Supplementary Material

Due to space limitations in the Main Text (MT), we include additional ablation and comparison experiments, derivations, benchmark details, implementations, and discussions in the supplementary material. We first present additional ablation studies (Sec. 1) and comparison experiments (Sec. 2). Then, we logically derive the conversion relationships among $\boldsymbol{v}$, $\boldsymbol{x_0}$ and $\boldsymbol{\epsilon}$ (Sec. 3) and provide the Text-LiDAR benchmark processing (Sec. 4). Next, the implementation and optimization details (Sec. 5) of T2LDM are presented. Finally, we discuss the limitations/future works (Sec. 6) and visualize additional results (Sec. 7).

1. Additional Ablation Study

1.1. Reconstruction Loss of SCRG

As discussed in Sec. 4.2 of MT, SCRG can employ arbitrary reconstruction losses to align the noise feature $F^{v_\theta}_{noise}$ from DN with the reconstructed feature $F^{x_\phi}_{recon}$ from GN, enhancing the geometric detail perception of $F^{v_\theta}_{noise}$. In Tab. 1, cosine similarity demonstrates better generation performance. This is because unlike MSE ($\mathcal{L}_1$ or $\mathcal{L}_2$), cosine similarity focuses on ***directional consistency*** rather than numerical scale consistency. In fact, ***semantic information is determined by the feature direction rather than the feature magnitude*** (semantic similarity emerges in the angular geometry of the embedding space [9]). For example, enlarging or shrinking a feature vector does not affect the semantic meaning. This also aligns with findings from prior representation learning studies [9, 14].

Loss	Gen. Sam.	Rea. Sam.	FSVD↓	FPVD↓	JSD↓	MMD↓
$\mathcal{L}_1$ Loss	10000	34149	66.12	64.25	0.29	3.05
MSE Loss	10000	34149	65.45	63.52	0.28	3.03
Cos. Sim.	10000	34149	64.21	62.85	0.26	3.01

Table 1. The results on nuScenes. Cosine similarity achieves superior generation results than other reconstruction losses.

1.2. Reconstruction Loss of DN

We also investigate the choice of reconstruction loss for DN. As presented in Tab. 2, Huber loss demonstrates superior performance. Since the target $\boldsymbol{v}$ is formed by combining $\boldsymbol{x_0}$ and $\boldsymbol{\epsilon}$, which exhibits highly the distribution difference. The distribution is inherently sharper and more susceptible to large deviations than that of $\boldsymbol{\epsilon}$. The quadratic penalty of $\mathcal{L}_2$ magnifies outliers in high-noise stages, which ultimately results in blurrier generation. Meanwhile, although the linear penalty renders $\mathcal{L}_1$ robust against outliers, the gradient discontinuity often results in unstable training behavior. In contrast, The Huber loss transitions to $\mathcal{L}_2$ loss in regions of low error and to $\mathcal{L}_1$ loss in regions of high error, making the more robust to outliers while maintaining continuous gradients. Therefore, ***Huber loss is often more suitable for diffusion models targeting $\boldsymbol{v}$ in LiDAR generation.***

Methods	Gen. Sam.	Rea. Sam.	FSVD↓	FPVD↓	JSD↓	MMD↓
$\mathcal{L}_1$ Loss	10000	76165	23.14	27.55	0.31	3.38
MSE Loss	10000	76165	25.14	29.33	0.32	3.40
Huber Loss	10000	76165	21.12	25.39	0.30	3.35

Table 2. The results on KITTI-360. Huber loss exhibits the better LiDAR generation quality for DDPMs targeting $\boldsymbol{v}$.

2. Additional Comparison Experiments

2.1. Semantic-to-LiDAR Generation

We also conduct Semantic-to-LiDAR generation on SemanticKITTI (23201 samples) [1] (the implementation in Sec. 5). Tab. 3 presents the results. By leveraging non-latent ControlNet [15], T2LDM demonstrates strong capability in generating LiDAR scenes conditioned on semantic maps. This further validates the effectiveness of ControlNet in non-latent DDPMs. Meanwhile, Fig. 10 and Fig. 11 further provides more visualization results for Semantic-to-LiDAR generation on SemanticKITTI [1] and nuScenes [2].

Methods	Gen. Sam.	Rea. Sam.	FSVD↓	FPVD↓	JSD↓	MMD↓
LiDM [10]	2000	23201	201.41	212.45	0.33	4.71
T2LDM	2000	23201	19.45	23.98	0.29	3.32

Table 3. The results on SemanticKITTI [1]. T2LDM demonstrates excellent results for Semantic-to-LiDAR generation.

3. Conversion Derivations of $\boldsymbol{v}$, $\boldsymbol{x_0}$, and $\boldsymbol{\epsilon}$

In this section, we logically derive the conversion relationships between $\boldsymbol{v}$, $\boldsymbol{x_0}$ and $\boldsymbol{\epsilon}$. Meanwhile, we also provide the explanation regarding the definition of $\boldsymbol{v}$.

3.1. Motivation for Velocity Parameterization

DDPMs typically predict $\boldsymbol{x_0}$ or $\boldsymbol{\epsilon}$. This uses the noise sample $\boldsymbol{x_t} = \boldsymbol{\mu_t} + \boldsymbol{\sigma_t}\boldsymbol{\epsilon}$ as input [3, 6, 8]:

$$\boldsymbol{x_t} = \sqrt{\overline{\alpha}_t}\boldsymbol{x_0} + \sqrt{1-\overline{\alpha}_t}\boldsymbol{\epsilon}. \tag{1}$$

However, the distributions of the target $\boldsymbol{x_0}$ and the target $\boldsymbol{\epsilon}$ remain constant in the training stage, *i.e.*, $\boldsymbol{x_0} \sim \mathcal{P}_{data}$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. ***This means that the training process inevitably becomes unstable (the overly oscillatory loss), due to the varying SNR level of the input $\boldsymbol{x_t}$.***

3.2. Definition of Velocity Parameterization

To address the imbalance of SNR level between $\boldsymbol{x_0}$ prediction and $\boldsymbol{\epsilon}$ prediction caused by the timestep variation, a velocity-field target $\boldsymbol{v}$ combining $\boldsymbol{x_0}$ and $\boldsymbol{\epsilon}$ is introduced [12]. This is achieved by expressing $\boldsymbol{v}$ as a balanced combination relative to $\boldsymbol{x_t}$, aiming to learn a smoother, more stable, and easier-to-optimize target:

$$\boldsymbol{v} = \sqrt{\overline{\alpha}_t}\boldsymbol{\epsilon} - \sqrt{1-\overline{\alpha}_t}\boldsymbol{x_0}. \tag{2}$$

The velocity parameterization defines $\boldsymbol{v}$ as a complementary linear combination of $\boldsymbol{x_0}$ and $\boldsymbol{\epsilon}$ using the same coefficients as in the forward diffusion process. This construction ensures that the statistical structure of $\boldsymbol{v}$ aligns with the noise level of the input $\boldsymbol{x_t}$, providing a well-balanced target across timesteps. Moreover, this definition yields a invertible linear mapping between ($\boldsymbol{x_t}$, $\boldsymbol{v}$) and ($\boldsymbol{x_0}$, $\boldsymbol{\epsilon}$), enabling consistent denoising dynamics and improved training stability compared to $\boldsymbol{x_0}$ prediction and $\boldsymbol{\epsilon}$ prediction:

$$\begin{pmatrix} x_t \\ v \end{pmatrix} = \begin{pmatrix} \sqrt{\bar{\alpha}_t} & \sqrt{1-\bar{\alpha}_t} \\ -\sqrt{1-\bar{\alpha}_t} & \sqrt{\bar{\alpha}_t} \end{pmatrix} \begin{pmatrix} x_0 \\ \epsilon \end{pmatrix}, \tag{3}$$

where $\det \begin{pmatrix} \sqrt{\bar{\alpha}_t} & \sqrt{1-\bar{\alpha}_t} \\ -\sqrt{1-\bar{\alpha}_t} & \sqrt{\bar{\alpha}_t} \end{pmatrix} = \bar{\alpha}_t + (1-\bar{\alpha}_t) = 1$. This means that ($\boldsymbol{x_t}$, $\boldsymbol{v}$) and ($\boldsymbol{x_0}$, $\boldsymbol{\epsilon}$) are invertible.

3.3. Conversion between $\boldsymbol{v}$, $\boldsymbol{x_0}$, and $\boldsymbol{\epsilon}$

This conversion between $\boldsymbol{v}$, $\boldsymbol{x_0}$ and $\boldsymbol{\epsilon}$ mainly focuses on inference sampling. The original DDPM inference sampling is formulated as (the target $\boldsymbol{\epsilon}$) [3, 7]:

$$\begin{aligned} \boldsymbol{x_{t-1}} = \frac{1}{\sqrt{\alpha_t}}(\boldsymbol{x_t} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon_{t-1}}) \\ + \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}(1-\alpha_t)}\boldsymbol{\epsilon}. \end{aligned} \tag{4}$$

Meanwhile, using Eq. 1, Eq. 2 and Eq. 4, $\boldsymbol{x_0}$ and $\boldsymbol{\epsilon}$ can be expressed in terms of $\boldsymbol{v}$ and $\boldsymbol{x_t}$:

$$\begin{aligned} \boldsymbol{\epsilon} &= \sqrt{1-\bar{a}_t}\boldsymbol{x_t} + \sqrt{\bar{a}_t}\boldsymbol{v}, \\ \boldsymbol{x_0} &= \sqrt{\bar{a}_t}\boldsymbol{x_t} - \sqrt{1-\bar{a}_t}\boldsymbol{v}, \\ \boldsymbol{x_0} &= \frac{\boldsymbol{x_t}}{\sqrt{\bar{a}_t}} - \frac{\sqrt{1-\bar{a}_t}\boldsymbol{\epsilon}}{\sqrt{\bar{a}_t}} \end{aligned} \tag{5}$$

Then, substituting Eq. 5 into Eq. 4 yields the inference sampling formulas for the target $\boldsymbol{x_0}$ and the target $\boldsymbol{v}$:

$$\begin{aligned} \boldsymbol{x_{t-1}} &= \frac{1}{\sqrt{\alpha_t}}\boldsymbol{x_0} + \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}(1-\alpha_t)}\boldsymbol{\epsilon}, \\ \boldsymbol{x_{t-1}} &= \frac{1}{\sqrt{\alpha_t}}(\boldsymbol{x_t} - \frac{1-\alpha_t}{\sigma_t}[\sigma_t\boldsymbol{x_t} + \sqrt{\bar{\alpha}_t}\,v_\theta]) + \tilde{\sigma}_t\boldsymbol{\epsilon} \end{aligned} \tag{6}$$

where $\sigma_t = \sqrt{1-\bar{\alpha}_t}$, $\tilde{\sigma}_t = \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}(1-\alpha_t)}$.
The above provides the conversion between $\boldsymbol{v}$, $\boldsymbol{x_0}$ and $\boldsymbol{\epsilon}$.

4. Text-LiDAR Benchmark

In this section, we first describe the text annotation process of T2nuScenes and formalize the generation controllability metric (TBK). Subsequently, we provide the text-related sample distribution and the comparison between the original and re-annotated text descriptions in nuScenes [2].

Our primary goal is to provide a principled annotation strategy for LiDAR text descriptions, leveraging existing priors to construct high-quality Text–LiDAR pairs. We sincerely hope that researchers can further refine or develop improved text annotation methods in the future.

4.1. Annotation Process

In nuScenes [2], we categorize the text descriptions into scene-level and object-level types. Meanwhile, ***different types of text descriptions are stored separately, allowing them to be freely combined***, as shown in Tab. 4.

For scene-level descriptions, we reorganize the original text prompts from nuScenes according to the principles outlined in Sec. 3.3 of SM.

Weather. In the original nuScenes dataset, two weather conditions are provided: ***Sunny*** and ***Rainy***.

Time. Similarly, the original nuScenes dataset divides time of a day into ***Day*** and ***Night***.

For object-level descriptions, we leverage the 3D box priors and fix the target class as "car" to generate text descriptions that is closer to human natural language. This allows us to extract prior information about the quantity, location, and orientation of the target object in the scene.

Quantity. The 3D boxes provide information about the number of the target object. Therefore, this can easily generate the following types of text descriptions regarding the quantity in the scene:

- No target object.
- One/Two/... target object/s.
- There is/are one/two/... target object/s in the scene..
- One/Two/... target object/s. One target object is in front. Other target object/s is/are behind.
- Less/More than five target objects.

As shown in Tab. 1 of MT, overly detailed text prompts can significantly increase the training difficulty of the generative model. Therefore, we recommend using more coarse-grained text prompts.

Location. Meanwhile, since 3D boxes contain precise object coordinates, we can derive the target relative position to other objects in the scene. Given the target object box center $(cx_{tar}, cy_{tar}, cz_{tar})$ and another object box center $(cx_{ano}, cy_{ano}, cz_{ano})$, their relative position is defined:

$$\begin{aligned} &(cx, cy, cz) = \\ &\qquad (cx_{tar} - cx_{ano}, cy_{tar} - cy_{ano}, cz_{tar} - cz_{ano}), \\ &p_1 = \begin{cases} ahead, & cx > threshold, \\ behind, & cx < -threhold, \\ aligned, & -threhold \leq cx \leq threshold, \end{cases} \end{aligned}$$

Level	Type	T2nuScenes Prompt Example	Sample Distribution	nuScenes Prompt Example
(■) Object	Quantity	(♠) Two cars.	857,1873,2099,29320	Two parked motorcycles, overtake.
		(♠) There are two cars in the scene.	857,1873,2099,29320	Go straight, two bicycles, turn right, two bendy busses.
		(♠) Two cars. One car is in front. One car is behind.	857,1873,2099,29320	Two turning trucks, wait, turn left.
		(♣) Two cars. → There are two cars in the scene.	857,1873,2099,29320	two oncoming bikes, parked motorcycle, bike rack.
		(♣) There are two cars in the scene. → Two cars.	857,1873,2099,29320	Ped standing, ped sitting, two parked motorcycles on road.
	Adjusted Text	(♦) Less/More than five cars.	10692,23457	-
	Location	One car is behind to the right of one pedestrian.	681,478,...,339,468 (41 texts)	Passing scooter, wait at intersection, turn right.
	Adjusted Text	(♦) No car./One car is around one pedestrian/barrier/truck.	12227,11534,3819,6523	-
	Orientation	One car is facing backward.	7416,6662,9994,8711,1366	-
	Adjusted Text	(♦) No car./One car is facing right/left.	1366,16127,16656	-
(▲) Scene	Weather	Rainy/Sunny.	6670,27479	Night, rain, bump, peds, congestion, parked car.
	Time	Night/Day.	3987,30162	Night, turn left, bumps, lightning.
-	Wea., Qua.	Rainy. Two cars.	1901,1706,...,205,42 (14 texts)	Rain, Wait at intersection, trucks, cars, ped.
-	Wea., Loc.	(★) Rainy. One car is around one pedestrian.	10101,9876,...,1658,853 (8 texts)	Rain, turn left, turn left.
-	Wea., Ori.	Rainy. One car is facing backward.	5740,5368,...,1294,114 (10 texts)	-
-	Tim., Qua.	Night. Two cars.	21656,1962,...,302,206 (14 texts)	Night, ped in dark, parked cars.
-	Tim., Loc.	Night. One car is around one pedestrian.	10140,10099,...,315,104 (8 texts)	Night, buses, peds, rain, right turn.
-	Tim., Ori.	Night. One car is around one pedestrian.	8981,7655,...,695,464 (10 texts)	-
-	Qua., Loc., Ori.	Two cars. One car is around one pedestrian. One car is facing backward.	2795,2152,...,24,7 (103 texts)	-
-	Wea., Qua., Loc., Ori.	Rainy, Two cars. One car is around one pedestrian. One car is facing backward.	1850,1182,...,2,1 (198 texts)	-
-	Tim., Qua., Loc., Ori.	Rainy, Two cars. One car is around one pedestrian. One car is facing backward.	2472,2013,...,1,1 (185 texts)	-

Table 4. Results of different text forms. "Text1. → Text2." means that the model is trained with the text form of "Text2", while using "Text1" as conditional input in inference. "Wea., Loc." denotes "Weather, Location", exhibiting the more uniform sample distribution. Meanwhile, original text descriptions (the last column) in nuScenes are more unnatural than those (the third column) in T2nuScenes.

$$p_2 = \begin{cases} left, & cy > threshold, \\ right, & cy < -threhold, \\ center, & -threhold \leq cy \leq threshold, \end{cases} \tag{7}$$

where the distance threshold is set 2.0m.

Subsequently, we can use p_1 and p_2 to form text descriptions of scene layouts, such as "One target object is p_1 to the p_2 of another object".

However, as mentioned in Sec. 3.3 of MT, an overly dispersed sample distribution can significantly degrade generation quality and controllability. Therefore, instead of focusing on specific relative positions, ***we only determine whether the target object and another object co-occur in the same scene***.

Finally, this yields the following type of text description regarding the location in the scene:

- No target object.
- One target object is ahead/behind/aligned to the left/right/center of another object.
- One target object is around another object.

Orientation. Furthermore, the 3D box also provides the orientation angle (in radians, yaw) of the target object. Therefore, we can also easily obtain the orientation of the target object in the scene:

$$deg = degrees(yaw),$$
$$o = \begin{cases} forward, & 315 \leq deg \ or \ 45 > deg, \\ left, & 45 \leq deg < 135, \\ backward, & 135 \leq deg < 225, \\ right, & 225 \leq deg < 315 \end{cases} \tag{8}$$

where $degrees(\cdot)$ is a function that converts radians to degrees for consistent angular representation.

Then, we can use o to describe the target object orientation in the scene, for example: "One target object is facing o". Therefore, we can generate orientation-based descriptions of the scene:

- No target object.
- One target object is facing forward/left/backward/right.

The above process outlines how 3D box priors can be used to generate scene descriptions in natural language. Compared to directly using 3D boxes as conditions, natural language prompts are more accessible and flexible for providing semantic guidance to generative models.

4.2. Generation Controllability Evaluation Metric

We first utilize an existing 3D detector [5] to predict 3D boxes by detecting the generated LiDAR scenes. The matching **R**ate between the **T**ext prompts and the predicted 3D **B**oxes is used to measure the controllability of the generative model. For example, the text prompts for the three generated LiDAR scenes are: "Two cars", "One car", and "Five cars". The detector identified 1, 3, and 5 cars in the three LiDAR scenes, respectively. Thus, the **T**ext-to-**B**ox matching **R**ate (TBR) is 1/3≈33.33%.

Therefore, TBR can be formalized as: TBR = The Number of Correctly Matched LiDAR Scenes / The Number of Generated LiDAR Scenes.

Benefiting from text descriptions derived from 3D box priors, we can quantitatively evaluate conditional controllability of generative models in Text-to-LiDAR generation.

4.3. Sample Distributions

In Tab. 4, we provide more detailed information regarding the distribution of text description samples compared to Tab. 1 in MT. We can observe that the "Wea., Loc." combination exhibits a more uniform distribution compared to others, due to the minimum sample size of 853. This is significantly higher than other text combinations. Meanwhile, the diversity of the text descriptions is also crucial. Therefore, we consider "Wea., Loc." to be the optimal combination of text prompts in nuScenes.

4.4. T2nuScenes vs. nuScenes

Tab. 4 also presents the comparison between the text descriptions of T2nuScenes and nuScenes. Clearly, the original text descriptions in nuScenes deviate significantly from natural human language. This hinders the Text-to-LiDAR generation effectiveness in training and the generalization of text conditions during inference, leading to the degradation in generation quality and controllability. Tab. 5 further presents the comparative result of T2LDM generation performance on T2nuScenes and nuScenes. T2nuScenes demonstrates superior training effectiveness.

Methods	Gen. Sam.	Rea. Sam.	FSVD↓	FPVD↓	JSD↓	MMD↓
nuScenes [2]	10000	34149	68.44	66.92	0.30	3.05
T2nuScenes	10000	34149	66.93	65.84	0.28	3.05

Table 5. The text-guided results on nuScenes and T2nuScenes.

5. Implementation

In this section, we provide the hardware specifications and training time required for T2LDM. Meanwhile, we also describe the model hyperparameters and the implementation of the non-latent ControlNet [15] for T2LDM.

5.1. Hardware requirement and Training Time

We used ***8 NVIDIA 4090 GPUs*** to train T2LDM on nuScenes [2], KTIIT-360 [4] and SemanticKITTI [1], which took approximately ***47 hours, 58 hours and 57 hours*** on unconditional and conditional generation, respectively. For non-latent ControlNet [15], we fine-tuned T2LDM (only using frozen DN) on nuScenes and SemanticKITTI about ***35 hours and 48 hours*** for Sparse-to-Dense, Dense-to-Sparse, and Semantic-to-LiDAR generation.

5.2. Model Hyperparameters

We implement a highly flexible network framework that constructs a corresponding U-Net architecture from input lists of Encoder, Middle, and Decoder, as illustrated in Fig. 1(left). T2LDM adopts the U-Net architecture from Stable Diffusion [11]. Meanwhile, due to the limited sample size of LiDAR datasets, an excessive number of parameters can easily lead to model collapse, resulting in oversmoothed and homogeneous generation results, as seen in LiDM [10]. Therefore, T2lDM employs a smaller channel dimension than Stable Diffusion, as shown in Fig. 1(left).

Config	Parameter
T	1024
Time Embedding	Cos-Sin (384)
schedule	cosine
SNR gamma	5.0
diffusion target	v
GN iterations	100k
λ	(0.001, 0.01, 0.1, 1.0)
λ interval	25k
CFG scale	4.0
CFG dropout	0.1
CLIP model	ViT-L/14
CLIP channel	768
conv type	circular
base channel	64
channels	(1,2,4,4)
strides	(1, 2), (2, 2), (2, 2)
attention types	(linear, linear, linear, vanilla)
attention heads	(2,4,8,8)
attention pe	rope
skip connection scale	sqrt(2)
norm types	(group, group, group, group)
position encoding	dpe
use ema	True
ema decay	0.9997
ema update	1

Table 6. The parameters of network framework for T2LDM.

nuScenes [2]		KITTI-360 [4]		SemanticKITTI [1]	
Config	Parameter	Config	Parameter	Config	Parameter
Optimizer	Adam	Optimizer	Adam	Optimizer	Adam
Scheduler	LR Cosine	Scheduler	LR Cosine	Scheduler	LR Cosine
LR	1e-4	LR	1e-4	LR	1e-4
Weight De.	0.01	Weight De.	0.01	Weight De.	0.01
Batch Size	16	Batch Size	16	Batch Size	16
Iterations	400K	Iterations	400K	Iterations	400K
resolution	(32,1024)	resolution	(64,1024)	resolution	(64,1024)
depth range	(0.01,50.0)	depth range	(1.45,80.0)	depth range	(1.45,80.0)
fov	(-3,25)	fov	(-3,25)	fov	(-3,25)
use intensity	True	use intensity	True	use intensity	True

Table 7. The training hyperparameters of T2LDM.

Furthermore, the detailed parameters of the network architecture of T2LDM are described in Tab. 6, while the training hyperparameters for each benchmark (nuScenes, KITTI-360, SemanticKITTI) are shown in Tab. 7.

5.3. Network Architecture

In Fig. 1, Encoder, Middle, and Decoder consist of four modules: ***ResBlock (RB), AttentionBlock (AB), DownsamplingBlock (DB), and UpsamplingBlock (UB)***.

ResBlock. RB consists of two GroupNorm layers, two activation layers, two circular convolution [13] with a residual skip. We adopt the Swish activation $f(x) = x \cdot sigmoid(x)$. Meanwhile, the time embedding is injected through a MLP layers. The output is computed as $x + h$.

$$\begin{aligned} h &= Conv_1(Swish_1(GN_1(x))), \\ h &= h + mlp(temb), \\ h &= Conv_2(Swish_2(GN_2(h))), \\ x &= h + x. \end{aligned} \tag{9}$$

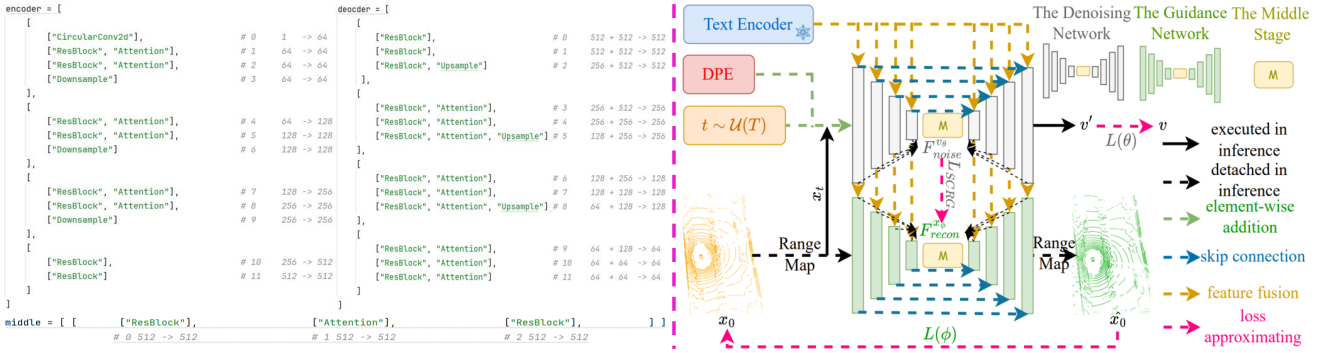

Figure 1. The overall framework of T2LDM. DN and GN are composed of an Encoder, a Middle Stage, and a Decoder. The core modules of T2LDM include: ResBlock (RB), AttentionBlock (AB), DownsamplingBlock (DB), and UpsamplingBlock (UB). To effectively process the spherical projection of LiDAR data, T2LDM incorporates Circular Convolution [13] to adapt to the unfolded Range Map.

Following Stable Diffusion [11], we apply zero-initialization to the $Conv_2(\cdot)$ layers used the skip connections to enhance training stability.

AttentionBlock. T2LDM employs linear AB and conventional AB for unconditional and conditional generation.

For linear AB (unconditional generation), given $F^{v_\theta}_{noise} \in \mathbb{R}^{l \times C^{v_\theta}} \rightarrow (Q, K, V) \in \mathbb{R}^{C \times l}$ via mlps, a self-attention block is conducted:

$$O = (\frac{softmax(K)V^T}{\sqrt{C}})Q + F^{v_\theta}_{noise}, \\ F = ffn(O) + O. \tag{10}$$

For conventional AB (conditional generation), given $F^{v_\theta}_{noise} \in \mathbb{R}^{l \times C^{v_\theta}} \rightarrow (Q) \in \mathbb{R}^{l \times C}$ and $F^{CLIP}_{text} \in \mathbb{R}^{n \times 768} \rightarrow (K, V) \in \mathbb{R}^{n \times C}$ via mlps, a cross-attention block is conducted:

$$O = softmax(\frac{QK^T}{\sqrt{C}})V + F^{v_\theta}_{noise}, \\ F = ffn(O) + O. \tag{11}$$

```python
class Downsample(nn.Module):
    def __init__(self, in_channels, with_conv, stride):
        super().__init__()
        self.with_conv = with_conv
        self.stride = stride
        if self.with_conv:
            k, p = DOWNSAMPLE_STRIDE2KERNEL_DICT[stride], DOWNSAMPLE_STRIDE2PAD_DICT[stride]
            self.conv = CircularConv2d( *args: in_channels, in_channels, kernel_size=k, stride=stride, padding=p)

    def forward(self, x):
        if self.with_conv:
            x = self.conv(x)
        else:
            x = torch.nn.functional.avg_pool2d(x, kernel_size=self.stride, stride=self.stride)
        return x
```

Figure 2. The code of the downsampling Block.

$l = h \times w$. Linear AB operates attention along the channel dimension ($W \in \mathbb{R}^{C \times l}$), resulting in linear computational complexity. In contrast, conventional AB integrates similar features in the spatial dimension ($W \in \mathbb{R}^{l \times C}$), enhancing contextual modeling capacity.

```python
class Upsample(nn.Module):
    def __init__(self, in_channels, with_conv, stride):
        super().__init__()
        self.with_conv = with_conv
        self.stride = stride
        if self.with_conv:
            k, p = UPSAMPLE_STRIDE2KERNEL_DICT[stride], UPSAMPLE_STRIDE2PAD_DICT[stride]
            self.conv = CircularConv2d( *args: in_channels, in_channels, kernel_size=k, padding=p)

    def forward(self, x):
        x = torch.nn.functional.interpolate(x, scale_factor=self.stride, mode='bilinear', align_corners=True)
        if self.with_conv:
            x = self.conv(x)
        return x
```

Figure 3. The code of the upsampling Block.

```python
class CircularConv2d(nn.Conv2d):
    def __init__(self, *args, **kwargs):
        if 'padding' in kwargs:
            self.is_pad = True
            if isinstance(kwargs['padding'], int):
                h1 = h2 = v1 = v2 = kwargs['padding']
            elif isinstance(kwargs['padding'], tuple):
                h1, h2, v1, v2 = kwargs['padding']
            else:
                raise NotImplementedError
            self.h_pad, self.v_pad = (h1, h2, 0, 0), (0, 0, v1, v2)
            del kwargs['padding']
        else:
            self.is_pad = False
        super().__init__(*args, **kwargs)
    def forward(self, x: Tensor) -> Tensor:
        if self.is_pad:
            if sum(self.h_pad) > 0:
                x = nn.functional.pad(x, self.h_pad, mode="circular")  # horizontal pad
            if sum(self.v_pad) > 0:
                x = nn.functional.pad(x, self.v_pad, mode="constant")  # vertical pad
        x = self._conv_forward(x, self.weight, self.bias)
        return x
```

Figure 4. The code of Circular Convolution.

DownsamplingBlock. DB progressively downsamples the range map using 2D pooling. The corresponding code is provided in Fig. 2.

UpsamplingBlock. UB progressively upsamples the range map using bilinear interpolation. The corresponding code is provided in Fig. 3.

Circular Convolution. Circular Convolution applies circular padding along the horizontal axis and zero padding vertically to preserve the periodic structure of range-view inputs. Compare to traditional convolution, this eliminates boundary discontinuities at the $0°/360°$ transition. The corresponding code is provided in Fig. 4.

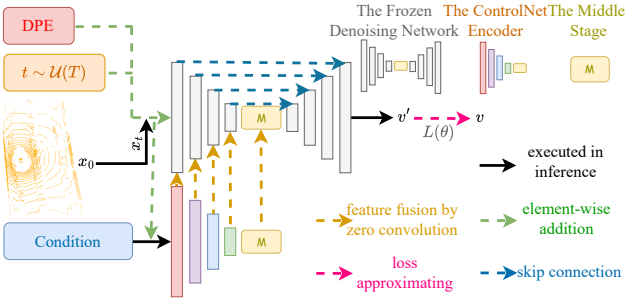

Figure 5. The overall framework of non-latent ControlNet.

5.4. Non-Latent ControlNet

To the best of our knowledge, ***we present the first attempt to integrate ControlNet [15] into non-latent DDPMs in 3D generation***. Similar to the approach in latent ControlNet [15], we freeze DN of unconditional T2LDM and introduce an additional encoder ($\mathcal{E}_c$), which shares the same architecture as the encoder in DN, to learn the conditional features. Experimental results demonstrate that T2LDM, leveraging the non-latent ControlNet, can achieves significant controllable generation (see Sec. 5.4 of MT, Fig. 10, and Fig. 11).

Sparse-to-Dense/Dense-to-Sparse Generation. During training, $\mathcal{E}_c$ takes sparse point clouds as input to extract conditional features (the process of sparse point clouds in Sec. 5.4 of MT). Subsequently, zero convolution layers are utilized to aggregate features from corresponding layers in DN and $\mathcal{E}_c$. The skip connections inject conditional information from $\mathcal{E}_c$ into the decoder of DN. Finally, the reconstruction loss from DN is back-propagated to $\mathcal{E}_c$ to update the parameters. Fig. 5 illustrates the overall framework. Meanwhile, since the output LiDAR data shape is determined by the input noise size, T2LDM can perform upsampling or downsampling at arbitrary rates, as demonstrated in Fig. 6 of MT. Furthermore, following the inference track of PUDM [6], T2LDM further improves generation results.

Semantic-to-LiDAR Generation. Similarly, following an analogous procedure, we feed the normalized semantic map (Semantic Map / Class Num) into $\mathcal{E}_c$ to achieve LiDAR scene generation conditioned on the semantic map.

6. Limitations and Future Works

6.1. Limited Generality for Text Annotation

Although leveraging object detection priors to generate LiDAR scene text descriptions is effective, most existing LiDAR datasets (e.g., KITTI-360 [4]) lack 3D box annotations. This inspires us to develop alternative priors for generating effective scene descriptions.

6.2. Lack of Multi-Conditional Control

Currently, T2LDM can only achieve effective controllable generation with a single condition. However, existing LiDAR data often possesses multiple annotation priors, such as semantic maps and 3D bounding boxes. In future work, we will integrate multi-conditional controllable generation into T2LDM.

6.3. Increased Parameter Demand

Despite SCRG only requiring a Guidance Network (GN), sharing the same architecture as the Denoising Network (DN), for joint training in the early stages and being decoupled during inference, the doubled parameter count still imposes high resource demands. Therefore, the future research will explore the path of pruning GN to a lightweight version, alleviating computational overhead.

7. More Visualization Results

We provide additional generation visualization results:

- Fig. 6 (unconditional generation on nuScenes [2])
- Fig. 7 (unconditional generation on KITTI-360 [4])
- Fig. 8 (text-guided generation on nuScenes [2])
- Fig. 9 (reconstruction input via DN) on KITTI-360 [4] (up) and nuScenes [2] (bottom)
- Fig. 10 (Semantic-to-LiDAR generation on SemanticKITTI [1])
- Fig. 11 (Semantic-to-LiDAR generation on nuScenes [2])

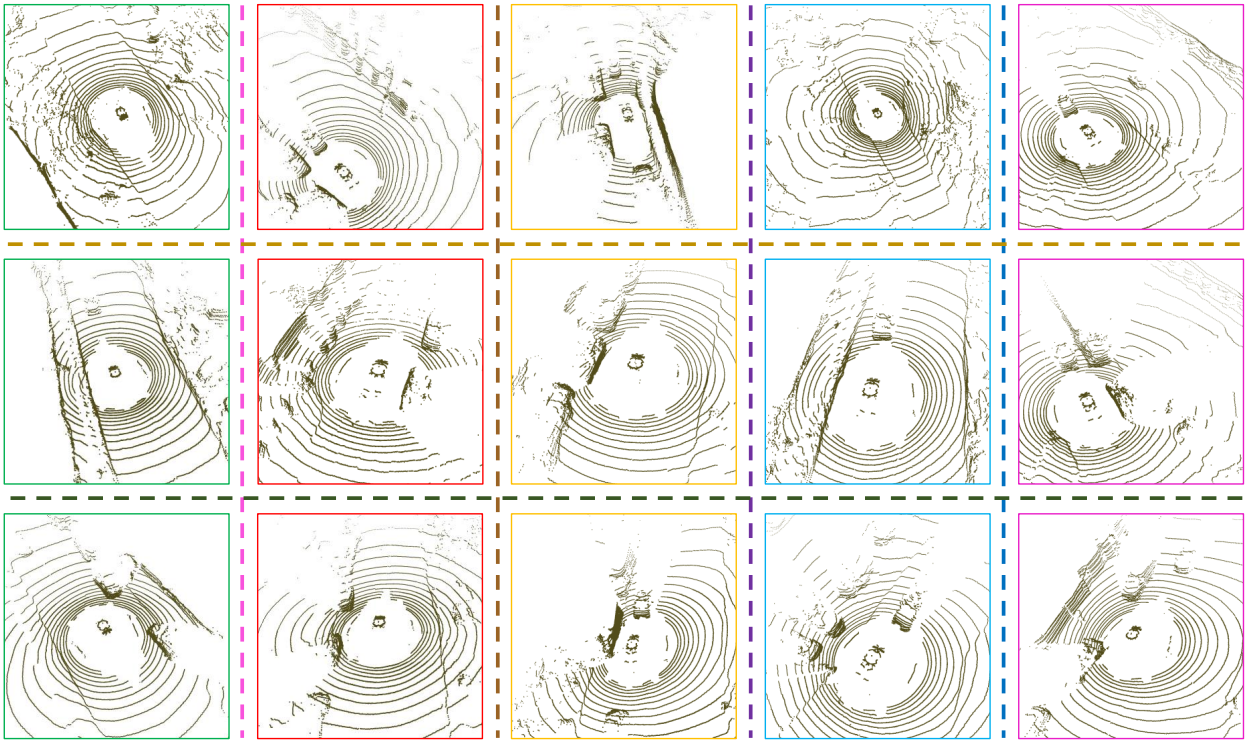

Figure 6. The visualization of unconditional generation for T2LDM on nuScenes [2].

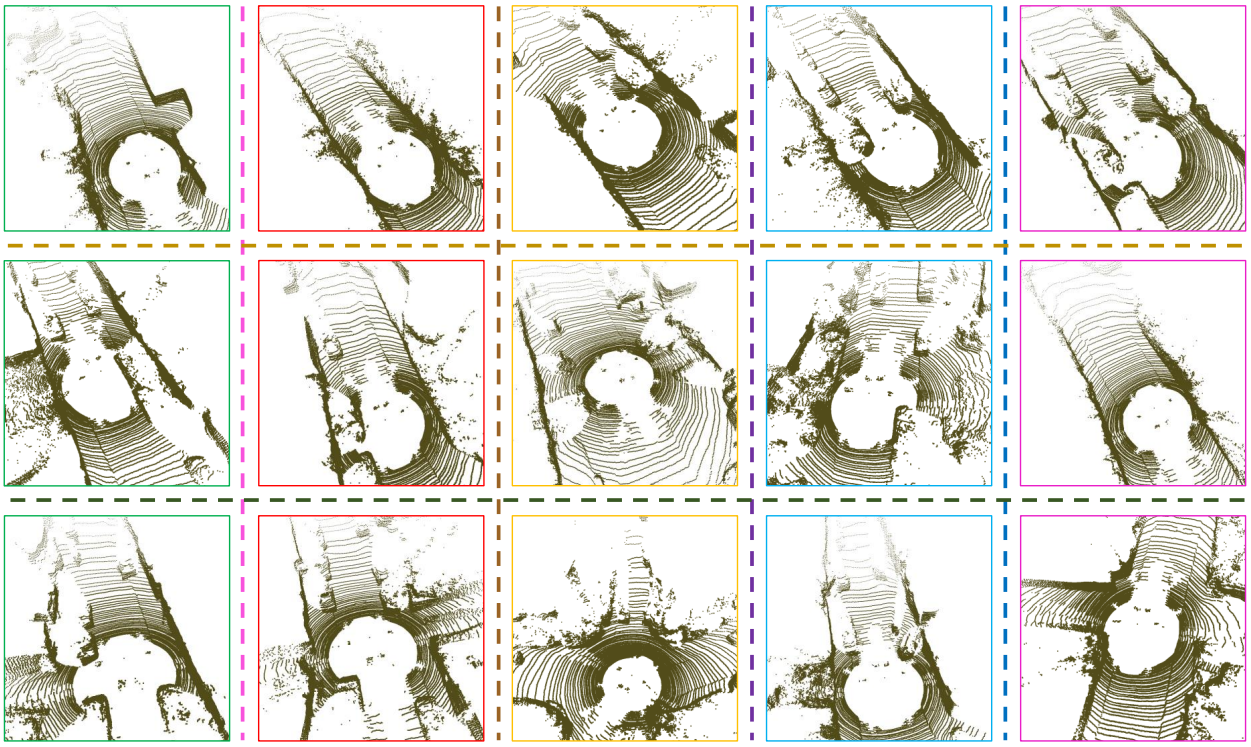

Figure 7. The visualization of unconditional generation for T2LDM on KITTI-360 [4].

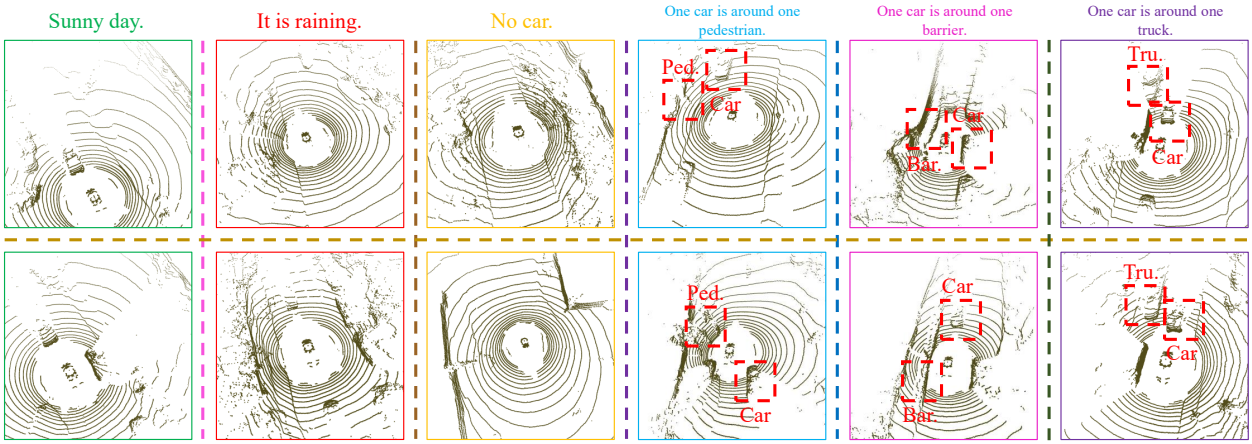

Figure 8. The visualization of text-guided generation for T2LDM on nuScenes [2].

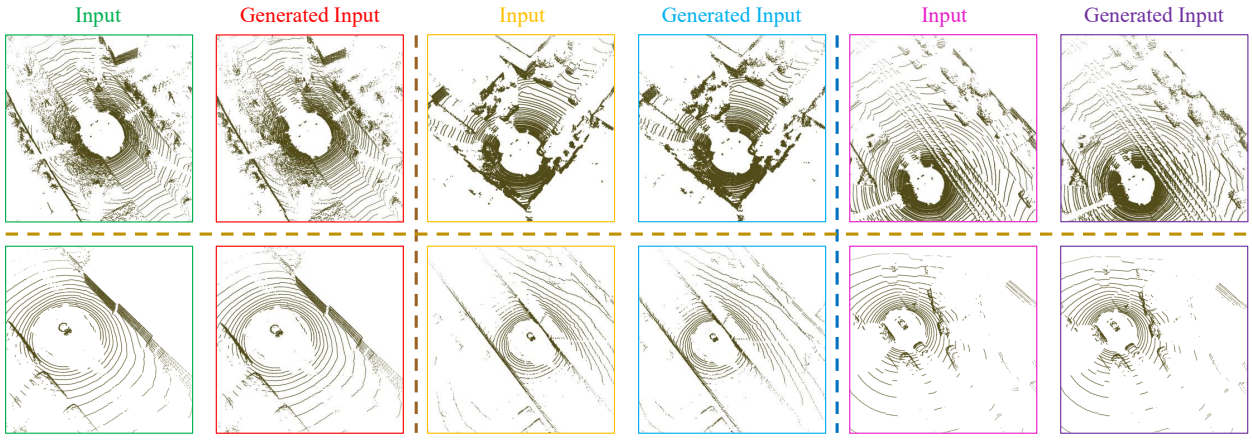

Figure 9. The visualization of reconstruction input via DN on KITTI-360 [4] (up) and nuScenes [2] (bottom).

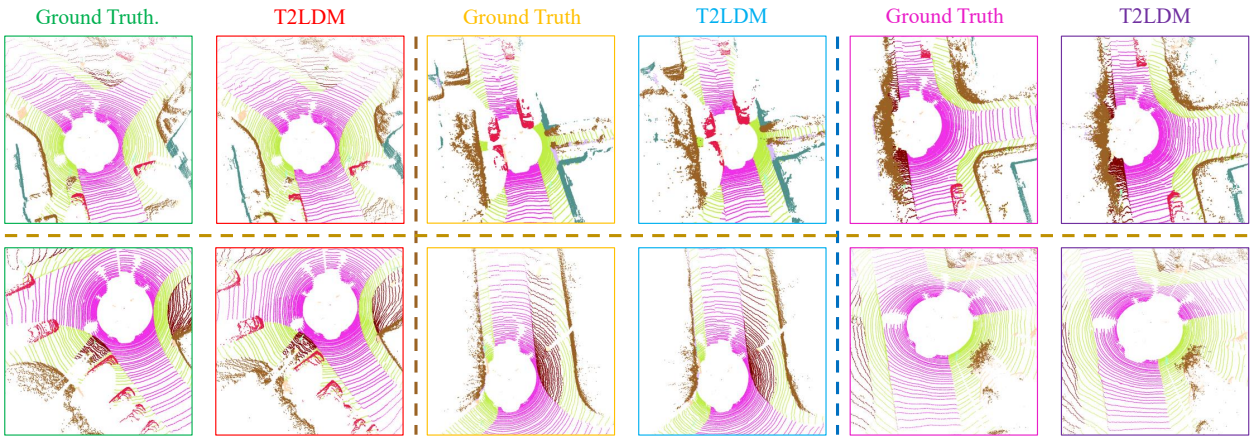

Figure 10. The visualization of Semantic-to-LiDAR generation for T2LDM on SemanticKITTI [1].

Figure 11. The visualization of Semantic-to-LiDAR generation for T2LDM on nuScenes [2].

References

[1] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9297–9307, 2019. 1, 4, 6, 8

[2] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 1, 2, 4, 6, 7, 8, 9

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1, 2

[4] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3292–3310, 2022. 4, 6, 7, 8

[5] Shuai Liu, Mingyue Cui, Boyang Li, Quanmin Liang, Tinghe Hong, Kai Huang, and Yunxiao Shan. Fshnet: Fully sparse hybrid network for 3d object detection. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 8900–8909, 2025. 3

[6] Wentao Qu, Yuantian Shao, Lingwu Meng, Xiaoshui Huang, and Liang Xiao. A conditional denoising diffusion probabilistic model for point cloud upsampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20786–20795, 2024. 1, 6

[7] Wentao Qu, Guofeng Mei, Jing Wang, Yujiao Wu, Xiaoshui Huang, and Liang Xiao. Robust single-stage fully sparse 3d object detection via detachable latent diffusion. *arXiv preprint arXiv:2508.03252*, 2025. 2

[8] Wentao Qu, Jing Wang, YongShun Gong, Xiaoshui Huang, and Liang Xiao. An end-to-end robust point cloud semantic segmentation network with single-step conditional diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 27325–27335, 2025. 1

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021. 1

[10] Haoxi Ran, Vitor Guizilini, and Yue Wang. Towards realistic scene generation with lidar diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14738–14748, 2024. 1, 4

[11] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 4, 5

[12] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022. 1

[13] Colton Stearns, Alex Fu, Jiateng Liu, Jeong Joon Park, Davis Rempe, Despoina Paschalidou, and Leonidas J Guibas. Curvecloudnet: Processing point clouds with 1d structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27981–27991, 2024. 4, 5

[14] Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. *arXiv preprint arXiv:2410.06940*, 2024. 1

[15] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3836–3847, 2023. 1, 4, 6