SANA-Sprint: One-Step Diffusion with Continuous-Time Consistency Distillation

Supplementary Material

A. Pseudo Code for Training-Free Transformation from Flow to Trigflow

In this section, we provide a concise implementation of the transformation from a trained flow matching model to a TrigFlow model without requiring additional training. This transformation is based on the theoretical equivalence established in Appendix D. The core idea is to first convert the TrigFlow timestep $t_{\texttt{Trig}}$ to its corresponding flow matching timestep $t_{\texttt{FM}}$. Then, the input feature $\boldsymbol{x}_{\texttt{Trig}}$ is scaled accordingly to obtain $\boldsymbol{x}_{\texttt{FM}}$. The output of the flow matching model is then transformed using a linear combination to produce the final TrigFlow output. The following pseudo code implements this transformation efficiently, ensuring consistency between the two formulations.

```
class TrigFlowModel(FlowMatchingModel):
    def forward(self, x_trig, t_trig, c):
        t_fm = torch.sin(t_trig) / (torch.cos(t_trig) + torch.sin(t_trig))
        x_fm = x_trig * torch.sqrt(t_fm**2 + (1 - t_fm)**2)

        fm_model_out = super().forward(x_fm, t_fm, c)
        trig_model_out = ((1 - 2 * t_fm) * x_fm + (1 - 2 * t_fm + 2 * t_fm**2) *
            fm_model_out) / torch.sqrt(t_fm**2 + (1 - t_fm)**2)

        return trig_model_out
```

B. Transformation Algorithm

We present an algorithm for training-free transformation from a flow matching model to its TrigFlow counterpart. Given a noisy sample, its corresponding TrigFlow timestep, and a pre-trained flow matching model, the algorithm computes the equivalent flow matching timestep, rescales the input, and applies a deterministic transformation to obtain the TrigFlow output. The detailed procedure is outlined in Algorithm 1.

Algorithm 1 Training-Free Transformation to TrigFlow

1: **Input:** Noisy data $\frac{\boldsymbol{x}_{t,\texttt{Trig}}}{\sigma_d}$, timestep $t_{\texttt{Trig}}$, condition $\boldsymbol{y}$, flow matching model $\boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, y)$
2: Compute $t_{\texttt{FM}}$ from $t_{\texttt{Trig}}$ via $t_{\texttt{FM}} = \frac{\sin(t_{\texttt{Trig}})}{\sin(t_{\texttt{Trig}})+\cos(t_{\texttt{Trig}})}$
3: Compute $\boldsymbol{x}_{t,\texttt{FM}}$ from $\boldsymbol{x}_{t,\texttt{Trig}}$ via $x_{t,\texttt{FM}} = \frac{\boldsymbol{x}_{t,\texttt{Trig}}}{\sigma_d} \cdot \sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}$
4: Evaluate $\boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, y)$
5: Transform the model output via $\widehat{\boldsymbol{F}_{\boldsymbol{\theta}}}\left(\frac{\boldsymbol{x}_{t,\texttt{Trig}}}{\sigma_d}, t_{\texttt{Trig}}, \boldsymbol{y}\right) = \frac{1}{\sqrt{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}}\left[(1 - 2t_{\texttt{FM}})\boldsymbol{x}_{t,\texttt{FM}} + (1 - 2t_{\texttt{FM}} + 2t_{\texttt{FM}}^2)\boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y})\right]$
6: **Output:** Transformed result

C. Training Algorithm of SANA-Sprint

In this section, we present the detailed training algorithm for SANA-Sprint. To emphasize the differences from the standard sCM training algorithm, we highlight the modified steps in light blue. The following algorithm outlines the complete training procedure, including the key transformations and parameter updates specific to SANA-Sprint.

D. Proof of Proposition 3.1

Before presenting the formal proof, we first provide some context to understand the necessity of the transformation. In score-based generative models such as diffusion, flow matching, and TrigFlow, denoising is typically performed under certain conditions of data scales and signal-to-noise ratios (SNRs) that match the training setup. However, directly applying flow matching to denoise data generated by TrigFlow is not feasible due to mismatches in time parameterization, SNR, and output

Algorithm 2 Training Algorithm of SANA-Sprint

Input: dataset $\mathcal{D}$ with std. $\sigma_d = 0.5$, pretrained flow model $\boldsymbol{F}_{\text{pretrain}}$ with parameter $\boldsymbol{\theta}_{\text{pretrain}}$, student model $\boldsymbol{F}_\theta$, discriminator head $D_{\boldsymbol{\psi}}$, weighting $w_{\boldsymbol{\phi}}$, learning rate η, generator distribution $(P_{\text{mean, G}}, P_{\text{std, G}})$, discriminator distribution $(P_{\text{mean, D}}, P_{\text{std, D}})$, constant c, warmup iteration H, max-time weighting p, condition $\boldsymbol{y}$.
Init: transform $\boldsymbol{F}_{\text{pretrain}}$ and $\boldsymbol{F}_\theta$ to TrigFlow model using **Algorithm 1**, init student model and discriminator backbone with $\boldsymbol{\theta}_{\text{pretrain}}$, Iters $\leftarrow 0$.
repeat
  update discriminator $\boldsymbol{\psi}$:
    $\boldsymbol{x}_0 \sim \mathcal{D}$, $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \sigma_d^2 \boldsymbol{I})$, $\tau \sim \mathcal{N}(P_{\text{mean, G}}, P_{\text{std, G}}^2)$, $t \leftarrow \arctan(\frac{e^\tau}{\sigma_d})$
    if $p > 0$, $\xi \sim \mathrm{U}[0,1]$, $t \leftarrow \frac{\pi}{2}$ if $\xi < p$
    $\boldsymbol{x}_t \leftarrow \cos(t)\boldsymbol{x}_0 + \sin(t)\boldsymbol{z}$, $\hat{\boldsymbol{x}}_0^{\boldsymbol{f}_{\boldsymbol{\theta}^-}} \leftarrow \boldsymbol{f}_{\boldsymbol{\theta}^-}(\boldsymbol{x}_t, t, \boldsymbol{y})$
    $\tau \sim \mathcal{N}(P_{\text{mean, D}}, P_{\text{std, D}}^2)$, $s \leftarrow \arctan(\frac{e^\tau}{\sigma_d})$
    $\boldsymbol{x}_s \leftarrow \cos(s)\boldsymbol{x}_0 + \sin(s)\boldsymbol{z}$, $\hat{\boldsymbol{x}}_s^{\boldsymbol{f}_{\boldsymbol{\theta}^-}} \leftarrow \cos(s)\hat{\boldsymbol{x}}_0^{\boldsymbol{f}_{\boldsymbol{\theta}^-}} + \sin(s)\boldsymbol{z}$
    $\mathcal{L}_{\text{adv}}^D(\boldsymbol{\psi}) \leftarrow \mathbb{E}_{\boldsymbol{x}_0,s}\Big[\sum_k \mathrm{ReLU}\Big(1 - D_{\boldsymbol{\psi},k}(\boldsymbol{F}_{\boldsymbol{\theta}_{\text{pre}},k}(\boldsymbol{x}_s, s, \boldsymbol{y}))\Big)\Big] + \mathbb{E}_{\boldsymbol{x}_0,s,t}\Big[\sum_k \mathrm{ReLU}\Big(1 + D_{\boldsymbol{\psi},k}(\boldsymbol{F}_{\boldsymbol{\theta}_{\text{pre}},k}(\hat{\boldsymbol{x}}_s^{\boldsymbol{f}_{\boldsymbol{\theta}^-}}, s, \boldsymbol{y}))\Big)\Big]$
    $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \eta \nabla_{\boldsymbol{\psi}} \mathcal{L}_{\text{adv}}^D(\boldsymbol{\psi})$ ▷ Discriminator step
    Iters $\leftarrow$ Iters $+ 1$
  update student model $\boldsymbol{\theta}$ and weighting $\boldsymbol{\phi}$:
    $\boldsymbol{x}_0 \sim \mathcal{D}$, $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \sigma_d^2 \boldsymbol{I})$, $\tau \sim \mathcal{N}(P_{\text{mean, G}}, P_{\text{std, G}}^2)$, $t \leftarrow \arctan(\frac{e^\tau}{\sigma_d})$
    $\boldsymbol{x}_t \leftarrow \cos(t)\boldsymbol{x}_0 + \sin(t)\boldsymbol{z}$
    $\frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}t} \leftarrow \sigma_d \boldsymbol{F}_{\text{pretrain, cfg}}(\frac{\boldsymbol{x}_t}{\sigma_d}, t)$
    $r \leftarrow \min(1, \text{Iters}/H)$ ▷ Tangent warmup
    $\boldsymbol{g} \leftarrow -\cos^2(t)(\sigma_d \boldsymbol{F}_{\boldsymbol{\theta}^-} - \frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}t}) - r \cdot \cos(t)\sin(t)(\boldsymbol{x}_t + \sigma_d \frac{\mathrm{d}\boldsymbol{F}_{\boldsymbol{\theta}^-}}{\mathrm{d}t})$ ▷ JVP rearrangement
    $\boldsymbol{g} \leftarrow \boldsymbol{g}/(\|\boldsymbol{g}\| + c)$ ▷ Tangent normalization
    $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow \frac{e^{w_{\boldsymbol{\phi}}(t)}}{D}\|\boldsymbol{F}_{\boldsymbol{\theta}}(\frac{\boldsymbol{x}_t}{\sigma_d}, t) - \boldsymbol{F}_{\boldsymbol{\theta}^-}(\frac{\boldsymbol{x}_t}{\sigma_d}, t) - \boldsymbol{g}\|_2^2 - w_{\boldsymbol{\phi}}(t)$ ▷ sCM loss
    if $p > 0$, $\xi \sim \mathrm{U}[0,1]$, $t \leftarrow \frac{\pi}{2}$ if $\xi < p$
    $\boldsymbol{x}_t \leftarrow \cos(t)\boldsymbol{x}_0 + \sin(t)\boldsymbol{z}$, $\hat{\boldsymbol{x}}_0^{\boldsymbol{f}_{\boldsymbol{\theta}}} \leftarrow \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, \boldsymbol{y})$
    $\hat{\boldsymbol{x}}_s^{\boldsymbol{f}_{\boldsymbol{\theta}}} \leftarrow \cos(s)\hat{\boldsymbol{x}}_0^{\boldsymbol{f}_{\boldsymbol{\theta}}} + \sin(s)\boldsymbol{z}$
    $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) - \mathbb{E}_{\boldsymbol{x}_0,s,t}\Big[\sum_k D_{\boldsymbol{\psi},k}(\boldsymbol{F}_{\boldsymbol{\theta}_{\text{pre}},k}(\hat{\boldsymbol{x}}_s^{\boldsymbol{f}_{\boldsymbol{\theta}}}, s, \boldsymbol{y}))\Big]$ ▷ GAN loss
    $(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\phi}) - \eta \nabla_{\theta,\phi} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$ ▷ Generator step
    Iters $\leftarrow$ Iters $+ 1$
until convergence

necessitating explicit transformations to align the input and output between the two models. The following proof provides the explicit transformation required to connect the TrigFlow-scheduled data to the flow-matching framework.

Proof. Under the TrigFlow framework, the noisy input sample is given by

$$\frac{\boldsymbol{x}_{t,\texttt{Trig}}}{\sigma_d} = \cos(t_{\texttt{Trig}})\frac{\boldsymbol{x}_0}{\sigma_d} + \sin(t_{\texttt{Trig}})\frac{\boldsymbol{z}}{\sigma_d}. \tag{13}$$

Since both $\boldsymbol{x}_0$ and $\boldsymbol{z}$ originally have a standard deviation of σ_d, we absorb σ_d into these variables so that they are normalized to have a standard deviation of 1. This normalization aligns with the conventions used in flow matching models. Note that the signal-to-noise ratios (SNRs) for flow matching models and TrigFlow models are given by

$$\text{SNR}(t_{\texttt{FM}}) = (\frac{1 - t_{\texttt{FM}}}{t_{\texttt{FM}}})^2, \quad \text{SNR}(t_{\texttt{Trig}}) = (\frac{\cos(t_{\texttt{Trig}})}{\sin(t_{\texttt{Trig}})})^2 = (\frac{1}{\tan(t_{\texttt{Trig}})})^2. \tag{14}$$

To ensure an equivalent SNR under the flow matching framework, we seek the corresponding time $t_{\texttt{FM}}$ that satisfies:

$$(\frac{1 - t_{\texttt{FM}}}{t_{\texttt{FM}}})^2 = (\frac{1}{\tan(t_{\texttt{Trig}})})^2. \tag{15}$$

Solving this equation, we obtain the relationship between $t_{\texttt{FM}}$ and $t_{\texttt{Trig}}$:

$$t_{\texttt{FM}} = \frac{\sin(t_{\texttt{Trig}})}{\sin(t_{\texttt{Trig}}) + \cos(t_{\texttt{Trig}})}, \quad t_{Trig} = \arctan\left(\frac{t_{FM}}{1 - t_{FM}}\right). \tag{16}$$

Under this transformation, the SNRs of $\boldsymbol{x}_{t,\texttt{FM}}$ and $\boldsymbol{x}_{t,\texttt{Trig}}$ remain equal; however, their scales differ due to the following formulations:

$$\boldsymbol{x}_{t,\texttt{FM}} = (1 - t_{\texttt{FM}})\boldsymbol{x}_0 + t_{\texttt{FM}}\boldsymbol{z}, \quad \boldsymbol{x}_{t,\texttt{Trig}} = \cos(t_{\texttt{Trig}})\boldsymbol{x}_0 + \sin(t_{\texttt{Trig}})\boldsymbol{z}, \tag{17}$$

Since $(1 - t_{\texttt{FM}})^2 + t_{\texttt{FM}}^2$ is generally not equal to $\cos^2(t_{\texttt{Trig}}) + \sin^2(t_{\texttt{Trig}}) = 1$ (except when $t_{\texttt{FM}} = 0$ or 1), a scale adjustment is needed. To align their scales, we introduce a scale factor function $\lambda(t_{\texttt{FM}})$ that satisfies

$$\lambda(t_{\texttt{FM}}) \cdot \cos(t_{\texttt{Trig}}) = (1 - t_{\texttt{FM}}), \text{and } \lambda(t_{\texttt{FM}}) \cdot \sin(t_{\texttt{Trig}}) = t_{\texttt{FM}}, \tag{18}$$

Therefore, the scale factor is determined as follows

$$\lambda(t_{\texttt{FM}}) = \frac{1 - t_{\texttt{FM}}}{\cos(\arctan(\frac{t_{\texttt{FM}}}{1-t_{\texttt{FM}}}))} = \frac{t_{\texttt{FM}}}{\sin(\arctan(\frac{t_{\texttt{FM}}}{1-t_{\texttt{FM}}}))} = \sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}. \tag{19}$$

The transformed $\boldsymbol{x}_{t,\texttt{Trig}}$ follows the same distribution as the flow matching model's training distribution, achieving our desired objective. Next, we aim to determine the optimal estimator for the TrigFlow model $\boldsymbol{F}_{\boldsymbol{\theta}}$, given $\boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y})$. We first consider an ideal scenario where the model's capacity is sufficiently large. In this case, the flow matching model reaches its optimal solution:

$$\boldsymbol{v}^*(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y}) = \mathbb{E}[\boldsymbol{z} - \boldsymbol{x}_0 | \boldsymbol{x}_{t_{\texttt{FM}}}, \boldsymbol{y}], \tag{20}$$

as the conditional expectation minimizes the mean squared error (MSE) loss. Similarly, the optimal solution of the TrigFlow model is given by

$$\boldsymbol{F}^*(\boldsymbol{x}_{t,\texttt{Trig}}, t_{\texttt{Trig}}, \boldsymbol{y}) = \mathbb{E}[\cos(t_{\texttt{Trig}})\boldsymbol{z} - \sin(t_{\texttt{Trig}})\boldsymbol{x}_0 | \boldsymbol{x}_{t_{\texttt{Trig}}}, \boldsymbol{y}]. \tag{21}$$

Noting that

$$\cos(t_{\texttt{Trig}}) = \frac{1 - t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}}, \quad \sin(t_{\texttt{Trig}}) = \frac{t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}}, \tag{22}$$

we leverage the linearity of conditional expectation to derive

$$\begin{aligned}
&\frac{1 - 2t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \cdot \boldsymbol{x}_{t_{\texttt{FM}}} + \frac{1 - 2t_{\texttt{FM}} + 2t_{\texttt{FM}}^2}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \mathbb{E}[\boldsymbol{z} - \boldsymbol{x}_0 | \boldsymbol{x}_{t_{\texttt{FM}}}, \boldsymbol{y}] \\
=&\frac{1 - 2t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \mathbb{E}[(1 - t_{\texttt{FM}}) \cdot \boldsymbol{x}_0 + t_{\texttt{FM}} \cdot \boldsymbol{z} | \boldsymbol{x}_{t_{\texttt{FM}}}, \boldsymbol{y}] + \frac{1 - 2t_{\texttt{FM}} + 2t_{\texttt{FM}}^2}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \mathbb{E}[\boldsymbol{z} - \boldsymbol{x}_0 | \boldsymbol{x}_{t_{\texttt{FM}}}, \boldsymbol{y}] \\
=&\mathbb{E}[\frac{1 - t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \boldsymbol{z} - \frac{t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \boldsymbol{x}_0 | \boldsymbol{x}_{t_{\texttt{FM}}}, \boldsymbol{y}] \\
=&\mathbb{E}[\cos(t_{\texttt{Trig}})\boldsymbol{z} - \sin(t_{\texttt{Trig}})\boldsymbol{x}_0 | x_{t_{\texttt{Trig}}}, \boldsymbol{y}].
\end{aligned} \tag{23}$$

Consequently, we obtain

$$\boldsymbol{F}^*(\boldsymbol{x}_{t,\texttt{Trig}}, t_{\texttt{Trig}}, \boldsymbol{y}) = \frac{1}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \Big[(1 - 2t_{\texttt{FM}})\boldsymbol{x}_{t,\texttt{FM}} + (1 - 2t_{\texttt{FM}} + 2t_{\texttt{FM}}^2)\boldsymbol{v}^*(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y})\Big]. \tag{24}$$

Next, we consider a more realistic scenario where the model's capacity is limited, leading to the learned velocity field

$$\boldsymbol{v}_{\boldsymbol{\theta}^*}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y}) = \min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{z}, t}[w(t) \| \boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y}) - (\boldsymbol{z} - \boldsymbol{x}_0) \|^2]. \tag{25}$$

Under our parameterization, training the TrigFlow model amounts to minimizing

$$\begin{aligned}
&\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{z}, t} \left[\left\| \frac{1 - 2t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \cdot \boldsymbol{x}_{t_{\texttt{FM}}} + \frac{1 - 2t_{\texttt{FM}} + 2t_{\texttt{FM}}^2}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y}) - (\cos(t_{\texttt{Trig}})\boldsymbol{z} - \sin(t_{\texttt{Trig}})\boldsymbol{x}_0) \right\|^2 \right] \\
=&\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{z}, t} \left[\left\| \frac{1 - 2t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \cdot \boldsymbol{x}_{t_{\texttt{FM}}} + \frac{1 - 2t_{\texttt{FM}} + 2t_{\texttt{FM}}^2}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y}) - \left(\frac{1 - t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \boldsymbol{z} - \frac{t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2 + (1 - t_{\texttt{FM}})^2}} \boldsymbol{x}_0 \right) \right\|^2 \right]
\end{aligned} \tag{26}$$

Substituting $\boldsymbol{x}_{t,\texttt{FM}} = (1-t_{\texttt{FM}})\boldsymbol{x}_0 + t_{\texttt{FM}}\boldsymbol{z}$, the above expression simplifies to

$$
\begin{aligned}
&\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_0,\boldsymbol{z},t}\left[\left\| \frac{1-2t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}}\cdot \boldsymbol{x}_{t_{\texttt{FM}}} + \frac{1-2t_{\texttt{FM}}+2t_{\texttt{FM}}^2}{\sqrt{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}}\boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y}) - \left(\frac{1-t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}}\boldsymbol{z} - \frac{t_{\texttt{FM}}}{\sqrt{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}}\boldsymbol{x}_0\right)\right\|^2\right] \\
=&\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_0,\boldsymbol{z},t}\left[\left\| \frac{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}{\sqrt{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}}\boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y}) - \left(\frac{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}{\sqrt{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}}\boldsymbol{z} - \frac{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}{\sqrt{t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2}}\boldsymbol{x}_0\right)\right\|^2\right] \\
=&\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_0,\boldsymbol{z},t}\left[\left(t_{\texttt{FM}}^2+(1-t_{\texttt{FM}})^2\right)\left\|\boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,\texttt{FM}}, t_{\texttt{FM}}, \boldsymbol{y}) - (\boldsymbol{z}-\boldsymbol{x}_0)\right\|^2\right]
\end{aligned}
\tag{27}
$$

Thus, training the TrigFlow model with our parameterization is equivalent to training the flow matching model, apart from differences in the loss weighting function $w(t)$ and the timestep sampling distribution $p(t)$.

□

E. Full Related Work

Text to Image Generation Text-to-image generation has experienced transformative advancements in both efficiency and model design. The field gained early traction with Stable Diffusion [48], which set the stage for scalable high-resolution synthesis. A pivotal shift occurred with Diffusion Transformers (DiT)[42], which replaced conventional U-Net architectures with transformer-based designs, unlocking improved scalability and computational efficiency. Building on this innovation, PixArt-α[6] demonstrated competitive image quality while slashing training costs to only 10.8% of those required by Stable Diffusion v1.5 [48]. Recent breakthroughs have further pushed the boundaries of compositional generation. Large-scale models like FLUX [23] and Stable Diffusion 3 [10] have scaled up to ultra-high-resolution synthesis and introduced multi-modal capabilities through frameworks such as the Multi-modal Diffusion Transformer (MM-DiT)[9]. Playground v3[30] achieved state-of-the-art image quality by seamlessly integrating diffusion models with Large Language Models (LLMs)[8], while PixArt-Σ[5] showcased direct 4K image generation using a compact 0.6B parameter model, emphasizing computational efficiency alongside high-quality outputs. Efficiency-driven innovations have also gained momentum. SANA [63] introduced high-resolution synthesis capabilities through deep compression autoencoding [4] and linear attention mechanisms, enabling deployment on consumer-grade hardware like laptop GPUs. Additionally, advancements in linear attention mechanisms for class-conditional generation [3, 77], diffusion models without attention [57, 68], and cascade structures [44, 47, 58] have further optimized computational requirements while maintaining performance. These developments collectively underscore the field's rapid evolution toward more accessible, efficient, and versatile text-to-image generation technologies.

Diffusion Model Step Distillations Current methodologies primarily coalesce into two dominant paradigms: (1) trajectory-based distillation. Direct Distillation [35] directly learns noise-image mapping given by PF-ODE. Progressive Distillation [39, 49] makes the learning progress easier by progressively enlarging subintervals on the ODE trajectory. Consistency Models (CMs) [54] (e.g. LCM [36], CTM [20], MCM [13], PCM [60], sCM [34]) predict the solution $\boldsymbol{x}_0$ of the PF-ODE given $\boldsymbol{x}_t$ via self-consistency. (2) distribution-based distillation. It can be further divided into GAN [12]-based distillation and its variational score distillation (VSD) variants [37, 46, 50, 62, 66]. ADD [52] explored distilling diffusion models using adversarial training with pretrained feature extractor like DINOv2 [41] in pixel space. LADD [51] further utilize teacher diffusion models as feature extractors enabling direct discrimination in latent space, drastically saving the computation and GPU memories. [70] stabilize VSD with regression loss. SID [75] and SIM [38] propose improved algorithms for VSD.

Real-Time Image Generation Recent advancements in real-time image generation have focused on improving the efficiency and quality of diffusion models. PaGoDA [21] introduces a progressive approach for one-step generation across resolutions. Imagine-Flash also uses a backward distillation to accelerate diffusion inference. In model compression, BitsFusion [55] quantizes Stable Diffusion's UNet to 1.99 bits, and Weight Dilation [32] presents DilateQuant for enhanced performance. For mobile applications, MobileDiffusion [74] achieves sub-second generation times, with SnapFusion [26] and SnapGen [16] enabling 1024x1024 pixel image generation in about 1.4 seconds. SVDQuant [25] introduces 4-bit quantization for diffusion models, and when combined with SANA [63], enables fast generation of high-quality images on consumer GPUs, bridging the gap between model performance and real-time applications.

F. More Details

F.1. Experimental Setup

Model Architecture Following the pruning technology in SANA-1.5 [64], our teacher models are fine-tuned from SANA 0.6B and 1.6B, respectively. The architecture, training data, and other hyperparameters remain consistent with SANA-1.5 [64].

Training Details We conduct distributed training using PyTorch's Distributed Data Parallel (DDP) across 32 NVIDIA A100 GPUs on 4 DGX nodes. Our two-phase strategy involves fine-tuning the teacher model with dense time embedding and QK normalization at a learning rate of 2e-5 for 5,000 iterations (global batch size of 1,024), as discussed in Sec. 3.1. Then, we perform timestep distillation through the proposed framework at a learning rate of 2e-6 with a global batch size of 512 for 20,000 iterations. As Flash Attention JVP kernel support is not available in PyTorch [34], we retain Linear Attention [63] to auto-compute the JVP.

Evaluation Protocol We use multiple metrics: FID, CLIP Score, and GenEval [11], comparing with state-of-the-art methods. FID and CLIP Score are evaluated on the MJHQ-30K dataset [24]. GenEval measures text-image alignment with 553 test prompts, emphasizing its ability to reflect alignment and show improvement potential. We also provide visualizations to compare state-of-the-art methods and highlight our performance.

F.2. More Ablations

Inference Timestep Search Fig. 6 illustrates the process of timestep optimization for inference across 1, 2, and 4 steps, comparing the performance of 0.6B and 1.6B models in terms of FID (top row) and CLIP-Scores (bottom row). The optimization follows a sequential search strategy: first, we determine the optimal t_{max} for 1-step inference using $\arctan(n/0.5)$, inspired by EDM [18], where n is searched for the maximum timestep. Using this t_{max}, we then search for the intermediate timestep t_{2nd} in 2-step inference. For 4-step inference, the timesteps for the first two steps are fixed to their previously optimized values, while the third (t_{3rd}) and fourth timesteps (t_{4th}) are searched sequentially. In each case, the x-axis represents the timestep being optimized at the current step, ensuring that earlier steps use their best-found values to maximize overall performance. This hierarchical approach enables efficient timestep selection for multi-step inference settings.

Controlling the sCM Noise Distribution In the sCM-only experiments, we investigate the impact of different noise distribution parameter settings on model performance. The noise distribution is defined as $t = \arctan\left(\frac{e^{\tau}}{\sigma_d}\right)$, where $\tau \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$. Starting from the initial parameters $(-0.8, 1.6)$ proposed in sCM [34], we experiment with various mean and standard deviation configurations to evaluate their effects. By tracking FID and CLIP-Score trends over 40k training iterations, we identify $(P_{\text{mean}}, P_{\text{std}}) = (0.0, 1.6)$, represented by the green curve in Fig. 7, as the optimal setting. This configuration consistently reduces FID while improving CLIP-Score, resulting in superior generation quality and text-image alignment. We also observe that extreme mean values, such as $P_{\text{mean}} = 0.6$ or $P_{\text{mean}} = -0.8$, lead to significant training instability and even failure in some cases. Consequently, we adopt $(0.0, 1.6)$ as the default parameter setting.

Controlling the LADD Discriminator Noise Distribution Generative features change with the noise level, offering structured feedback at high noise and texture-related feedback at low noise [51]. We compare the results for different mean and standard deviation settings in $t = \arctan\left(\frac{e^{\tau}}{\sigma_d}\right)$, where $\tau \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$ for LADD discriminator. Building on the optimal mean and standard deviation settings $(0.0, 1.6)$ identified for sCM, we further explore the best noise configuration for the LADD's discriminator. In Fig. 8 and Tab. 8, we visualize the probability distributions of t sampled under different mean and standard deviation settings, as well as the corresponding FID and CLIP-Score results when applied in the LADD loss. Based on these analyses, we identify $(-0.6, 1.0)$ as the optimal setting, which achieves a more balanced feature distribution across high and low noise levels while maintaining stable training dynamics. Consequently, we adopt $(-0.6, 1.0)$ as the default configuration for LADD loss.

F.3. More Qualitative Results

SANA-Sprint-ControlNet Visualization Images In Fig. 9, we demonstrate the visualization capabilities of our SANA-Sprint-ControlNet, which efficiently achieves impressive results in only **0.4 seconds** using a **2-step** generation process, producing high-quality images at a resolution of 1024 $\times$ 1024 pixels. The visualization process begins with an input image,

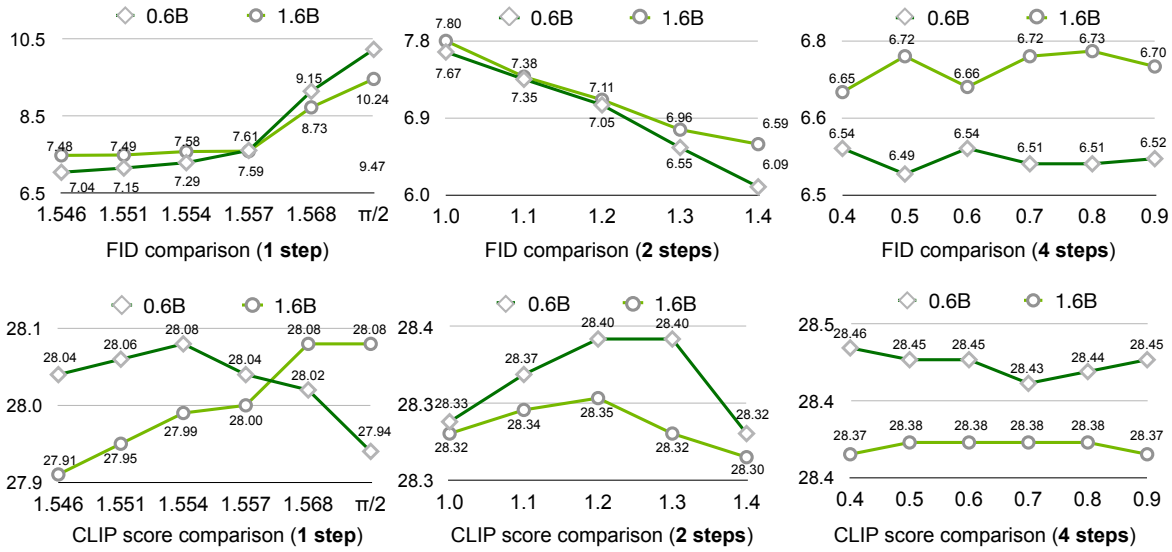

Figure 6. **Inference timesteps search.** This figure illustrates the performance of timesteps search for achieving optimal results during inference with 0.6B and 1.6B models. The subplots compare FID (top row) and CLIP-Score (bottom row) across different timesteps for 1-step, 2-step, and 4-step inference settings. The x-axis represents the timestep being searched at the current step; for multi-step settings (e.g., 4 steps), the timesteps for earlier steps are fixed to their previously optimized values.

Table 7. Inference timestep settings for both SANA-Sprint 0.6B and 1.6B models.

	1 step	2 steps	4 steps
Timestep T	$[\pi/2, 0.0]$	$[\arctan(200/0.5), 1.3, 0.0]$	$[\arctan(200/0.5), 1.3, 1.1, 0.6, 0.0]$

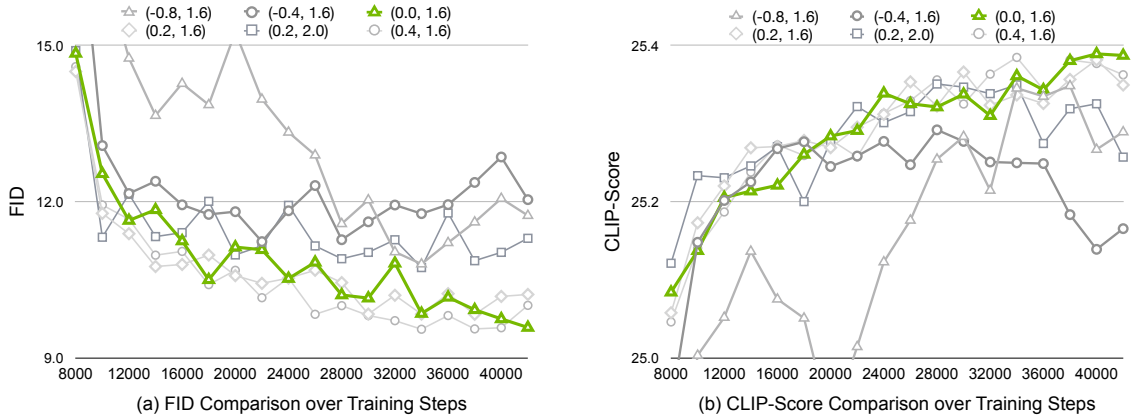

Figure 7. **Controlling the sCM noise distribution.** This figure compares FID and CLIP-Score across different noise distribution settings over 40k training steps in sCM-only experiments. The green curve $(P_{\text{mean}}, P_{\text{std}}) = (0.0, 1.6)$ demonstrates optimal performance, achieving stable training dynamics and superior generation quality.

which is processed using a HED detection model to extract the scribe graph. This scribe graph, combined with a given prompt, is used to generate the corresponding image in the second column. The third column presents a blended image that combines the generated image with the scribe graph, highlighting the precise control of the model through boundary alignment. This visualization showcases the model's ability to accurately interpret prompts and maintain robust control over generated images.

More Visualization Images In Fig. 11, we present images generated by our model using various prompts. SANA-Sprint showcases comprehensive generation capabilities, including high-fidelity detail rendering, accurate semantic understanding, and reliable text generation, all achieved with only **2-step sampling**. In particular, the model efficiently produces high-quality images of 1024×1024 pixels in only **0.24 seconds** on an NVIDIA A100 GPU. The samples demonstrate versatility in various

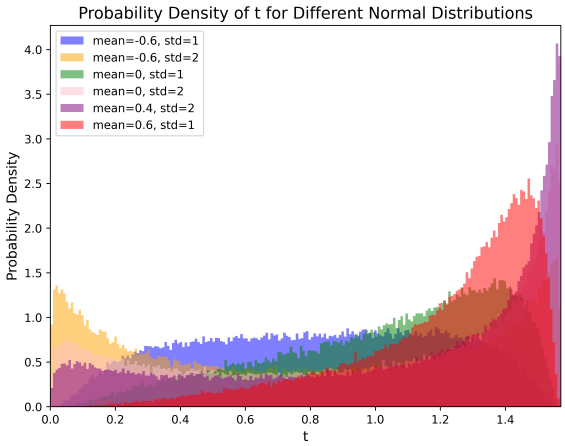

Figure 8. **Controlling the LADD noise distribution.** We vary the parameters of a logit-normal distribution for biasing the sampling of the LADD teacher noise level. When biasing towards very high noise levels (m = 0.4, s = 2), we observe unstable training.

Table 8. Comparison of different noise distributions for LADD loss.

Mean, Std	FID ↓	CLIP ↑
(-0.6, 1.0)	9.48	28.08
(-0.6, 2.0)	10.36	28.03
(0.0, 1.0)	13.11	27.18
(0.0, 2.0)	11.25	27.96
(0.4, 2.0)	9.77	28.00
(0.6, 1.0)	12.85	27.32

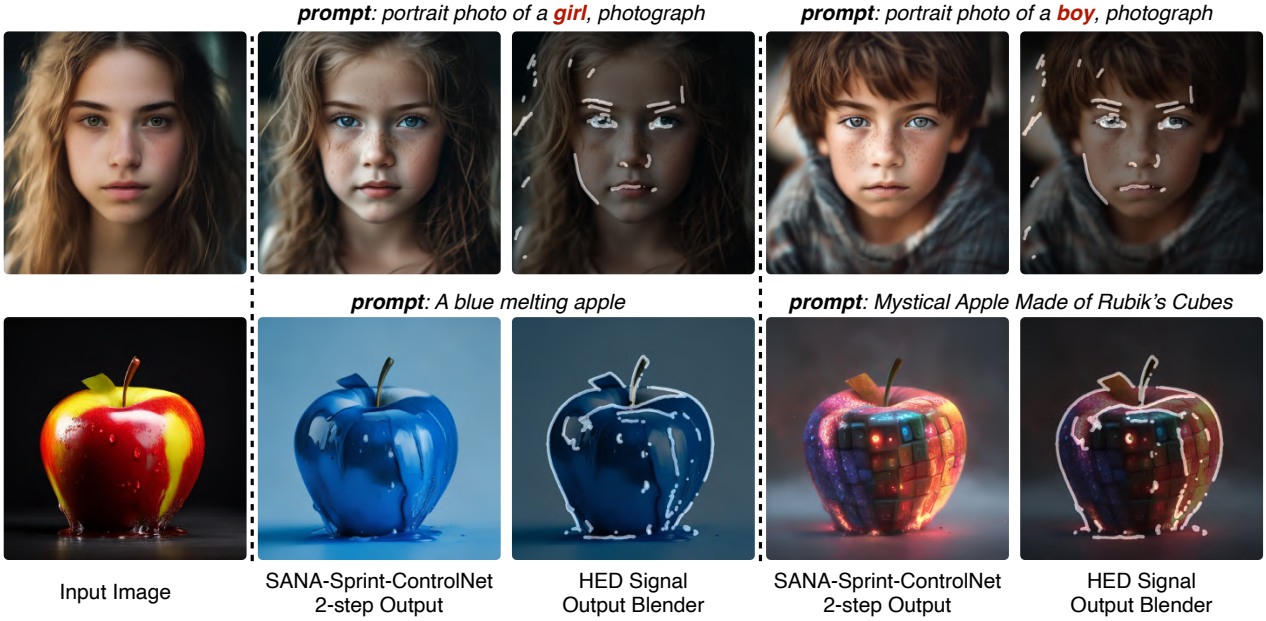

Figure 9. **Visualization of SANA-Sprint-ControlNet's capabilities.** The model outputs high-quality images of 1024 × 1024 pixels in only **2 steps** and **0.3 seconds** on an NVIDIA H100 GPU. The process involves processing the input image (first column) to extract a scribe graph, which, along with a prompt, generates an image (second column). The blended image (third column) highlights precise boundary alignment and control, demonstrating the model's robust control capabilities.

scenarios, from in tricate textures and complex compositions to accurate text rendering, highlighting the robust image quality of the model in both artistic and practical tasks.

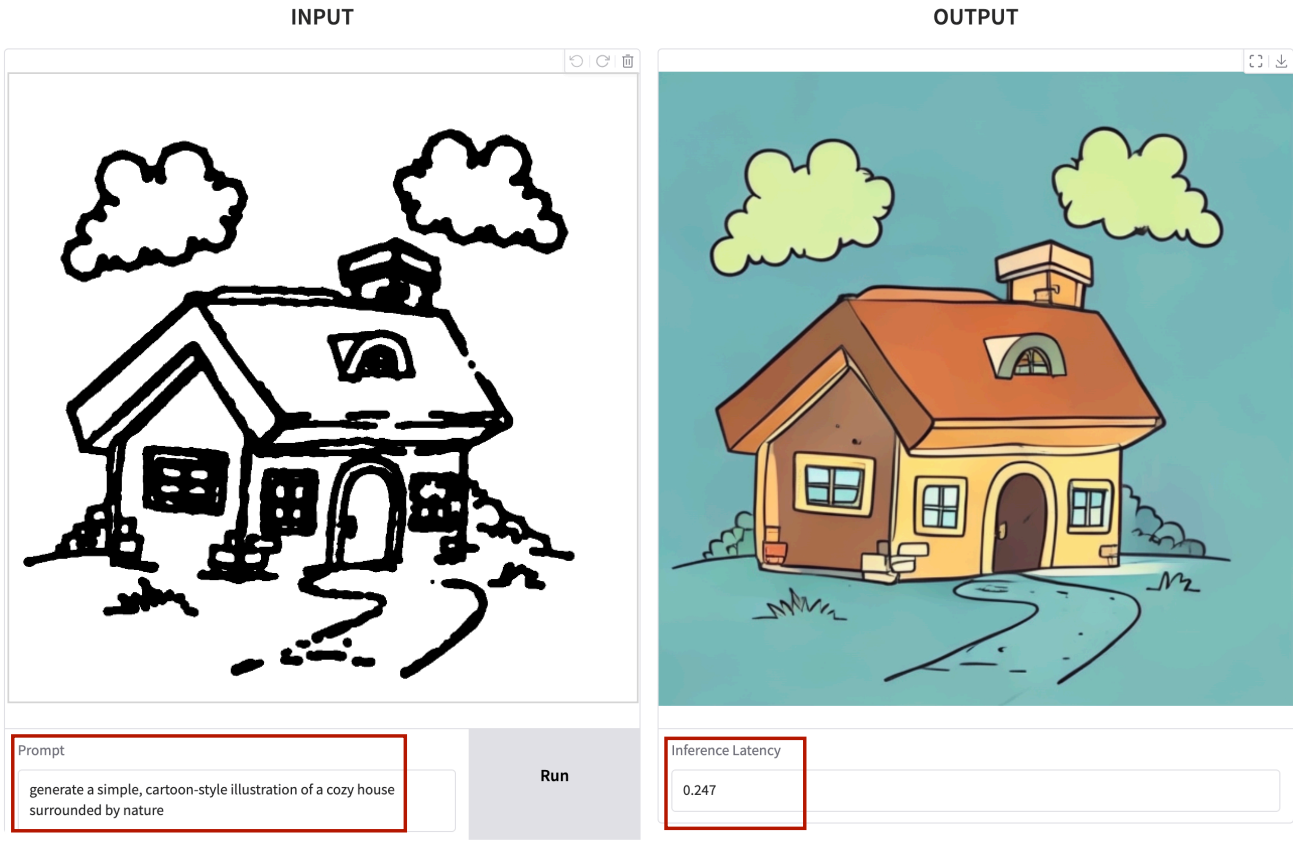

Figure 10. **ControlNet Demo: Hand-Crafted Scribble to Stunning Image. Left**: A hand-crafted scribble created with a brush. **Right**: The result generated by the Sana-Sprint-ControlNet model, strictly following the scribble and prompt. **Inference Latency**: The model achieves remarkable speed, generating the 1024×1024 images in only **1 step** and **0.25 seconds** on H100 GPU, as shown in the right red box. This demo showcases the model's exceptional control and efficiency, adhering closely to the user's input while producing visually appealing results.

Figure 11. **Generated images with SANA-Sprint.** The model outputs high-quality images of 1024 $\times$ 1024 pixels in **2 steps** and **0.24 seconds** on an NVIDIA A100 GPU, showcasing comprehensive generation capabilities with high-fidelity details and accurate text rendering, handling diverse scenarios with robust image quality.