbol{x} + \boldsymbol{c}_{\text{out}}(\sigma)\,F_\theta\big(\boldsymbol{c}_{\text{in}}(\sigma)\,\boldsymbol{x};\,\boldsymbol{c}_{\text{noise}}(\sigma)\big). \quad (1)$$

This preconditioning parameterization is a common practice to avoid large variation in gradient magnitudes brought by various noise levels. As shown in Eq. (1), the denoiser D_θ is not directly employed as a neural network. Instead, a different network F_θ is trained to learn D_θ. In our SD-DiT, the student branch is wrapped as D_θ in Eq. (1) with skip connection preconditioning. For simplicity, we did not introduce this parameterization in the main paper. We follow the default hyper-parameters of EDM for the skip connection $\boldsymbol{c}_{\text{skip}}(\sigma)$, the noise level $\boldsymbol{c}_{\text{noise}}(\sigma)$ and the input $\boldsymbol{c}_{\text{in}}(\sigma)$ and output magnitudes $\boldsymbol{c}_{\text{out}}(\sigma)$. Besides, the student noise distribution $p_{\sigma_{\text{S}}}$ follows the $p_{\sigma_{\text{train}}}$ in EDM's setting:

$$\ln(p_{\sigma_{\text{S}}}) \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}), \quad (2)$$

where $P_{\text{mean}} = -1.2$ and $P_{\text{std}} = 1.2$. Note that we draw the approximate log-normal probability density distribution (i.e., the black dashed line in Fig. 6 in main paper) of the corresponding σ_{S} according to this Eq. (2). During the sampling stage, we use the default time steps schedule of EDM:

$$\sigma_{i<N} = \big(\sigma_{\max}{}^{\frac{1}{\rho}} + \tfrac{i}{N-1}(\sigma_{\min}{}^{\frac{1}{\rho}} - \sigma_{\max}{}^{\frac{1}{\rho}})\big)^{\rho}, \sigma_N = 0, \quad (3)$$

where sampling steps $N = 40$, $\rho = 7$, $\sigma_{\min} = 0.002$ and $\sigma_{\max} = 80$. Following EDM, we utilize the second-order Heun ODE solver for sampling. We follow the paradigm of LDM [5] to perform diffusion generation in the latent space of the frozen pre-trained VAE model [5], which downsamples a $256 \times 256 \times 3$ image into a $32 \times 32 \times 4$ latent variable. More implementation details can be referred in Tab. A1.

Network parameters. The teacher-student design will double the parameters of a typical DiT. But at inference, the teacher network will be removed, and thus no parameter burden is introduced. In this sense, the model size of learned SD-DiT-XL/2 is 740.6M, which is comparable to MaskDiT-XL/2 (730.1M). During training, the additional teacher network is directly updated by EMA without SGD backward propagation, thereby only requiring extremely lightweight computational cost compared to standard backward propagation. At inference, the teacher network is completely removed and no burden is introduced.

Table A1. Configs for training SD-DiT on 256×256 ImageNet-1K.

Configs	SD-DiT-S/2	SD-DiT-B/2	SD-DiT-XL/2
total batch size		256	
learning rate		1e-4	
training iterations	400k	400k	2400k
optimizer		AdamW with β_1, β_2=$0.9, 0.999$	
EMA momentum		from 0.996 to 0.999	
student temperature		0.1	
teacher temperature		from 0.09 to 0.099 (warmup 5 epochs)	

2. Additional Experimental Results

How about training with a larger batch size? MaskDiT-XL/2 attains the best FID score with fewer training steps, attributed to a large batch size of 1024. For a more comprehensive comparison, we experiment by training SD-DiT-XL/2 with 1024 batch size, and the FID is 16.78 (150k steps), which is better than MaskDiT-XL/2 (FID: 17.22 at 150k steps) [7].

Comparison at higher iterations. We experiment by training SD-DiT-XL/2 with higher iterations (3500k), and the

[1]Please refer to EDM [3] for more comprehensive details.

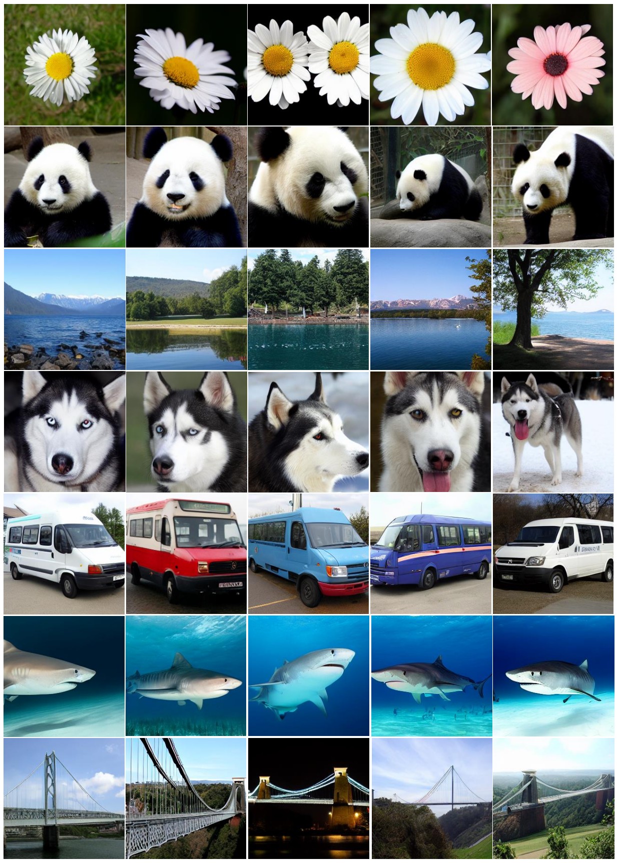

Figure A1. Qualitative results of our SD-DiT-XL/2. Label of each row (from top to bottom): Daisy, Giant panda, Lakeside, Eskimo dog, Minibus, Tiger shark, Suspension bridge.

FID is 6.74, which is comparable to MDT-XL/2 (FID: 6.65, 3500k) [2]. It is worth noting that, compared to , our SD-DiT-XL/2 only uses 45GB memory per GPU with faster training speed (much lower than the memory requirement of MDT-XL/2 [7]), leading to a better computational cost-performance trade-off.

Classifier-free guidance (CFG) results. We also experiment by upgrading our SD-DiT with CFG, and the FID of SD-DiT-XL/2 (+CFG) is 3.23, which is better than MaskDiT-XL/2 (without the unmask tuning stage) with CFG (FID: 4.54) [7].

3. Pytorch-like Pseudocode for SD-DiT

Algorithm A1 Pytorch-like Pseudocode of SD-DiT

```
# S_θ, T'_θ: student & teacher DiT encoder
# G_θ: student DiT decoder
# j_θ, j_θ': student and teacher MLP projection head
# C_cls: center (K) for cls dicrimitive loss
# C_patch: center (K) for patch dicrimitive loss
# τ_S, τ_T: student and teacher temperatures
# β, m_c, m_p: the momentum rates of network, center of
    C_cls, and center of C_patch
# x_0: We extract all the latents from raw image by VAE
    encoder to directly model our SD-DiT on the basis
    of Latent Diffusion Model.

T'_θ.params = S_θ.params
for x_0 in loader: # load a minibatch with N samples

    # construct noised student and teacher views
    x_{σ_S} = x_0 + n_S,  n_S ~ N(0, σ_S^2 I),  σ_S ∈ [σ_min, σ_max]
    x_{σ_T} = x_0 + n_T,  n_T ~ N(0, σ_min^2 I)
    # random mask M for student view
    v_{σ_S} = x_{σ_S} ⊙ (1 − M) # visible patches
    v̄_{σ_S} = x_{σ_S} ⊙ M       # invisible patches
    # forward student and tecaher encoder
    e_S = S_θ(v_{σ_S})  # forward visible student patches
    e_T = T_θ'(x_{σ_T})  # forward full teacher patches

    ######## Generative Loss ########
    # insert invisible patches onto visible tokens
        according to mask positions
    H = torch.gather(torch.cat(e_S,v̄), M)
    # feed complete token set to student decoder
    o_S = G_θ(H) # output all tokens for generative loss
    L_G = MSELoss(o_S, x_0).mean()

    ######## Discriminative Loss ########
    # forward projection head
    j(e_S^[cls]), j(e_S^patch) = j_θ(e_S) #cls token dim: [N,1,K]
    j(e_T^[cls]), j(e_T^patch) = j_θ'(e_T) #patch token dim:[N,L,K]
    # inter-view discriminative loss on CLS token
    L_D^cls= H(j(e_T^[cls]), j(e_S^[cls]), C_cls)
    # inter-view discriminative loss on patch tokens
    L_D^patch= H(j(e_T^patch), j(e_S^patch), C_patch)

    ##############################################
    Loss = L_G + L_D^cls + L_D^cls
    Loss.backward() # back-propagate
    update(θ) # SGD update for student branch

    # teacher and center updates
    θ'.params = β*θ'.params + (1-β)*θ.params
    # center updates by teacher patches and cls token
    C_cls = m_c*C_cls + (1-m_c)*j(e_T^[cls]).mean(dim=0)
    C_patch = m_p*C_patch + (1-m_p)*j(e_T^patch).mean(dim
        =0,1)

def H(T, S, C): # cross-entropy loss
    T = T.detach() # stop gradient
    S = softmax(S/τ_S, dim=1)
    T = softmax((T - C)/τ_T, dim=1) # center + sharpen
    return - (T * log(S)).sum(dim=1).mean()
```

Notes: Note that patch-level discriminative loss is solely performed over the visible patch tokens. Here we do not show them in the pseudocode for simplicity. Moreover, we do not show the skip-connection preconditioning in this pseudocode.

References

[1] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In *NeurIPS*, 2021.

[2] Shanghua Gao, Pan Zhou, Ming-Ming Cheng, and Shuicheng Yan. Masked diffusion transformer is a strong image synthesizer. In *ICCV*, 2023.

[3] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *NeurIPS*, 2022.

[4] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

[5] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[6] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *ICML*, 2023.

[7] Hongkai Zheng, Weili Nie, Arash Vahdat, and Anima Anandkumar. Fast training of diffusion models with masked transformers. *arXiv preprint arXiv:2306.09305*, 2023.