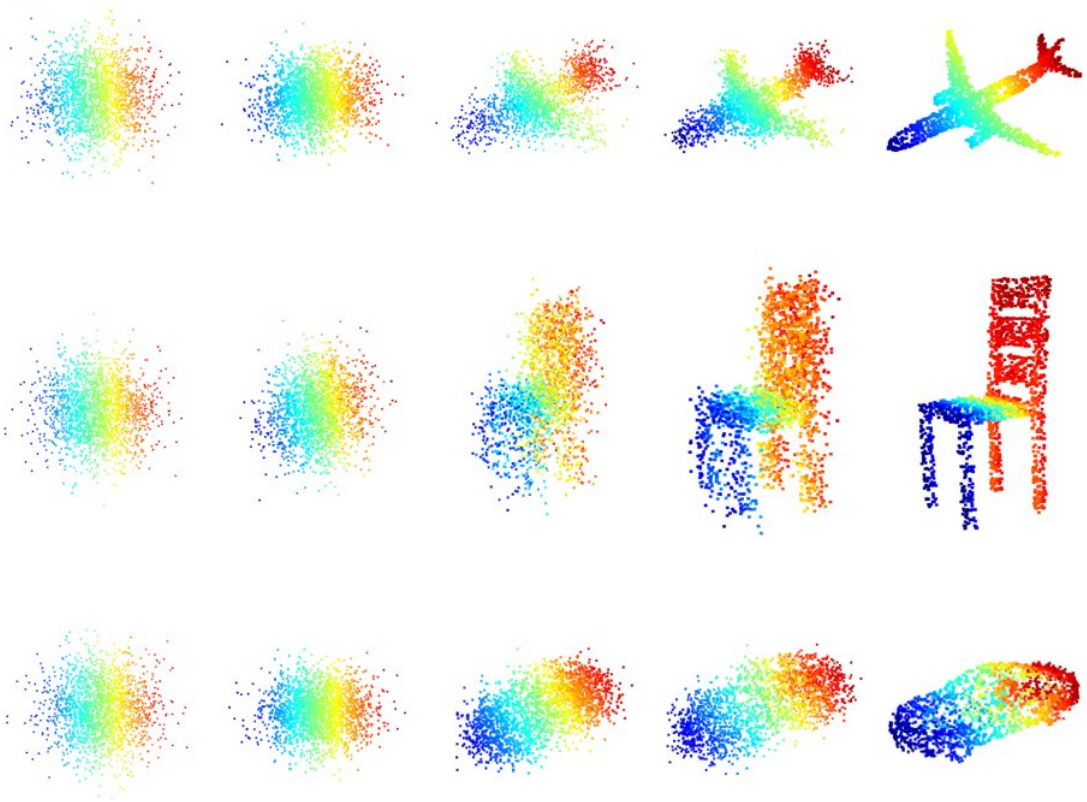
CHALMERS
UNIVERSITY OF TECHNOLOGY



3D Shape Generation through Point Transformer Diffusion

Master's thesis in Data science and AI

Ji Lan

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

3D Shape Generation through Point Transformer Diffusion

Ji Lan

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Group of Computer Vision and Medical Image Analysis
Chalmers University of Technology
Gothenburg, Sweden 2023

3D Shape Generation through Point Transformer Diffusion
Ji Lan

Supervisor: Lucas Brynte, PhD student, Chalmers University of Technology
Examiner: Fredrik Kahl, Professor, Chalmers University of Technology

Master's Thesis 2023
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Group of Computer Vision and Medical Image Analysis
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of shape generations for an airplane, a chair, and a car.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

3D Shape Generation through Point Transformer Diffusion
Ji Lan
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Diffusion models are a novel class of generative models, which have demonstrated promising results in 3D point cloud generation. Meanwhile, Transformer-based models have achieved impressive outcomes across various benchmarks in the field of 3D point cloud understanding. However, the integration of a Transformer based on the local self-attention mechanism with a diffusion model for 3D point cloud generation remains unexplored. In this work, we propose Point Transformer Diffusion (PTD), a probabilistic and flexible generative model. PTD integrates the standard Denoising Diffusion Probabilistic Model, adapted for 3D data, with Point Transformer, a local self-attention network specifically designed for 3D point clouds. To enhance PTD's performance, several adjustments and techniques are implemented to align the Point Transformer model more effectively with the diffusion model and with the specific generation task. Experiments demonstrate PTD's ability to generate realistic and diverse shapes. Furthermore, the evaluation results of PTD are comparable to, and in some cases even marginally superior to, those achieved by Point-Voxel Diffusion, which is a state-of-the-art approach. We hope that our work will inspire future investigations into architectures that combine diffusion models and Transformers, along with their application to a wide range of 3D generation tasks.

Keywords: 3D point clouds, 3D shape generation, diffusion models, Transformers.

Acknowledgements

I am incredibly grateful for the remarkable experience during my master's thesis. First and foremost, I would like to express my sincere appreciation to Dr. Fredrik Kahl for being my examiner and providing me with the exceptional opportunity to undertake this one-year thesis project. His guidance introduced me to the world of 3D computer vision research and inspired me to delve into diffusion models, allowing me to learn and explore cutting-edge techniques. Furthermore, he provided me with access to office facilities, the Alvis computer cluster[1], group meetings, and various symposiums, offering me a unique glimpse into the life of a PhD student.

I am equally grateful to Lucas Brynte for his invaluable role as my supervisor. Throughout the past year, we engaged in numerous insightful discussions, covering a wide range of topics, including theories from research papers, cluster and tool utilization, neural network model implementations, research methodologies, and lighthearted anecdotes. During the most challenging moments, when progress seemed elusive, Lucas provided unwavering support. He helped me overcome obstacles, offered illuminating advice, and encouraged me to persist in my explorations.

To all the esteemed members of the computer vision research group at Chalmers, including professors, researchers, postdocs, and PhD students: I am continually amazed by the intelligence, passion, and talent that each one of you possesses. It is truly fantastic to talk and learn from you week after week. I wish you all the best for your continued success in both your academic pursuits and professional careers.

Finally, I want to express my deep gratitude to my parents, who are living in China, and my girlfriend, who is working in the UK now. Without your understanding and support, I would not have embarked on the journey to chase my dream and realize my career aspirations. As this fantastic journey continues, I eagerly await the opportunity to convey my heartfelt appreciation to each of you in the acknowledgments of my future PhD thesis. Whenever I engage with any of you, I am filled with the abundance of support and love. I cherish and love each one of you, always.

Ji Lan, Gothenburg, August 2023

[1]The computations were enabled by resources provided by Chalmers e-Commons at Chalmers.

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis:

DDPM	Denoising Diffusion Probabilistic Models
VAE	Variational Auto-Encoder
GAN	Generative Adversarial Network
KL divergence	Kullback–Leibler divergence
ELBO	Evidence Lower Bound
OA	Overall Accuracy
mAcc	mean Accuracy within each category
IoU	Intersection over Union
mIoU	mean Intersection over Union
cat. mIoU	overall average category IoU
ins. mIoU	overall average instance IoU
CD	Chamfer Distance
EMD	Earth Mover's Distance
1-NN	1-nearest neighbor
1-NNA	1-nearest neighbor accuracy
kNN	k-nearest neighbors
FPS	farthest point sampling
IDW	inverse distance weighting
MLP	Multi-Layer Perceptron
ReLU	Rectified Linear Unit
CNN	Convolutional Neural Network
PVD	Point-Voxel Diffusion
PTD	Point Transformer Diffusion

Contents

List of Figures

List of Tables

1
Introduction

In this introductory chapter, we will present the background of the master's thesis, which encompasses the evolution of 3D generative models, two distinct types of 3D data representation, and models for point cloud learning. Subsequently, we will outline the objectives of the thesis, enumerate our contributions, and provide an overview of the thesis structure.

In recent years, 3D point clouds have gained popularity as a data representation due to several key factors. Firstly, the widespread availability and advancement of depth sensing technologies, such as LiDAR, have greatly improved the ease and accuracy of capturing 3D data. Secondly, 3D point clouds offer a compact yet information-rich representation of spatial data, with each point carrying three-dimensional coordinates that effectively describe its position in the world space. Lastly, their versatility makes them applicable across various industries, including robotics, autonomous vehicles, virtual reality, and architecture, where precise spatial information is crucial. Their ability to preserve raw geometric data without explicit connectivity or mesh structure makes them particularly valuable for tasks such as object recognition [44, 45] and scene understanding [55, 28]. As a result, research related to 3D point clouds has attracted increasing attention, leading to significant progress in various point cloud tasks, such as classification [44, 45, 43], segmentation [55, 28], and shape generation [62, 67, 74].

This thesis focuses on the research of shape generation for 3D point clouds. The generative modeling in this project utilizes DDPM [17], a prominent diffusion model that belongs to a novel class of generative models. Additionally, the thesis explores how a local self-attention network can serve as the backbone for the diffusion model, resulting in state-of-the-art outcomes. As a result, this thesis is targeted towards researchers interested in the application of generative neural networks to 3D point clouds, as well as machine learning engineers in the industry whose work involves the manipulation and analysis of 3D point clouds.

1.1 Background

1.1.1 3D generative models

Generative models, such as variational auto-encoders (VAEs) [22], generative adversarial networks (GANs) [13], and normalizing flows [48] , have demonstrated

significant success in the realm of 2D image generation [20, 3, 24]. Consequently, numerous effective 3D generative models have emerged based on these methods, including GANs [29], auto-regressive models [36], and flow-based models [67]. Despite this remarkable progress, these methods retain inherent limitations for modeling point clouds. For example, GANs present training challenges due to the delicate balance required in the adversarial process, where the generator and discriminator networks may struggle to converge or attain a stable equilibrium, leading to training difficulties. Similarly, auto-regressive models, assuming a generation order, can restrict their suitability for tasks requiring thorough data comprehension.

Since 2020, a novel class of generative models has emerged, commonly known as probabilistic diffusion models, denoising diffusion models, or simply diffusion models. These models have demonstrated remarkable capabilities in 2D image generation [18, 42, 9, 40]. In essence, a diffusion model is capable of transforming Gaussian noise into a desired output, such as an authentic image, by gradually removing the Gaussian noise. Approaches like DDPM [17] exemplify this methodology, inherently rooted in probabilistic principles, and have shown the ability to generate genuinely authentic 2D images. Due to their success with 2D images, diffusion models have been extended to generative modeling for 3D shapes [35, 74, 37, 69, 19], including meshes and point clouds.

1.1.2 3D point cloud representation

Unlike 2D images, 3D point clouds are inherently disordered and unstructured. Before delving into methods for learning, it is critical to establish an appropriate data representation. Broadly, the representation for 3D point clouds can be categorized into two primary forms: voxels and points.

3D voxelization is a widely used technique for converting irregular point clouds into regular voxel grids. As described in [64], this method divides the bounding box encapsulating the point cloud into small 3D cuboids using a process called rasterization. Each cuboid, or voxel, represents a small region of space within the bounding box. During voxelization, the algorithm identifies which voxels contain points from the original point cloud. Occupied voxels are retained, while empty voxels are typically discarded to conserve memory. The voxel-based representation capitalizes on sparsity, leveraging the fact that the majority of voxels are often unoccupied.

However, the process of converting 3D points into voxels inevitably results in a certain degree of geometric information loss. To address this issue, an alternative approach retains and directly utilizes the point clouds, thus preserving the integrity of the original 3D data. This approach avoids the inherent loss associated with voxelization and provides a more accurate representation of the intricate geometric details present in the original point cloud.

1.1.3 Models for point cloud learning

As the representation of 3D point clouds can be either voxels or points, two corresponding categories of models for point cloud learning emerge: voxel-based 3D models and point-based 3D models. Moreover, researchers have introduced a point-voxel paradigm for 3D modeling, aiming to combine the advantages of point-based methods and voxel-based methods.

On the one hand, voxel-based 3D models are tailored specifically for processing and learning from 3D voxel data. In previous years, these models often relied on Convolutional Neural Networks (CNNs), such as O-CNN [58], OctNet [50], and kd-Net [26], which incorporated techniques to effectively handle unoccupied voxels. Inspired by the effectiveness and efficiency of sparse convolution on voxel data, researchers have proposed various Transformer-based models designed for 3D voxel data, such as VoTr [39], VoxSeT [14] and SVT-Net [11]. Despite their good performance, these methods may still encounter limitations in capturing intricate geometric details due to information loss during the quantization process.

On the other hand, researchers have developed point-based 3D models that directly process point clouds without resorting to voxelization. PointNet [44] and PointNet++ [45] are pioneering works in the field of learning directly from sparse and unstructured point clouds. Building upon the success of PointNet and PointNet++, subsequent research efforts have introduced sophisticated model architectures aimed at capturing per-point features. In certain methodologies like DCNN [60] and GAC-Net [57] , the point cloud is transformed into a graph, enabling message passing within the graph structure. Alternatively, specific techniques employ continuous convolutions directly on the 3D point set, as exemplified by PCCN [59] and SpiderCNN [63]. Moreover, numerous contemporary methods leverage the capabilities of Transformer-based models, as demonstrated by Point Transformer [72] and Stratified Transformer [28].

Derived from the methodologies of voxel-based and point-based 3D models, a hybrid approach known as the point-voxel paradigm has been introduced. This approach aims to leverage the strengths of both categories and combine their advantages. Models within this category typically comprise a low-resolution voxel-based branch and a high-resolution point-based branch. The voxel-based branch captures coarse-grained neighborhood information, while the point-based branch augments it with fine-grained individual point features. Following this pattern, Point-Voxel CNN [31] and Point-Voxel Transformer [70] are two representative examples, built upon the architectures of CNN and Transformer, respectively.

1.2 Aims and objectives

As mentioned in Section 1.1.1, there has already been some research on applying diffusion models to generative modeling for 3D shapes. Point-Voxel Diffusion (PVD) stands out as a pioneering work in this area. It utilizes the standard DDPM as the

diffusion model and employs a single Point-Voxel CNN as the underlying neural network to predict noise. Much of the more recent research focuses on modifying the architecture of diffusion models to better suit the generation of 3D point clouds. For instance, in [35], the diffusion model is improved by incorporating normalizing flows. In [69], the approach is formulated as a VAE with two latent diffusion models [51]. The neural network models employed in these approaches include PointNet, PointNet++, or Point-Voxel CNN, none of which are Transformer-based models.

Given that the computation underlying Transformer models remains unaffected by the structure or order of input, they are naturally well suited for 3D point cloud learning. Consequently, we aim to investigate the synergy between a diffusion model and a Transformer-based model to potentially yield favorable outcomes in point cloud generation. Before embarking on my master's thesis, this specific avenue of exploration had not yet been pursued. In December 2022, OpenAI unveiled the Point·E method [41], which employs the standard Transformer [56] as the backbone for the diffusion model. However, Point·E uses the Transformer operating at a global scale, which means its Transformer blocks always consider the entire input when attending to different parts of it. In contrast, our project will utilize a Transformer-based model operating at a local scale, where these Transformer blocks extract features from local patches. This approach offers increased efficiency in terms of computation and memory utilization.

To fulfill the aim, we propose Point Transformer Diffusion (PTD), a probabilistic and flexible generative model that integrates DDPM [17], tailored for three-dimensional data, with Point Transformer [72], a self-attention network specifically designed for 3D point clouds. PTD is evaluated on the shape generation task using the same evaluation process as that for PVD [74], to ensure a fair comparison. Since the original Point Transformer model is targeted at tasks such as classification and segmentation, we have also explored the changes that should be made to the Point Transformer model in order to improve its performance on the generation task.

1.3 Contribution

Our main contributions include:

- We propose a novel probabilistic generative model, PTD[1], for 3D point clouds.
- We provide an implementation of Point Transformer suitable for conveniently processing input in batch mode and achieving comparable results to those presented in its paper [72]. [2]
- We prove that the combination of a diffusion model and a local self-attention Transformer is an effective pattern in point cloud generation.
- Through extensive experimentation, our model demonstrates competitive performance in point cloud generation.

[1]The codebase of PTD: https://github.com/jxl152/Point-Transformer-Diffusion
[2]Our implementation of Point Transformer: https://github.com/jxl152/Point-Transformer

1.4 Disposition

The thesis is composed of five chapters. Chapter 1 presents the essential background, states the aims and objectives of the research, and lists my contributions.

Chapter 2 provides an in-depth discussion of the fundamental theories that underpin the methods proposed in this thesis. It consists of two primary components that elucidate DDPM [17] and the self-attention mechanism, respectively.

Chapter 3 formally describes our method: Point Transformer Diffusion. It first formulates the diffusion model in 3D space and then provides a detailed description of Point Transformer [72] that serves as the backbone for the generative process. Subsequently, it introduces several techniques for performance improvement.

Chapter 4 presents the experimental results of the reproduction of PVD [74], the evaluation of Point Transformer, and the evaluation of PTD, each of which is accompanied by analysis and discussion.

Chapter 5 concludes the work with a summary of the contributions and results of this thesis, and presents potential directions for future research.

2

Theory

This chapter covers the essential theories relevant to the method proposed in my Master's thesis. Section 2.2 provides a detailed explanation of Denoising Diffusion Probabilistic Models (DDPM) [17], which is a classical diffusion model. To comprehend DDPM, a foundation in mathematical basics is necessary, as presented in Section 2.1. In the project, a Transformer model [72] is employed to represent the reverse process in the diffusion model, with the self-attention mechanism at its core, as elaborated in Section 2.3. Within the Transformer model, there exist three critical operations applied to 3D point clouds, which will be described in Section 2.4.

2.1 Math basics

2.1.1 The reparameterization trick

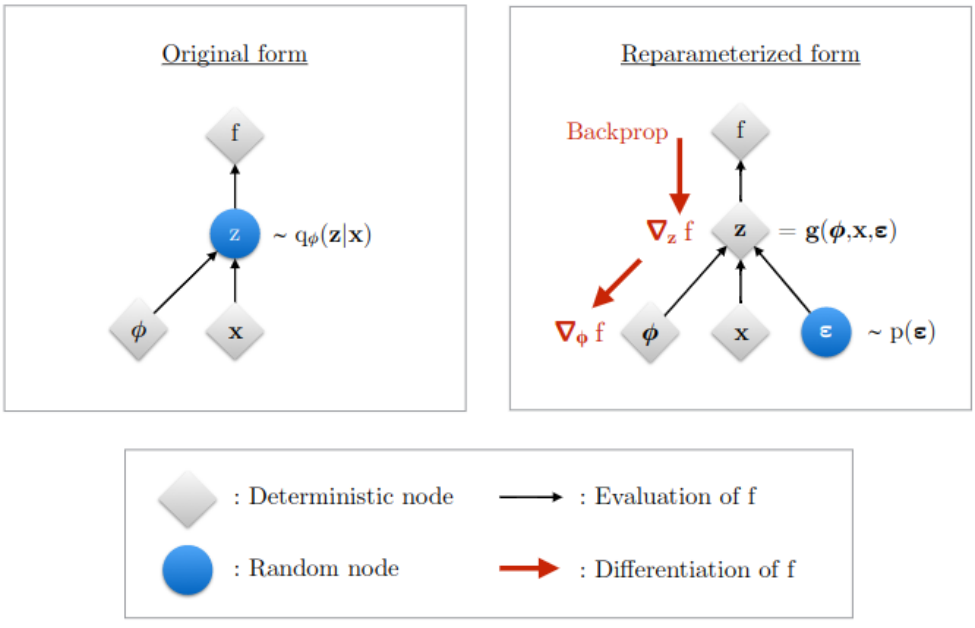


Figure 2.1: Illustration of the reparameterization trick. (Image source: [23])

In Figure 2.1, the original form (left) shows that $\mathbf{z}$ is a continuous random variable sampled from $q_\phi(\mathbf{z}|\mathbf{x})$. The objective f depends on both the input $\mathbf{x}$ and the parameters $\boldsymbol{\phi}$ through $\mathbf{z}$. However, during model training, it is not possible to

backpropagate gradients through the random variable $\mathbf{z}$. Fortunately, the reparameterization trick [22] [49] provides a solution to this problem. In the reparameterized form (right) depicted in Figure 2.1, the random variable $\mathbf{z}$ is expressed as a deterministic function: $\mathbf{z} = g(\boldsymbol{\phi}, \mathbf{x}, \boldsymbol{\epsilon})$, where $\boldsymbol{\epsilon}$ is a newly introduced random variable independent of $\boldsymbol{\phi}$ and $\mathbf{x}$. This reformulation enables backpropagation through $\mathbf{z}$ and the computation of the gradients $\nabla_\phi f$.

For instance, consider the multivariate Gaussian distribution with a diagonal covariance matrix: $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}_\phi(\mathbf{x})^2\mathbf{I})$. To leverage the reparameterization trick, we compute $\mathbf{z}$ as a deterministic function of the input $\mathbf{x}$ and an auxiliary noise variable $\boldsymbol{\epsilon}$:

$$\mathbf{z} = \boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon} \quad \text{, where } \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I}). \tag{2.1}$$

Here, $\odot$ denotes the element-wise product. It is important to note that we will employ this reparameterization technique to derive the closed form of $q(\mathbf{x}_t|\mathbf{x}_0)$ in Section 2.2.1 and design the sampling algorithm for diffusion models in Section 2.2.6.

2.1.2 KL divergence

The Kullback–Leibler divergence [27], also known as KL divergence and denoted as $D_{\mathrm{KL}}(P \,||\, Q)$, is a statistical distance used to quantify the dissimilarity between two probability distributions: P and Q. Eq. (2.2) demonstrates how to calculate the KL divergence for discrete probability distributions, while Eq. (2.3) provides the definition for continuous probability distributions.

$$D_{\mathrm{KL}}(P \,||\, Q) = \sum_{x \in \mathcal{X}} p(x) \log \left(\frac{p(x)}{q(x)} \right), \tag{2.2}$$

$$D_{\mathrm{KL}}(P \,||\, Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx. \tag{2.3}$$

To comprehend this thesis, it is essential to grasp two key aspects of the KL divergence. Firstly, the KL divergence is inherently non-negative, denoted as $D_{\mathrm{KL}}(P \,||\, Q) \geq 0$, which can be proven through Gibbs' inequality [38]. Equality is achieved only when the probability distributions P and Q are identical. This property will be employed in deriving the evidence lower bound in Section 2.1.3.

Secondly, when dealing with multivariate Gaussian distributions, calculating the KL divergence is straightforward. Consider two multivariate Gaussian distributions with means $\boldsymbol{\mu}_0$, $\boldsymbol{\mu}_1$, and covariance matrices $\boldsymbol{\Sigma}_0$, $\boldsymbol{\Sigma}_1$, all having the same dimension d. In such cases, the KL divergence between the distributions can be computed using the following formula:

$$D_{\mathrm{KL}}(\mathcal{N}_0 \,||\, \mathcal{N}_1) = \frac{1}{2} \left[\log \frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_0|} - d + \mathrm{tr}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}_1^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) \right]. \tag{2.4}$$

This formula will be used in the computation of the denoising matching term L_{t-1} in Section 2.2.4.

2.1.3 Evidence lower bound

The evidence lower bound, commonly known as ELBO, serves as the optimization objective for variational autoencoders [22]. In essence, it allows for the approximation of the intractable posterior distribution by optimizing a lower bound on the log-likelihood of the data.

Suppose we have a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ consisting of N i.i.d. samples, and we assume that $\mathbf{x}^{(i)}$ is generated by a random process with an unknown true probability distribution $p^*(\mathbf{x})$. Since $p^*(\mathbf{x})$ is unknown, we utilize a model $p_\theta(\mathbf{x})$ parameterized by $\boldsymbol{\theta}$ as an approximation, aiming for $p_\theta(\mathbf{x})$ to closely approximate $p^*(\mathbf{x})$ for any observed variable $\mathbf{x}$. Under the i.i.d. assumption, the likelihood of the dataset given the parameters can be factorized as the product of the probabilities of each sample. The factorization allows us to express the corresponding log-likelihood as follows:

$$\log p_\theta(\mathcal{D}) = \sum_{\mathbf{x}\in\mathcal{D}} \log p_\theta(\mathbf{x}). \tag{2.5}$$

The optimal parameter $\boldsymbol{\theta}^*$ is the one that maximizes the log-likelihood of the data:

$$\theta^* = \arg\max_\theta \sum_{\mathbf{x}\in\mathcal{D}} \log p_\theta(\mathbf{x}). \tag{2.6}$$

For fully-observed data, optimizing models such as $p_\theta(\mathbf{x})$ towards the maximum likelihood or log-likelihood objective is straightforward [23]. However, in many cases, especially when the dataset $\mathcal{D}$ is complex, it becomes necessary to learn a more flexible model with latent variables typically denoted by $\mathbf{z}$. This enables the model to represent a joint distribution $p_\theta(\mathbf{x}, \mathbf{z})$ that encompasses both the observed data and the latent variables.

Theoretically, we can calculate the marginal likelihood $p_\theta(\mathbf{x})$ from the joint distribution $p_\theta(\mathbf{x}, \mathbf{z})$ in two ways [34]. The first approach involves explicitly marginalizing out the latent variable $\mathbf{z}$ through integration, as shown in Eq. (2.7). Unfortunately, this method is intractable due to the integral operation. An alternative method relies on the chain rule of probability, as illustrated in Eq. (2.8). However, the presence of the unknown denominator $p_\theta(\mathbf{z}|\mathbf{x})$ renders this approach impractical.

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z})\, d\mathbf{z}, \tag{2.7}$$

$$p_\theta(\mathbf{x}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}. \tag{2.8}$$

To solve the aforementioned problems, we can introduce $q_\phi(\mathbf{z}|\mathbf{x})$, a flexible variational distribution parameterized by $\boldsymbol{\phi}$, as an approximation to the intractable true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ [22]. Then we can begin to derive the ELBO as follows:

$$\log p_\theta(\mathbf{x}) = \log p_\theta(\mathbf{x}) \int q_\phi(\mathbf{z}|\mathbf{x})\, d\mathbf{z} \tag{2.9}$$

$$= \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x})\, d\mathbf{z} \tag{2.10}$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x})] \tag{2.11}$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \tag{2.12}$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})q_\phi(\mathbf{z}|\mathbf{x})}\right] \tag{2.13}$$

$$= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\substack{\mathcal{L}_{\theta,\phi}(\mathbf{x}) \\ \text{(ELBO)}}} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]}_{D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\,||\,p_\theta(\mathbf{z}|\mathbf{x}))}. \tag{2.14}$$

In Eq. (2.14), the second term corresponds to the KL divergence between $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z}|\mathbf{x})$, which is always non-negative. This non-negativity property ensures that the first term $\mathcal{L}_{\theta,\phi}(\mathbf{x})$, known as the ELBO, serves as a lower bound on the evidence $\log p_\theta(\mathbf{x})$, as shown in Eq. (2.15).

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\,||\,p_\theta(\mathbf{z}|\mathbf{x})) \\ &\leq \log p_\theta(\mathbf{x}).\end{aligned} \tag{2.15}$$

Maximizing the ELBO serves as a proxy objective for optimizing a latent variable model [22]. The discrepancy between the ELBO and the evidence is quantified by the KL divergence $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\,||\,p_\theta(\mathbf{z}|\mathbf{x}))$. The closer the approximation $q_\phi(\mathbf{z}|\mathbf{x})$ comes to the true posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$, the smaller the discrepancy becomes. In the ideal scenario, if and only if $q_\phi(\mathbf{z}|\mathbf{x})$ perfectly matches $p_\theta(\mathbf{z}|\mathbf{x})$, the ELBO and the evidence become exactly equivalent.

2.2 Diffusion models

Diffusion models are a class of state-of-the-art generative models that have gained significant attention in recent years. They are a category of deep learning models specifically designed to capture complex dependencies in high-dimensional data, such as images, text, or audio. Several generative models based on diffusion have been put forward, sharing common underlying principles. In this thesis, we will focus on the most notable one, DDPM, which was initially introduced by Sohl-Dickstein et al. [54] and subsequently expanded upon by Ho. et al [17].

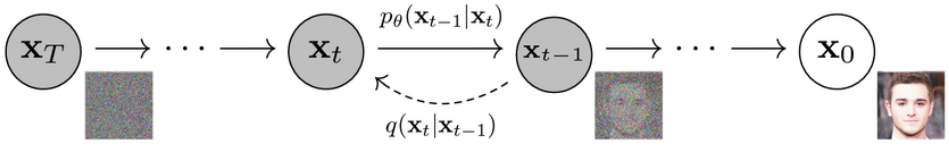


Figure 2.2: The directed graphical model illustrates the forward and reverse processes involved in a diffusion model. (Image source: [17])

The primary objective of a diffusion model is to model a specific distribution from random noise. Thus, the generated samples should closely approximate the original

samples in terms of their distributions. To this end, a diffusion model comprises of a forward process, where data is gradually subjected to noise, and a reverse process, where the noise is iteratively transformed back into a sample from the target distribution. In Figure 2.2, the dashed line annotated with $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ illustrates a step in the forward process, transitioning from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$, while the solid line annotated with $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ depicts a step in the reverse process, moving from $\mathbf{x}_t$ back to $\mathbf{x}_{t-1}$.

2.2.1 Forward process

Given a data point $\mathbf{x}_0$ sampled from a specific data distribution $q(\mathbf{x})$, the forward process is formally defined as a discrete-time Markov chain. This process progressively introduces Gaussian noise to the data, following a predefined variance schedule $\beta_1, ..., \beta_T$, resulting in a sequence of noisy samples $\mathbf{x}_1, ..., \mathbf{x}_T$. As shown in Figure 2.2, the data point $\mathbf{x}_t$ progressively diminishes its distinguishable features, with the step t increasing. Ultimately, $\mathbf{x}_T$ is nearly an isotropic Gaussian distribution.

Since the forward process is a Markov chain, the prediction of the probability density at time t is solely based on the preceding value at time $t-1$. Thus, the conditional probability density can be computed as follows:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}). \tag{2.16}$$

The conditional probability distribution of the whole process can then be calculated using the following formula:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}). \tag{2.17}$$

An important characteristic of the forward process is that we can sample x_t at any arbitrary timestep t in closed form. Let $\alpha_t := 1-\beta_t$, $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$, $\boldsymbol{\epsilon}_0, ..., \boldsymbol{\epsilon}_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\bar{\boldsymbol{\epsilon}}_{t-2}, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Based on Eq. (2.16), we use the reparameterization trick in a recursive manner:

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}_{t-2}\right) + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t - \alpha_t\alpha_{t-1}}\boldsymbol{\epsilon}_{t-2} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} \\
&= \cdots \\
&= \sqrt{\alpha_t \cdots \alpha_1}\mathbf{x}_0 + \sqrt{1-\alpha_t \cdots \alpha_1}\boldsymbol{\epsilon} \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}.
\end{aligned} \tag{2.18}$$

Therefore, to produce a sample $\mathbf{x}_t$, we can use the following formula:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}). \tag{2.19}$$

2.2.2 Reverse process

Ideally, if we know the reverse distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, we can start with a noise input $\mathbf{x}_T$ sampled from the Gaussian distribution $\mathcal{N}(0, \mathbf{I})$, run the reverse process, and eventually generate a novel data point that should conform to the original data distribution $q(\mathbf{x}_0)$. However, we cannot easily estimate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ because it requires the use of the entire dataset as follows:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t) = \frac{p(\mathbf{x}_{t-1}, \mathbf{x}_t|\mathcal{D})}{p(\mathbf{x}_t|\mathcal{D})}, \tag{2.20}$$

where $\mathcal{D}$ is the complete dataset. The dependency makes the computation of $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ intractable as the dataset size increases.

In practice, the reverse process utilizes an approximation method to estimate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ without explicitly involving the entire dataset. More specifically, it employs a neural network model p_θ to approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. The Gaussian transitions $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to be learned are parameterized with their mean and variance as follows:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)). \tag{2.21}$$

The joint distribution characterizing the whole reverse process, which starts at $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$, can then be calculated using the following formula:

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t). \tag{2.22}$$

2.2.3 Objective function

To ensure that the generated samples closely resemble the original samples in terms of their distribution, it is essential for $p_\theta(\mathbf{x}_0)$ to closely approximate the true distribution of any observed variable $\mathbf{x}_0$. Therefore, $\log p_\theta(\mathbf{x}_0)$ is a candidate as the objective function.

As discussed in Section 2.1.3, there are several challenges in optimizing $\log p_\theta(\mathbf{x}_0)$. Hence, it is preferable to resort to its lower bound, known as the evidence lower bound (ELBO). Following the derivation from Eq. (2.9) to (2.14), we can infer the ELBO for $\log p_\theta(\mathbf{x}_0)$:

$$\log p_\theta(\mathbf{x}_0) \geq \underbrace{\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right]}_{\text{ELBO}}. \tag{2.23}$$

Then, we can assign the loss function to the negative of the ELBO as follows:

$$-\log p_\theta(\mathbf{x}_0) \leq -\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] := \mathcal{L}. \tag{2.24}$$

To make the loss function $\mathcal{L}$ computationally tractable, we decompose it into T terms, with each term corresponding to a timestep, as follows:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \\ &= \mathbb{E}_q\left[-\log \frac{p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{\prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \\ &= \mathbb{E}_q\left[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\,||\,p(\mathbf{x}_T))}_{L_T} + \sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\,||\,p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}}\underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\right].\end{aligned} \tag{2.25}$$

The detailed derivation of this decomposition can be found in Eq. (A.1).

2.2.4 Analysis of L_T, $L_{1:T-1}$, and L_0

$\mathbf{L_T}$. The L_T term represents how close $\mathbf{x}_T$ is to the standard Gaussian. Since it has no learnable parameters, L_T can be ignored during training.

$\mathbf{L_{1:T-1}}$. $L_{t-1}(1 < t \leq T)$ represents the denoising matching term. It involves learning the desired denoising transition step $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ as an approximation to the tractable ground-truth denoising transition step $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$. Minimizing L_{t-1} is equivalent to aligning the two denoising steps as closely as possible, which is measured by their KL divergence.

More specifically, $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ is a Gaussian distribution, as shown in Eq. (2.26), with its detailed derivation presented in Eq. (A.2) in Appendix A.

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1};\boldsymbol{\mu}_q(\mathbf{x}_t,\mathbf{x}_0),\tilde{\beta}_t\mathbf{I}), \tag{2.26}$$

$$\text{where } \boldsymbol{\mu}_q(\mathbf{x}_t,\mathbf{x}_0) := \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 \tag{2.27}$$

$$\text{and } \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \tag{2.28}$$

Similarly, $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is also a Gaussian distribution, as shown in Eq. (2.21). It is worth noting that in practice, DDPM set $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t,t)$, the variance of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, to time-dependent constants $\sigma_t^2\mathbf{I}$. Thus, we have:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1};\boldsymbol{\mu}_\theta(\mathbf{x}_t,t),\sigma_t^2\mathbf{I}). \tag{2.29}$$

Therefore, L_{t-1} is the KL divergence between two multivariate Gaussian distributions. We can utilize Eq. (2.4) to compute L_{t-1}, and the resulting expression is as follows:

$$\begin{aligned}L_{t-1} &= D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\,||\,p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \\ &= D_{\mathrm{KL}}(\mathcal{N}(\mathbf{x}_{t-1};\boldsymbol{\mu}_q(\mathbf{x}_t,\mathbf{x}_0),\tilde{\beta}_t\mathbf{I})\,||\,\mathcal{N}(\mathbf{x}_{t-1};\boldsymbol{\mu}_\theta(\mathbf{x}_t,t),\sigma_t^2\mathbf{I})) \\ &\propto \frac{1}{2\sigma_t^2}\left|\left|\boldsymbol{\mu}_q(\mathbf{x}_t,\mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t,t)\right|\right|^2.\end{aligned} \tag{2.30}$$

The detailed derivation of Eq. (2.30) can be found in Eq. (A.6) in Appendix A.

Hence, it is evident that a simple and direct parameterization of $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ is a model that predicts $\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$. Nevertheless, DDMP further extended Eq. (2.30) with additional developments. Based on the expression for $q(\mathbf{x}_t|\mathbf{x}_0)$, we can rearrange Eq. (2.18) to yield:

$$\mathbf{x}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}{\sqrt{\bar{\alpha}_t}}. \tag{2.31}$$

By substituting this into Eq. (2.27), we can rederive $\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$ as:

$$\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}{\sqrt{\bar{\alpha}_t}}\right) \tag{2.32}$$

$$= \left(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} + \frac{\beta_t}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}}\right)\mathbf{x}_t - \frac{\beta_t\sqrt{1-\bar{\alpha}_t}}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}}\boldsymbol{\epsilon} \tag{2.33}$$

$$= \left(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} + \frac{1-\alpha_t}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}}\right)\mathbf{x}_t - \frac{(1-\alpha_t)\sqrt{1-\bar{\alpha}_t}}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}}\boldsymbol{\epsilon} \tag{2.34}$$

$$= \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\boldsymbol{\epsilon}. \tag{2.35}$$

Similarly, we can establish the approximate denoising transition mean $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ as:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t), \tag{2.36}$$

where $\boldsymbol{\epsilon}_\theta$ is a function approximator utilized to predict $\boldsymbol{\epsilon}$ based on $\mathbf{x}_t$.

Lastly, we substitute the equations (2.35) and (2.36) into Eq. (2.30):

$$L_{t-1} = \frac{1}{2\sigma_t^2}\left\|\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\right\|^2 \tag{2.37}$$

$$= \frac{1}{2\sigma_t^2}\left\|\frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\boldsymbol{\epsilon} - \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t + \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right\|^2 \tag{2.38}$$

$$= \frac{(1-\alpha_t)^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right\|^2. \tag{2.39}$$

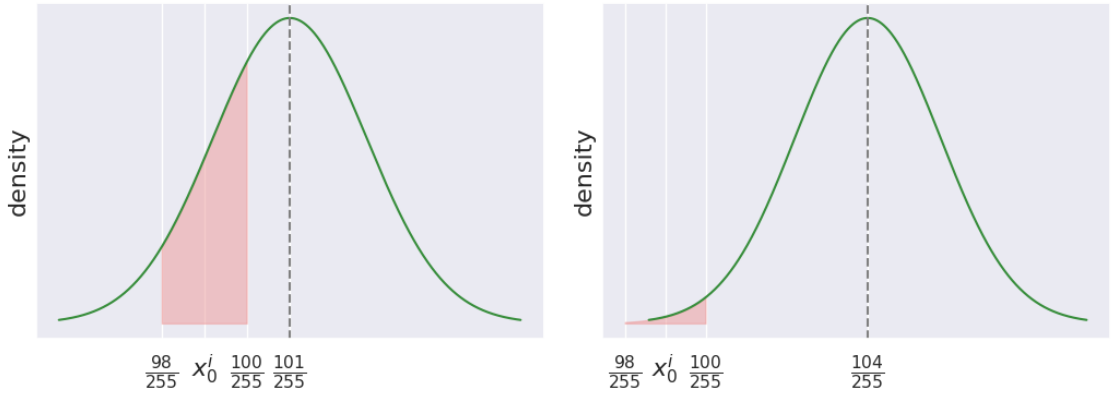
$\mathbf{L_0}$. The L_0 term can be interpreted as a reconstruction term [34]. In DDPM [17], all the work was conducted on image data consisting of integers ranging from 0 to 255. These integers were linearly scaled to the range of -1 to 1. In order to obtain discrete log likelihoods, [17] defined $p_\theta(\mathbf{x}_0|\mathbf{x}_1)$ as an independent discrete decoder derived from the Gaussian distribution $\mathcal{N}(\mathbf{x}_0; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1), \sigma_1^2\mathbf{I})$:

$$p_\theta(\mathbf{x}_0|\mathbf{x}_1) = \prod_{i=1}^{d}\int_{\delta_-(x_0^i)}^{\delta_+(x_0^i)} \mathcal{N}(x; \mu_\theta^i(\mathbf{x}_1, 1), \sigma_1^2)\, dx \tag{2.40}$$

$$\delta_+(x) = \begin{cases} \infty & \text{if } x = 1 \\ x + \frac{1}{255} & \text{if } x < 1 \end{cases} \tag{2.41}$$

$$\delta_-(x) = \begin{cases} -\infty & \text{if } x = -1 \\ x - \frac{1}{255} & \text{if } x > -1 \end{cases}, \tag{2.42}$$

where d represents the number of pixels, and the superscript i signifies the extraction of a specific pixel. The integral is calculated over the range $[\delta_-(x_0^i), \delta_+(x_0^i)]$, which captures the vicinity of the actual pixel value x_0^i. When the predicted mean, $\mu_\theta^i(\mathbf{x}_1, 1)$, closely matches the true pixel value, the integral yields a higher value. If this holds true for all pixels, the product will result in a high value, indicating a high probability $p_\theta(\mathbf{x}_0|\mathbf{x}_1)$. Conversely, when the predicted mean consistently deviates from the true value, the concentration of mass in the true pixel area decreases, leading to a lower overall probability. To enhance comprehension, Figure 2.3 provides a visual example.



(a) $\mu_\theta^i(\mathbf{x}_1, 1) = \frac{101}{255}$ **(b)** $\mu_\theta^i(\mathbf{x}_1, 1) = \frac{104}{255}$

Figure 2.3: Illustration of the discrete decoder designed for L_0. In this example, the actual value of the i_{th} pixel, x_0^i, is $\frac{99}{255}$. Using equations (2.42) and (2.41), we can determine the interval $[\delta_-(x_0^i), \delta_+(x_0^i)]$ as $[\frac{98}{255}, \frac{100}{255}]$. In (a), the predicted mean $\mu_\theta^i(\mathbf{x}_1, 1)$ is equal to $\frac{101}{255}$, which is closer to x_0^i compared to (b) where $\mu_\theta^i(\mathbf{x}_1, 1)$ is $\frac{104}{255}$. Consequently, the probability in (a) is significantly greater than that in (b).

2.2.5 Simplified training objective

The ELBO, as shown in Eq. (2.25), comprises terms derived from Eq. (2.39) and Eq. (2.40). The ELBO is differentiable with respect to $\boldsymbol{\theta}$ and can thus be readily used for training. Nevertheless, DDPM made further simplifications to the objective, resulting in the following expression:

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left|\left|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right|\right|^2\right] \tag{2.43}$$

$$= \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left|\left|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right|\right|^2\right]. \tag{2.44}$$

Here, where t is sampled from a uniform distribution between 1 and T. The case when $t = 1$ replaces the independent discrete decoder demonstrated by Eq. (2.40), which simplifies the implementation. For cases where $t > 1$, they correspond to an unweighted version of Eq. (2.39).

2.2.6 Algorithms

Finally, we will examine the training and sampling algorithms for DDPMs, which can be considered a summary of the content covered in Sections 2.2.1 to 2.2.5.

Algorithm 1 Training [17]

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, ..., T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 1 outlines the training procedure. Firstly, we sample a set of data $\mathbf{x}_0$ from the training dataset. Next, we sample t from a uniform distribution that spans the range between 1 and T. Additionally, we generate noise by sampling from a normal distribution. Following these initial steps, we proceed to optimize our loss function, as defined by Eq. (2.44), utilizing the gradient descent algorithm.

Algorithm 2 Sampling [17]

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, ..., 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Algorithm 2 presents an overview of the sampling process. We begin by sampling $\mathbf{x}_T$ from a normal distribution. Then, in an iterative manner, we utilize the reparameterization trick to sample $\mathbf{x}_{t-1}$, as indicated in the 4th line of the algorithm. The formula used for sampling is based on Eq. (2.29), where $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ is expressed by the Eq. (2.36). Ultimately, this iterative process leads us to obtain $\mathbf{x}_0$ as the final output.

It is worth noting that noise is not added when predicting $\mathbf{x}_0$ given $\mathbf{x}_1$. Specifically, when $t = 1$, the neural network predicts the noise $\boldsymbol{\epsilon}_\theta(\mathbf{x}_1, 1)$ within $\mathbf{x}_1$, which is then subtracted from $\mathbf{x}_1$ to obtain $\mathbf{x}_0$. As we have reached the final generation result, it is not advisable to introduce additional noise. The inclusion of additional noise would adversely impact the quality of $\mathbf{x}_0$ and compromise the overall outcome. Therefore, we refrain from adding noise at this stage to ensure the integrity and fidelity of the generated result.

2.3 Self-attention mechanism

The Transformer is a deep learning model architecture proposed by Vaswani et al [56]. Over the years, Transformers have revolutionized natural language processing [8] and have demonstrated outstanding performance across diverse domains, including 2D image analysis [10, 16, 30] and 3D point cloud processing [43, 72, 28]. At the core of these highly successful Transformers lies the self-attention mechanism.

Section 2.3.1 provides a detailed explanation of the fundamental concept underlying self-attention. Subsequently, Section 2.3.2 introduces the scaled dot-product attention mechanism, which was proposed and utilized in the standard Transformer [56]. Self-attention operators can be broadly categorized into two types: scalar attention [56] and vector attention [71], as described in Sections 2.3.3 and 2.3.4, respectively. It is worth noting that the Transformer-based model employed in our project is based on vector attention.

2.3.1 A basic version of self-attention

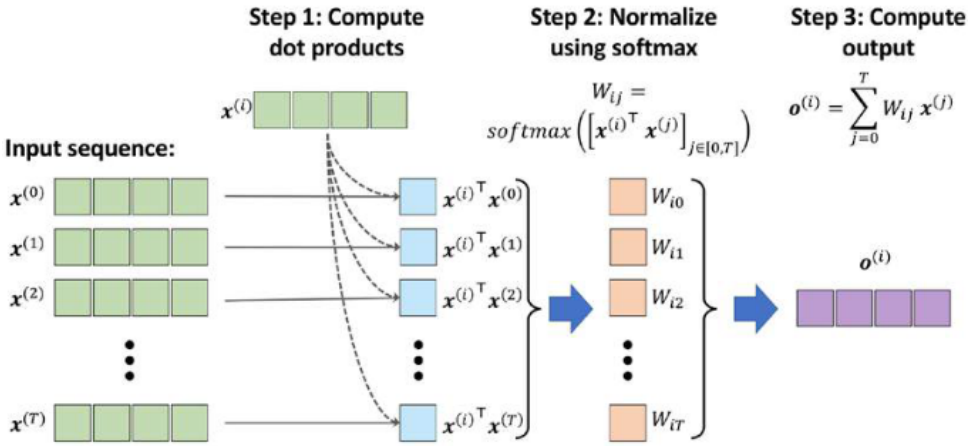


Figure 2.4: Illustration of a basic version of self-attention. Given a specific input element $\mathbf{x}^{(i)}$, we iterate over every element $\mathbf{x}^{(j)}$ in the input sequence. For each pair, we compute their dot product $\mathbf{x}^{(i)^T}\mathbf{x}^{(j)}$, which represents the similarity between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$. Subsequently, we normalize the dot products using the softmax function [4], obtaining the weight W_{ij}. The softmax normalization ensures that the weights will sum to 1, i.e., $\sum_{j=0}^{T} W_{ij} = 1$. In the final step, to generate the output, $\mathbf{o}^{(i)}$, we calculate the weighted sum of the entire input sequence using the computed weights, as expressed in Eq. (2.45). (Image source: [47])

To elucidate the fundamental concept underlying self-attention, we consider an input sequence, $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)}$, of length T, alongside the corresponding output sequence, $\mathbf{o}^{(0)}, \mathbf{o}^{(1)}, ..., \mathbf{o}^{(T)}$. Each element in these sequences, denoted as $\mathbf{x}^{(t)}$ or $\mathbf{o}^{(t)}$, represents a d-dimensional vector (i.e., $\mathbf{x}^{(t)}, \mathbf{o}^{(t)} \in \mathbb{R}^d$).

The primary objective of self-attention is to capture and model the interdependencies between each element in the output sequence and all elements in the input sequence. To achieve this, self-attention computes a weighted sum of the input sequence, where the weights determine the importance of each input element. Specifically, for the ith input element, the corresponding output value, denoted as $\mathbf{o}^{(i)}$, is computed as the weighted sum of all input elements according to the equation:

$$\mathbf{o}^{(i)} = \sum_{j=0}^{T} W_{ij}\mathbf{x}^{(j)}, \tag{2.45}$$

which is accomplished through three distinct steps, as illustrated in Figure 2.4.

2.3.2 Scaled dot-product attention

The basic self-attention mechanism does not incorporate any learnable parameters in the computation of the output values. If a model relies solely on this basic self-attention, it may have limitations in terms of its ability to update or modify the attention values during the optimization process for a given sequence.

To address this limitation, the standard Transformer [56] introduces a specific attention mechanism known as "scaled dot-product attention". This attention mechanism offers increased flexibility and better adaptability for model optimization. It achieves this by utilizing three weight matrices, denoted as $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$, which serve as trainable parameters during the model's training process. These weight matrices are used to project the input embeddings into three distinct matrices: query, key, and value. This transformation enables the model to capture meaningful relationships and dependencies between different elements of the input sequence.

Specifically, consider $\mathbf{X} = (\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)})$ as the stack of every input embedding $\mathbf{x}^{(t)} \in \mathbb{R}^d$, which makes $\mathbf{X}$ a $T \times d$ matrix. The query, key, and value matrices can be generated using the following procedure:

- The query $\mathbf{Q} = \mathbf{X}\mathbf{W}_q$. Here, $\mathbf{W}_q$ is a $d \times d_q$ matrix, resulting in $\mathbf{Q}$ being a $T \times d_q$ matrix.
- The key $\mathbf{K} = \mathbf{X}\mathbf{W}_k$. Here, $\mathbf{W}_k$ is a $d \times d_k$ matrix, resulting in $\mathbf{K}$ being a $T \times d_k$ matrix. Additionally, the scaled dot-product attention follows $d_q = d_k$.
- The value $\mathbf{V} = \mathbf{X}\mathbf{W}_v$. Here, $\mathbf{W}_v$ is a $d \times d_v$ matrix, resulting in $\mathbf{V}$ being a $T \times d_v$ matrix.

After the linear projection, the scaled dot-product attention mechanism computes the matrix of outputs as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V}, \tag{2.46}$$

which is visualized by Figure 2.5.

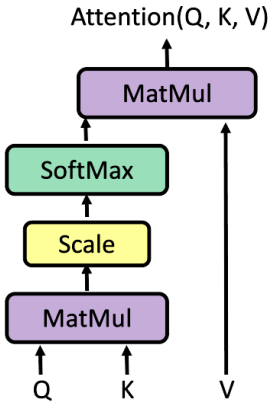


Figure 2.5: Illustration of scaled dot-product attention. First, the scaled dot-product attention mechanism computes the matrix multiplication between $\mathbf{Q}$ and $\mathbf{K}$, resulting in the product $\mathbf{QK}^T$. This product is then scaled by $\frac{1}{\sqrt{d_k}}$. Next, the scaled product is normalized using the softmax function, producing the normalized weight matrix. Finally, the attention is obtained by performing the matrix multiplication between the normalized weight matrix and the value matrix $\mathbf{V}$.

The scaled dot-product attention, built upon the dot-product attention mechanism, incorporates an additional scaling factor of $\frac{1}{\sqrt{d_k}}$. When d_k is a large value, certain dot products within $\mathbf{QK}^T$ can significantly increase in magnitude, pushing the softmax function into situations where it has very small gradients. To mitigate this effect, a scaling factor of $\frac{1}{\sqrt{d_k}}$ is applied before normalizing the dot products. This scaling operation helps maintain a balance between different dimensions and prevents any single dimension from dominating the attention computation.

2.3.3 Scalar attention

In scalar attention, the attention weights are computed by taking the dot product between the query and key vectors, followed by a softmax function for normalization. This type of attention assigns a single scalar weight to each key vector. Accordingly, the scaled dot-product attention falls under the category of scalar attention.

Consider a set of feature vectors $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$. The scalar dot-product attention layer is represented by the following equation:

$$\mathbf{y}^{(i)} = \sum_{\mathbf{x}^{(j)} \in \mathbf{X}} \rho(\varphi(\mathbf{x}^{(i)})^T \psi(\mathbf{x}^{(j)}) + \delta)\alpha(\mathbf{x}^{(j)}). \tag{2.47}$$

Here, $\mathbf{y}^{(i)}$ represents the output feature. The functions φ, ψ, and α perform point-wise feature transformations, such as linear projections or Multi-Layer Perceptrons (MLPs). Additionally, δ is a position encoding function, while ρ signifies a normalization function.

In this way, we can interpret the scaled dot-product attention in a more general way. Specifically, the weight matrices $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ correspond to the functions φ, ψ,

and α, respectively. Therefore, the query matrix $\mathbf{Q}$ corresponds to $\varphi(\mathbf{x}^{(i)})$, the key matrix $\mathbf{K}$ corresponds to $\psi(\mathbf{x}^{(j)})$, and the value matrix $\mathbf{V}$ corresponds to $\alpha(\mathbf{x}^{(j)})$. Additionally, the softmax function corresponds to the normalization function ρ.

Although there are two differences between Eq. (2.46) and Eq. (2.47), they can be explained. The first difference is that Eq. (2.46) uses a scaling factor of $\frac{1}{\sqrt{d_k}}$. This is because Eq. (2.47) represents a "scalar dot-product attention" layer without considering the scaling, but it can be easily extended to a version of "scalar scaled dot-product attention" by incorporating a scaling factor before normalization. The second difference is that Eq. (2.47) includes a position encoding function δ. In fact, the positional encoding is utilized in the standard Transformer to capture sequential information within the self-attention mechanism. Since the positional encoding is directly added to the input embeddings, it is not explicitly specified in Eq. (2.46). In comparison, the explicit mention of positional encoding in Eq. (2.47) provides a clearer indication of its usage.

2.3.4 Vector attention

In vector attention [71], the computation of attention weights differs from scalar attention. Instead of producing a single scalar weight, vector attention generates a weight vector that matches the dimension of the value vectors. Each element of the weight vector represents the attention weight for a specific dimension or feature of the value vectors. Consequently, vector attention enables the capture of more fine-grained information about the importance of different dimensions or features.

We can represent a vector attention layer using the equation:

$$\mathbf{y}^{(i)} = \sum_{\mathbf{x}^{(j)} \in \mathcal{X}} \rho\Big(\gamma(\beta(\varphi(\mathbf{x}^{(i)}), \psi(\mathbf{x}^{(j)})) + \delta)\Big) \odot \alpha(\mathbf{x}^{(j)}). \tag{2.48}$$

In this equation, β represents a relation function. Unlike scalar attention, where the dot product between $\varphi(\mathbf{x}^{(i)})$ and $\psi(\mathbf{x}^{(j)})$ is always performed, the vector attention mechanism offers more flexibility by providing alternative operations such as subtraction. Additionally, γ is a mapping function, such as an MLP, that generates attention vectors for the purpose of feature aggregation.

2.4 Algorithms for 3D point clouds

Neural network models leverage numerous necessary and useful algorithms or operations for processing 3D point clouds. In our project, the Transformer-based model utilizes three key algorithms: k-nearest neighbors, farthest point sampling, and inverse distance weighting. These algorithms play crucial roles in enabling the model to effectively process and extract features from 3D point cloud data.

2.4.1 K-nearest neighbors

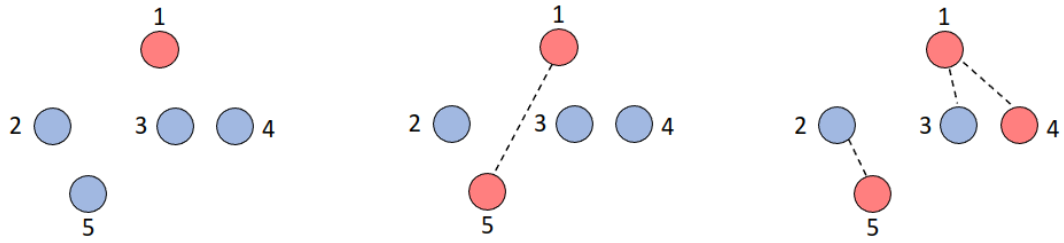
In statistics, the k-nearest neighbors algorithm, also known as kNN, is a widely used non-parametric supervised learning method. This versatile algorithm is applied in numerous tasks, including classification and regression, with the primary goal of identifying the k nearest neighbors of a given query point.

The utilization of kNN varies depending on the task at hand. In this section, we will focus on the algorithm of using kNN in our project. With regard to 3D point clouds, we assume that each point cloud as a set of N points with xyz-coordinates, denoted by $\mathbf{x} \in \mathbb{R}^{N\times 3}$. For the ith point, $\mathbf{x}^i = (\mathbf{x}_1^i, \mathbf{x}_2^i, \mathbf{x}_3^i)$, we can find its k nearest neighbors as follows:

1. Calculate the Euclidean distance between $\mathbf{x}^i$ and each data point in the dataset:
$$d_{ij} = ||\mathbf{x}^i - \mathbf{x}^j||, \tag{2.49}$$
which represents the distance between $\mathbf{x}^i$ and $\mathbf{x}^j$.
2. Sort the distances in ascending order.
3. Select the k data points with the smallest distances. These are the k nearest neighbors of $\mathbf{x}^i$ in the dataset.

2.4.2 Farthest point sampling

There are many algorithms for downsampling 3D point clouds, such as random sampling, voxel grid downsampling, and farthest point sampling (FPS). In this section, we will focus on the theory of the FPS algorithm.



(a) 1st sampled point **(b)** 2nd sampled point **(c)** 3rd sampled point

Figure 2.6: An example of farthest point sampling. Initially, the FPS algorithm randomly selects a point. As shown in (a), the point labeled 1 is chosen as the 1st sampled point. At this stage, the set *sampled* $= \{1\}$, and the set *remaining* $= \{2, 3, 4, 5\}$. For every point in *remaining*, its nearest neighbor is the point labeled 1. Thus, the distances between all the points in *remaining* and their nearest neighbor in *sampled* are denoted as d_{21}, d_{31}, d_{41}, and d_{51}, where d_{ij} represents the distance between the point labeled i and the point labeled j. Among these distances, d_{51} is the largest, so the point labeled 5 is selected as the 2nd sampled point, as shown in (b). Now, the set *sampled* $= \{1, 5\}$, and the set *remaining* $= \{2, 3, 4\}$. For the points in *remaining*, their nearest neighbors in *sampled* are 5, 1, and 1, respectively. Among the distances of d_{25}, d_{31}, and d_{41}, d_{41} is the largest, so the point labeled 4 is selected as the 3rd sampled point, as shown in (c).

Consider a point cloud consisting of N points. To obtain a subset of S points using the FPS algorithm, we need to recursively select a point until we have S samples. First, we can define two sets: *sampled*, which includes the selected points, and *remaining*, which includes the points not yet selected. Then, we can choose a point as follows:

1. For each point in the *remaining* set, find its nearest neighbor in the *sampled* set and calculate their distance. Record the distances of all such pairs.
2. Select the point from the *remaining* set with the largest distance to its nearest neighbor in the *sampled* set, and move it to the *sampled* set.

Figure 2.6 depicts a simple example showcasing the utilization of the FPS algorithm. This illustrative example demonstrates how the FPS algorithm is applied to select 3 points from a set of 5 points.

2.4.3 Inverse distance weighting

Encoder-decoder architectures, such as U-Net [52], primarily consist of two paths: a contracting path, known as the encoder, and an expansion path, known as the decoder. Since the encoder downsamples the input data and compresses its spatial information, the decoder needs to perform upsampling to reconstruct the resolution back to the original data size. Interpolation is employed to achieve this upsampling in the decoder. It plays a key role in enhancing the spatial resolution of the feature maps, restoring them to the original data size, which is essential for accurate segmentation. Various interpolation techniques are available, including trilinear interpolation and nearest-neighbor interpolation.

In this section, we will concentrate on a specific interpolation algorithm known as inverse distance weighting. This interpolation technique has found widespread application in 3D point cloud processing [45] and is also utilized in the Transformer-based model used in our project.

Consider a set consisting of N 3D points $\{\mathbf{x}_i, \mathbf{f}_i | \text{ for } \mathbf{x}_i \in \mathbb{R}^3, \mathbf{f}_i \in \mathbb{R}^C\}_{i=1}^N$, where $\mathbf{x}_i$ is the xyz-coordinates and $\mathbf{f}_i$ is the feature vector for the ith point. The interpolation function based on inverse distance weighting can be defined as follows:

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &= \frac{\sum_{i \in k\mathrm{NN}(\mathbf{x})} w_i(\mathbf{x})\, \mathbf{f}_i}{\sum_{i \in k\mathrm{NN}(\mathbf{x})} w_i(\mathbf{x})}, \\ \text{where } w_i(\mathbf{x}) &= \frac{1}{d(\mathbf{x},\ \mathbf{x}_i)^p}. \end{aligned} \tag{2.50}$$

In Eq. (2.50), $\mathbf{x}$ represents an interpolated (unknown) point from the set of points where we aim to propagate features, while $\mathbf{x}_i$ represents an interpolating (known) point within the provided set. The term $k\mathrm{NN}(\mathbf{x})$ denotes the k nearest neighbors of the unknown point $\mathbf{x}$ in the provided set. Additionally, $d(\mathbf{x},\ \mathbf{x}_i)$ represents the distance between $\mathbf{x}$ and $\mathbf{x}_i$, and p is a positive real number referred to as the power parameter. Evidently, with the distance $d(\mathbf{x},\ \mathbf{x}_i)$ increasing, the weight $w_i(\mathbf{x})$

decreases. In our project, the term $d(\mathbf{x},\ \mathbf{x}_i)$ is defined as the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}_i$, the value of p is set to 1, and k is set to 3.

3
Methods

In this chapter, we will introduce our method: Point Transformer Diffusion (PTD). We will first describe the formulation of PTD as a diffusion model in Section 3.1. Next, the reverse process of PTD is parameterized as a single Point Transformer [72], which we will discuss in Section 3.2. After presenting the two key components, we will introduce three techniques that improve PTD's performance in Section 3.3. Finally, we will provide a description of the relevant datasets and evaluation metrics in Sections 3.4 and 3.5, respectively.

3.1 Diffusion model for 3D point clouds

For the generative modeling of 3D shapes, PTD leverages DDPM [17] as the diffusion model and adapts it for 3D point clouds. Figure 3.1 depicts its forward and reverse processes, which closely resemble those of the original DDPM. The only difference is the data type, which is a 3D point cloud instead of an image.

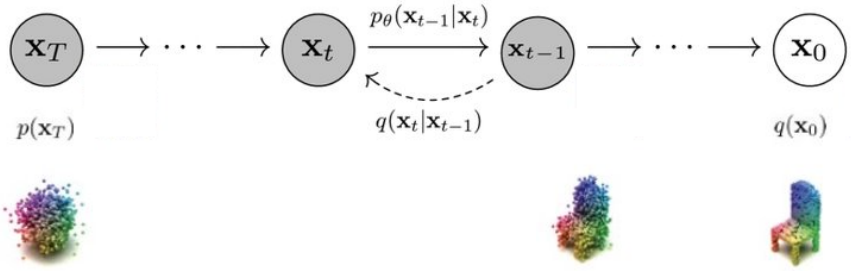


Figure 3.1: Visualization of the forward and reverse processes of a diffusion model for 3D point clouds. The forward process starts from the ground truth, $\mathbf{x}_0 \in \mathbb{R}^{N\times 3}$, and gradually adds noise using $q(\mathbf{x}_t|\mathbf{x}_{t-1})$. It ends with $\mathbf{x}_T$, which follows an isotropic Gaussian distribution. Conversely, the reverse process gradually denoises Gaussian noise, generating outputs adhering to the distribution $q(\mathbf{x}_0)$. Each step of the reverse process is based on the prediction of a neural network model, $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$.

3.1.1 Formulation

In this section, we will formulate the diffusion model for 3D point clouds. In the context of 3D point clouds, we can assume each point cloud input as a set of N points with xyz-coordinates, denoted by $\mathbf{x} \in \mathbb{R}^{N\times 3}$.

Forward and Reverse processes. Referring to Figure 3.1, the two transitions are defined as follows, given a predefined increasing variance schedule $\beta_1, ..., \beta_T$:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}), \tag{3.1}$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \beta_t\mathbf{I}). \tag{3.2}$$

As the forward process is defined as a discrete-time Markov chain, we express the conditional probability distribution as:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}). \tag{3.3}$$

Similarly, with the reverse process starting at $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$, the characteristic joint distribution is calculated using:

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t). \tag{3.4}$$

Section 2.2.1 and 2.2.2 provide detailed explanations and derivations related to the forward and reverse processes.

Training. Since maximizing the marginal likelihood $p_\theta(\mathbf{x_0})$ directly is intractable, we use the ELBO as a proxy objective. As extensively discussed in sections from 2.2.3 to 2.2.5, the final loss can be minimized by applying an $\mathcal{L}_2$ loss between the noise $\boldsymbol{\epsilon}$ and the model output $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$:

$$\mathcal{L}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\Big[\Big|\Big|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big|\Big|^2\Big],\ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{3.5}$$

Intuitively, the model predicts the noise vector required to restore the 3D shape's integrity. The complete training procedure is precisely described in Algorithm 1.

Sampling. The sampling process begins by sampling $\mathbf{x}_T$ from a normal distribution. Then, it progressively samples from $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ from T to 1 using:

$$\begin{aligned} &\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big) + \sqrt{\beta_t}\mathbf{z}, \\ &\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \text{ if } t > 1, \text{ else } \mathbf{z} = \mathbf{0}. \end{aligned} \tag{3.6}$$

Here, $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$. The complete sampling procedure is exactly presented in Algorithm 2.

3.1.2 Implementation

In the project, we utilized the implementation of Point-Voxel Diffusion (PVD) [74]. The implementation of PVD is based on that of DDPM [17] but adapts it for 3D point clouds. The neural network model employed by PVD is a single point-voxel CNN [13], which has proven to be successful in various tasks related to 3D point cloud learning. We replaced it with a Transformer-based model, which will be introduced in detail in the next section.

3.2 Backbone: Point Transformer

Since PTD is a diffusion model specifically designed for 3D point clouds, it is crucial to utilize a deep learning model adept at dealing with such data. Zhao et al. introduced Point Transformer [72], which utilizes self-attention networks for processing 3D point clouds. This approach has demonstrated its effectiveness across multiple tasks, such as semantic scene segmentation, object part segmentation, and object classification.

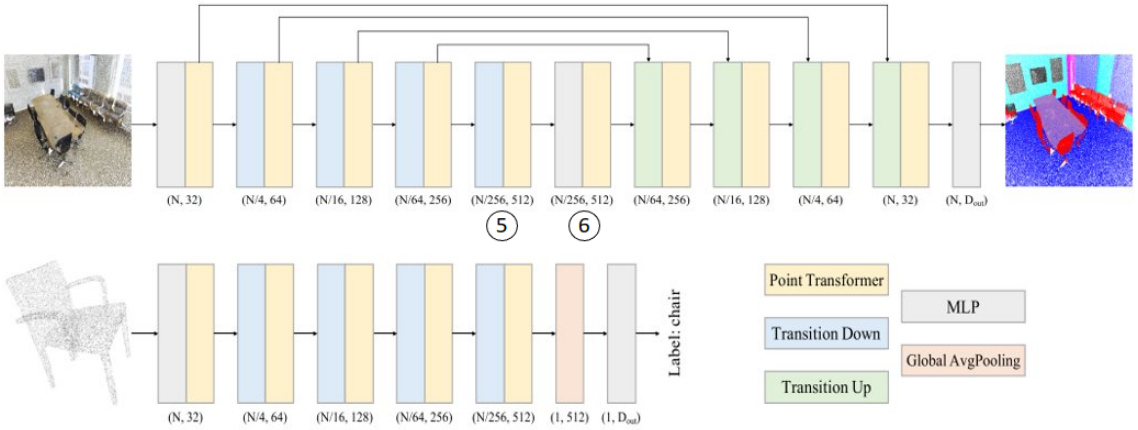


Figure 3.2: Point Transformer networks for semantic segmentation (top) and classification (bottom). There is a flaw in the visualization of the network for semantic segmentation. In the figure, it shows that the output of the 5th stage is directly passed to the 6th stage. However, in reality, the output of the 5th stage is first aggregated by an average pooling block to generate a global feature. Subsequently, the output of the 5th stage is concatenated with the global feature before being passed to the 6th stage. (Image source: [72])

Figure 3.2 visualizes two network architectures for classification and semantic segmentation. We will focus on the semantic segmentation network for two reasons. Firstly, it serves as the neural network model for the diffusion model in our project. Secondly, the semantic segmentation network is an extension of the classification network, so we have already mastered the classification network once we understand the semantic segmentation network.

Overall, the Point Transformer's semantic segmentation network utilizes a U-Net [52] architecture, which incorporates both a contracting path and an expansive path. The contracting path acts as the feature encoder, while the expansive path functions as the feature decoder.

The feature encoder consists of five stages that handle downsampled point sets in a progressive manner. Each stage is associated with a specific downsampling rate, namely $[1, 4, 4, 4, 4]$, resulting in corresponding point set sizes of $[N, N/4, N/16, N/64, N/256]$, where N represents the number of input points. The first stage is composed of an MLP block and a point transformer block. The MLP block projects the xyz-

coordinates into 32-dimensional embeddings for all N points. The other four stages include a transition down block followed by a point transformer block. Further details about the point transformer block and the transition down block will be presented in Section 3.2.1 and 3.2.2, respectively.

Similarly, the feature decoder comprises five stages that process progressively upsampled point sets. The first stage consists of an MLP block and a point transformer block, while the other four stages consist of a transition up block followed by a point transformer block. For these four stages, each of them is coupled with a corresponding stage in the encoder using a skip connection. More details regarding the transition up block will be discussed in Section 3.2.3.

After the final decoder stage generates a feature vector for each point, the output head applies an MLP to map these features to the final logits. To illustrate its usage, consider two examples. In the task of 3D object part segmentation, the output head predicts scores for all parts, and the final prediction corresponds to the part with the highest score. In our project, when Point Transformer is used as the neural network model within the diffusion model, the output head predicts the noise added during the forward process.

3.2.1 Point transformer block

In Point Transformer, every point transformer block is structured as a residual net, as depicted in Figure 3.3. It consists of a point transformer layer, two linear projection layers, and a residual connection [15].

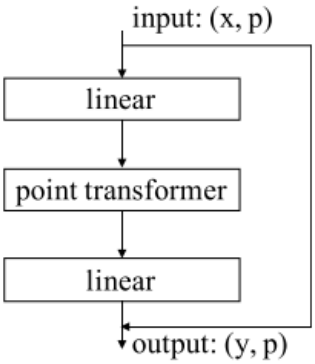


Figure 3.3: Point transformer block. The input consists of a set of feature vectors $\mathbf{x}$ with their corresponding 3D coordinates $\mathbf{p}$. The point transformer block enables the exchange of information among these localized feature vectors, resulting in new feature vectors $\mathbf{y}$ with the same coordinates $\mathbf{p}$ as the output. (Image source: [72])

Point transformer layer. The core component of a point transformer block is the point transformer layer. This layer utilizes the vector self-attention mechanism, which can be represented as follows:

$$\mathbf{y}_i = \sum_{\mathbf{x}_j \in \mathcal{X}(i)} \rho\Big(\gamma(\varphi(\mathbf{x}_i) - \psi(\mathbf{x}_j) + \delta)\Big) \odot \Big(\alpha(\mathbf{x}_j) + \delta\Big). \tag{3.7}$$

Here, the linear functions φ, ψ, and α perform pointwise feature transformations to create distinct weight matrices. The position encoding function δ and the mapping function γ use two separate MLPs. The softmax function ρ is applied to perform normalization. Regarding the data, the subset $\mathcal{X}(i) \subseteq \mathcal{X}$ consists of the k nearest neighbors of $\mathbf{x}_i$. Additionally, Eq. (3.7) can be clearly illustrated in Figure 3.4.

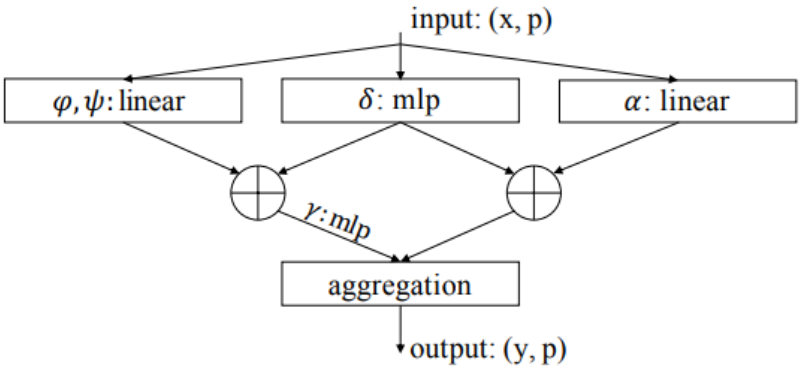


Figure 3.4: Point transformer layer. (Image source: [72])

Position encoding. In attention-based methods, position encoding plays a crucial role in enabling the operator to adapt to the local structure in the data [56]. In the point transformer layer, δ serves as a trainable, parameterized position encoding function that leverages the relative positions between 3D points, which is defined as follows:

$$\delta = \theta(\mathbf{p}_i - \mathbf{p}_j). \tag{3.8}$$

Here, we use the coordinates for the 3D points i and j, denoted as $\mathbf{p}_i$ and $\mathbf{p}_j$, respectively. The encoding function θ implemented as an MLP containing two linear layers and one ReLU nonlinearity. Importantly, the position encoding θ undergoes end-to-end training alongside the other subnetworks.

3.2.2 Transition down block

The transition down block is a critical component that plays a central role in downsampling the spatial dimensions of the input data to capture essential context and higher-level features. As depicted in Figure 3.2, each transition down block retains a quarter of the input points while simultaneously doubling the number of feature channels.

Figure 3.5 presents a schematic representation of the transition down block, illustrating its three-step data processing. The input comprises feature vectors denoted as $\mathbf{x}$ and xyz coordinates represented by $\mathbf{p}_1$, corresponding to the input points. The output also consists of feature vectors, designated as $\mathbf{y}$, and xyz coordinates, designated as $\mathbf{p}_2$, respectively. In the first step, farthest point sampling (FPS) is applied to $\mathbf{p}_1$ to select a well-distributed subset, denoted as $\mathbf{p}_2$, with the required cardinality. Next, the second step first searches k nearest neighbors for each point in $\mathbf{p}_2$ through kNN computation on $\mathbf{p}_1$. Subsequently, for each point in $\mathbf{p}_2$, grouped with

their corresponding k neighbors in $\mathbf{p}_1$, their features in $\mathbf{x}$ undergo a linear transformation, followed by batch normalization and ReLU activation. Finally, in the third step, max pooling is performed on each point in $\mathbf{p}_2$ based on its k neighbors in $\mathbf{p}_1$, yielding the output features $\mathbf{y}$.

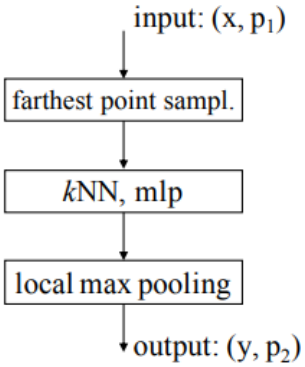


Figure 3.5: Transition down block. The main purpose of the transition down block is to reduce the cardinality of the point set as required. As a result, $\mathbf{p}_2$ is formed as a subset of $\mathbf{p}_1$. (Image source: [72])

3.2.3 Transition up block

In the expansive path, each transition up block is responsible for increasing the spatial dimensions of the feature maps. This crucial process helps to restore the resolution lost during the contracting path, enabling the network to recover finer details.

To enhance information flow during upsampling, skip connections are utilized to connect corresponding feature maps from the contracting path, as illustrated in Figure 3.2. These skip connections play a significant role by enabling the decoder to access high-resolution feature maps from the encoder, thereby aiding in the accurate reconstruction of fine details.

Figure 3.6 presents a schematic representation of the transition up block, illustrating its three-step data processing. Unlike a transition down block, a transition up block takes two inputs. The first input, denoted as input_1, is from the preceding decoder stage, consisting of feature vectors represented by $\mathbf{x}_1$ and xyz coordinates represented by $\mathbf{p}_1$. The second input, input_2, comes from the corresponding stage in the encoder via a skip connection. The output also consists of feature vectors, designated as $\mathbf{y}$, and xyz coordinates, designated as $\mathbf{p}_2$. In the first step, the feature vectors $\mathbf{x}_1$ undergo a linear transformation, followed by batch normalization and ReLU activation. Subsequently, these transformed features from $\mathbf{x}_1$ are mapped onto the higher-resolution points $\mathbf{p}_2$ using an interpolation technique: inverse distance weighting (IDW). The interpolated features are then summarized with the linearly transformed features from $\mathbf{x}_2$, resulting in the output features $\mathbf{y}$.

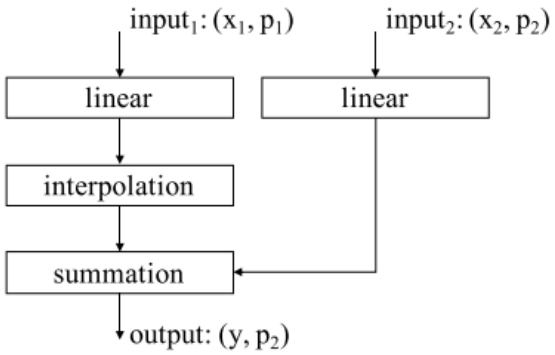


Figure 3.6: Transition up block. The main purpose of the transition up block is to map features from the downsampled points in $\mathbf{p}_2$ onto all the points in $\mathbf{p}_1$ that is a superset of $\mathbf{p}_2$. (Image source: [72])

3.2.4 Implementation

In the project, we learned from and compared several repositories on GitHub, and then implemented our own version of Point Transformer from scratch.

[2] was the first codebase we attempted to use, but it has three significant drawbacks. Firstly, it implements several functions, such as kNN, FPS, and IDW, purely in Python without utilizing CUDA. Consequently, the program occupies a large amount of GPU memory, requiring manual cache-clearing in the code, which proves to be quite time-consuming. Despite having explicit cache-clearing operations, the inefficient utilization of GPU memory leads to a limitation in the batch size for the data. Secondly, it incorporates code borrowed from [65] to build the transition down blocks and the transition up blocks in Point Transformer. However, the implementation in many places does not conform to the description in the paper of Point Transformer [72]. Lastly, the reproduced results using this codebase fall moderately below the results reported in [72]. On the positive side, this codebase is easy to read and manipulate, which facilitated my initial experimentation.

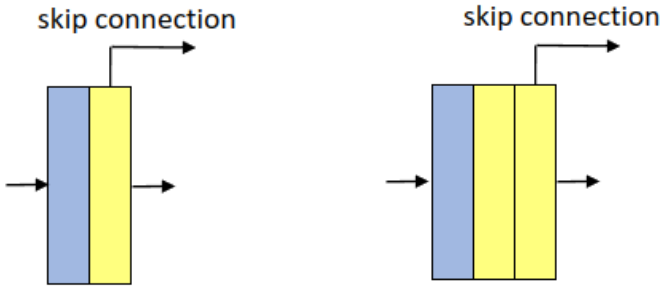
Then, we discovered another codebase named Pointcept [7], which was recently publicized in 2023. It is a comprehensive repository with implementations of many neural network models, developed and maintained by researchers in the lab of the Point Transformer's author, making it akin to the official implementation. Nevertheless, we did not directly use the implementation of Point Transformer due to its data processing mode, which is not 100% compatible with my project. In this implementation, the data needs to be formed as (N, X), where N represents the total number of points in the batch, and X represents the features and coordinates of these points. Additionally, a variable list denoted as O is used to record the offset information, acting as the separator of point clouds in batch data. However, the data used in the experiments of our project is at the object level, meaning that every point cloud has the same number of points. Consequently, it is more straightforward to process data formed in the batch mode as (B, N, X), where B represents the batch size, and N represents the number of points for each object.

Finally, we decided to create a customized version of Point Transformer that aligns perfectly with the specific requirements of our project. For the foundational blocks in Point Transformer, we referred to the implementation in Pointcept and adapted it to handle data in batch mode. To carry out operations such as kNN, FPS, and IDW on point clouds, we leveraged the CUDA implementation from PointWeb [73]. As for the training and evaluation scripts, we drew from two GitHub projects, namely [2] and [65]. In each part, there exists necessary modifications for seamless code integration and bug fixes.

3.3 Three techniques for improvement

3.3.1 Model structure and hyperparameters

In Figure 3.2, all stages have one point transformer block, forming the basic version of Point Transformer. However, depending on the complexity of the tasks or datasets, additional point transformer blocks may be required. In our implementation of Point Transformer, it is easy to increase the number of point transformer blocks at any stage without affecting the other stages. Figure 3.7 symbolically illustrates the process of adding an extra point transformer block to an encoder stage.



(a) basic structure **(b)** a variant

Figure 3.7: An example of the modification to the model structure. Consistent with Figure 3.2, the blue rectangle represents a transition down block, and the yellow one represents a point transformer block. In comparison to the structure in (a), the structure in (b) includes an additional point transformer block.

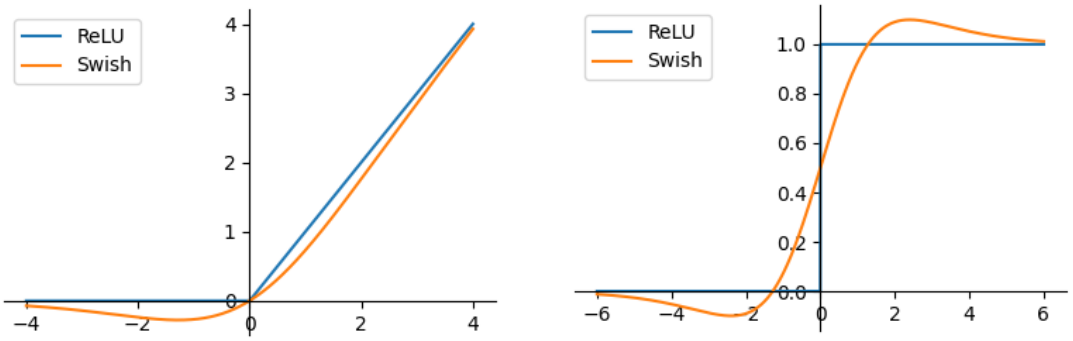
Additionally, there are two critical model hyperparameters significantly influencing performance. The first one is the downsampling rates. In the basic version of Point Transformer, each transition down block retains a quarter of the input data points. However, in certain cases, preserving more information at particular stages becomes essential. To address this, we can decrease the downsampling rate of the transition down block in that stage. By adjusting this hyperparameter, we gain better control over the amount of information retained at each stage.

Another critical model hyperparameter is the number of nearest neighbors, denoted as k, used in both the point transformer and transition down blocks. This parameter

determines the local neighborhood around each point. If the value of k is too small, the model may lack sufficient context for accurate predictions. Conversely, if it is too large, many neighbors may be farther and less relevant, leading to the introduction of excessive noise rather than useful information. Therefore, carefully selecting the appropriate value of k based on the specific tasks and datasets is essential for optimal performance.

3.3.2 Non-linear activation function

In Point Transformer, the activation function used is ReLU, which is one of the most commonly-used activation functions. However, the choice of activation functions can have a substantial impact on training dynamics and task performance in deep networks. Hence, exploring and testing different activation functions could potentially yield rewarding results. In our project, an alternative activation function named Swish [46] was employed, defined as $f(x) = x \cdot \text{sigmoid}(x)$, to replace ReLU in certain parts of the Point Transformer-based model.



(a) ReLU and Swish functions **(b)** ReLU and Swish derivatives

Figure 3.8: ReLU and Swish functions and derivatives.

Swish offers two advantages over ReLU. Firstly, as depicted in Figure 3.8 (a), Swish is a smooth activation function, ensuring continuity and differentiability across its entire domain. In contrast, ReLU has a non-differentiable point at the origin, which can create challenges in certain optimization algorithms, particularly those relying on gradient-based methods. Secondly, as illustrated in Figure 3.8 (b), the ReLU activation function exhibits a "dying ReLU" problem, wherein the gradient becomes zero for negative inputs. This leads to inactive neurons that no longer contribute to the network's training. On the contrary, Swish provides non-zero gradients for all inputs, mitigating the "dying ReLU" issue and potentially enhancing learning in deep networks. Apart from these theoretical advantages, Swish has demonstrated better generalization performance compared to ReLU in certain scenarios, particularly in deeper neural network architectures [46].

Although Swish offers several advantages over ReLU, it also presents some potential disadvantages. The primary concern is its computational cost. Given that the

Transformer-based model is already computationally expensive, and the diffusion model is inefficient in sampling, introducing Swish may further burden the computational resources. Additionally, while Swish has demonstrated benefits over ReLU in certain scenarios, its advantages may not consistently apply to all network architectures and tasks. Its performance could vary depending on the specific model, dataset, and task at hand. Therefore, careful consideration should be given to the trade-offs before incorporating Swish into the model.

Based on the preceding discussion, we decided to conduct experiments by replacing ReLU with Swish either in the first MLP block, the last MLP block, or in both of them in the Point Transformer network depicted in Figure 3.2. The first MLP block is responsible for projecting the xyz-coordinates into 32-dimensional embeddings, while the last MLP block maps the learned features to the final logits. Since these two blocks are unique in the network, using Swish in these locations could be beneficial in avoiding the "dying ReLU" problem and ensuring that all the information is effectively utilized at the first and last stages. Furthermore, incorporating Swish in these specific blocks, without altering the other activation functions, will not significantly increase the computational burden. This targeted replacement of ReLU with Swish allows for a controlled assessment of its impact on the model's performance.

3.3.3 Learning rate warm-up

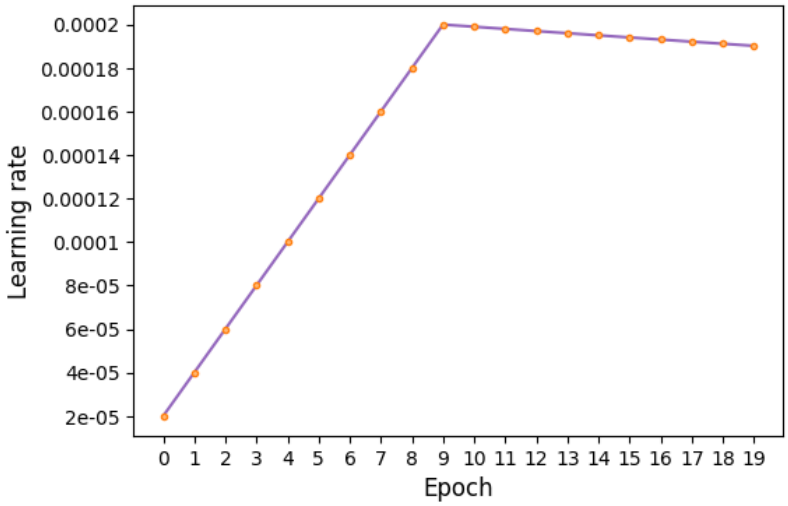


Figure 3.9: An example of linear warm-up. The learning rate starts at 0.00002 and then gradually increases linearly to 0.0002 during the first ten epochs, which represent the period of linear warm-up. Following this warm-up phase, the learning rate continues to vary according to the original schedule.

The mechanism of "learning rate warm-up" involves gradually increasing the learning rate during the initial stages of training, starting from a smaller value and progressively reaching the intended value instead of using the planned learning rate directly. This gradual increase serves several purposes: firstly, it enables the model to be-

gin training with smaller learning rates, facilitating more effective exploration of the parameter space in the early phases; secondly, it reduces the risk of employing large and potentially detrimental weight updates in the initial stages, which could destabilize the training process. Overall, applying learning rate warm-up provides a smoother and more stable optimization process, potentially leading to improved convergence and better generalization of deep learning models.

The implementation of this mechanism varies, and in the project, we utilized a linear warm-up learning rate scheduler. For example, as depicted in Figure 3.9, the learning rate is gradually increased in a linear fashion from a smaller initial value to the desired maximum value during the early stages of model training.

3.4 Datasets

In the project, we utilized three distinct datasets, and the following three subsections provide a detailed introduction to each dataset.

3.4.1 ModelNet40

The ModelNet40 dataset [61] encompasses synthetic object point clouds and stands as a significant benchmark for point cloud analysis. Its popularity stems from the dataset's wide-ranging categories, well-defined shapes, and meticulous construction. The dataset itself consists of 12,311 CAD models, categorized into 40 distinct classes. Out of these, 9,834 meshes are designated for training, while the remaining 2,468 are set aside for testing. Figure 3.10 displays examples of major object categories and provides dataset statistics.

Figure 3.10: ModelNet40 Dataset. On the left, a word cloud visualization depicts the ModelNet dataset, where the size of each word indicates the number of 3D models in each category. Larger font sizes represent categories with more instances. On the right, examples of 3D chair models are displayed. (Image source: [61])

In the project, we follow the data preparation procedure outlined by Qi et al. [45]. Specifically, the point cloud data points are uniformly sampled from the mesh surfaces. Subsequently, the data points are further preprocessed by relocating them to the origin and scaling them to fit within a unit sphere.

3.4.2 ShapeNet

ShapeNet [6] is an extensive collection of 3D CAD models representing a wide range of objects. It serves as a richly-annotated and large-scale repository of shapes. The repository includes over 300 million models, with 220,000 of them classified into 3,135 classes, organized based on the WordNet taxonomy.

In the project, we utilize the dataset provided by PointFlow [67], which comprises 15,000 points per shape uniformly sampled from the corresponding mesh surface in ShapeNet [6]. For efficient training and testing, the data preprocessing module samples 2,048 points for each shape. Subsequently, the data is processed according to the established procedures in PointFlow.

Figure 3.11: Examples of point clouds sampled by PointFlow from the ShapeNet dataset. Each of them contains 2,048 points. From left to right: airplane, chair, and car. (Image source: [67])

3.4.3 ShapeNet-Part

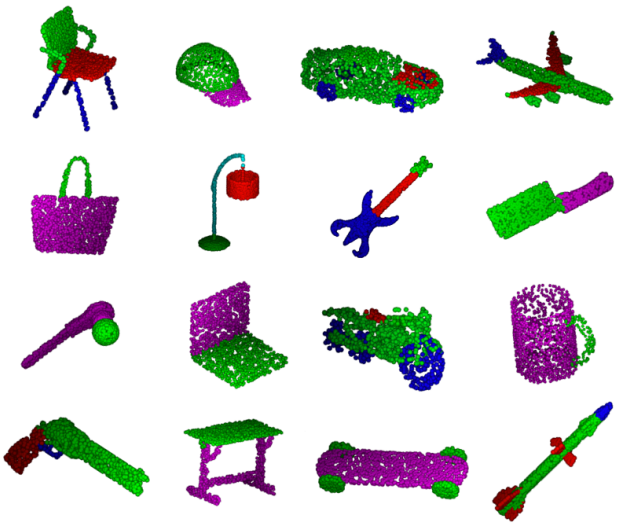


Figure 3.12: Point cloud examples of 16 categories in the ShapeNet-Part dataset. These categories are chair, cap, car, airplane, bag, lamp, guitar, knife, earphone, laptop, motorbike, mug, pistol, table, skateboard, and rocket. (Image source: [66])

The ShapeNet-Part dataset [68] is specifically annotated for the task of 3D object part segmentation. It comprises 16,880 models, which are classified into 16 categories. The dataset is split into 14,006 models for training and 2,874 models for

testing. Each category consists of a varying number of parts, ranging from 2 to 6, resulting in a total of 50 different parts across the dataset. In the project, we utilize the sampled point sets generated by Qi et al. [45].

3.5 Evaluation metrics

In the project, we began by testing our implementation of Point Transformer in both classification and segmentation tasks, using appropriate accuracy and segmentation metrics. Subsequently, we evaluated PVD and PTD in a 3D shape generation task, requiring the use of suitable distance metrics. In the following subsections, we will delve into the evaluation metrics used in the project.

3.5.1 Accuracy

The metric of accuracy plays a significant role in evaluating classification models, as it provides an indication of their ability to make correct predictions by comparing them to the actual labels, thus offering a comprehensive overview of their performance. In this section, we will explore two specific accuracy metrics: mean accuracy within each category (mAcc) and overall accuracy (OA) across all classes.

OA is a widely used evaluation metric in classification tasks, measuring the overall correctness of the classifier across all categories. It is calculated by dividing the number of correct decisions by the total number of cases, as follows:

$$\text{OA} = \frac{[No.\ \text{correct decisions}]}{[No.\ \text{cases}]}. \tag{3.9}$$

On the other hand, mAcc is another commonly used evaluation metric in classification tasks. It calculates the average accuracy for each category individually and then computes the mean of these accuracies. By making a few modifications to Eq. (3.9), we can derive the formula for calculating mAcc:

$$\text{mAcc} = \frac{1}{C}\sum_{i=1}^{C}\frac{[No.\ \text{correct decisions}]_i}{[No.\ \text{cases}]_i}, \tag{3.10}$$

where C represents the total number of categories.

3.5.2 Intersection over Union

Intersection over Union (IoU), also referred to as the Jaccard index, is a well-established measure for quantifying the similarity between two finite sample sets. Generally, for two finite sample sets A and B, their IoU is defined as well as computed as the ratio of the intersection ($A \cap B$) to the union ($A \cup B$) of sets A and B, as shown in Eq. (3.11).

$$\begin{aligned} \mathrm{IoU}(A, B) &= \frac{|A \cap B|}{|A \cup B|} \\ &= \frac{|A \cap B|}{|A| + |B| - |A \cap B|}. \end{aligned} \tag{3.11}$$

The use of IoU in computer vision tasks was initially introduced by [12] within the context of object detection on 2D images. Since then, IoU has become a common evaluation metric in computer vision research. Its application has also extended to other domains, including 3D point cloud analysis. Specifically, in the task of 3D object part segmentation, the metrics of overall average category IoU (cat. mIoU) and overall average instance IoU (ins. mIoU) are widely employed.

To understand cat. mIoU and ins. mIoU, we first need to delve into the computation of mean Intersection over Union (mIoU). Regarding 3D object part segmentation, PointNet [44] approaches it as a per-point classification problem. For each object O belonging to the category C, the mIoU of the object is calculated as follows:

1. Compute the IoU between the ground truth and prediction for each part type within category C. In case the union of the ground truth and prediction points is empty, consider the part IoU as 1.
2. Average the IoUs across all part types within category C to determine the mIoU for object O.

After obtaining the mIoU for each object, we can then compute cat. mIoU and ins. mIoU, respectively. ins. mIoU represents the average mIoU across all objects, which can be expressed as follows:

$$\text{ins.mIoU} = \frac{1}{N} \sum_{i=1}^{N} [\text{mIoU}]_i, \tag{3.12}$$

where N represents the total number of objects in the evaluation dataset.

On the other hand, cat. mIoU represents the average mIoU across all categories. It is computed by first calculating the average mIoU for each category and then averaging these category-wise mIoUs, expressed as follows:

$$\text{cat.mIoU} = \frac{1}{C} \sum_{i=1}^{C} \Big(\frac{1}{N_i} \sum_{j=1}^{N_i} [\text{mIoU}]_{ij} \Big), \tag{3.13}$$

where C represents the total number of categories, and N_i represents the total number of objects in the ith category.

3.5.3 Distance

Chamfer Distance (CD) [33] and Earth Mover's Distance (EMD) [53] are two common metrics in computer vision. They are frequently employed in point cloud analysis to measure the similarity between two point clouds.

CD quantifies the dissimilarity between two point sets by measuring the distances between the points in one set and their nearest neighbors in the other set. Consider two point clouds, X and Y, with an equal number of points. CD is defined as follows:

$$\mathrm{CD}(X, Y) = \sum_{x \in X} \min_{y \in Y} ||x - y||_2^2 + \sum_{y \in Y} \min_{x \in X} ||x - y||_2^2. \tag{3.14}$$

In contrast, EMD treats the point clouds as probability distributions and determines the minimum cost required to transform one point cloud distribution into another. It takes into account the spatial distance between points as the cost of moving mass from one point to another. The formulation is as follows:

$$\mathrm{EMD}(X, Y) = \min_{\phi: X \to Y} \sum_{x \in X} ||x - \phi(x)||_2, \tag{3.15}$$

where ϕ represents a bijection between X and Y.

Various methods utilize CD or EMD to measure the quality in 3D point cloud generation. In our project, the 1-nearest neighbor accuracy (1-NNA) method proposed by Lopez-Paz and Oquab [32] is employed. Let S_g represent the set of generated point clouds, S_r denote the set of reference point clouds where $|S_r| = |S_g|$, and S_{-X} be the union of S_r and S_g without the element X: $S_{-X} = S_r \cup S_g - \{X\}$. Additionally, let N_X be the nearest neighbor of X in S_{-X}. The 1-NNA measures the leave-one-out accuracy of the 1-NN classifier:

$$\text{1-NNA}(S_g, S_r) = \frac{\sum_{X \in S_g} \mathbb{I}[N_X \in S_g] + \sum_{Y \in S_r} \mathbb{I}[N_Y \in S_r]}{|S_g| + |S_r|}, \tag{3.16}$$

where $\mathbb{I}[\cdot]$ denotes the indicator function. For each sample, the 1-NN classifier assigns it to either S_r or S_g based on the label of its nearest sample, which can be determined using either CD or EMD. In the scenario where S_g and S_r originate from the same distribution, as the number of samples increases, the classifier's accuracy is expected to converge to 50%. The closer the accuracy is to 50%, the more similar S_g and S_r are, indicating better learning of the target distribution by the model. In practice, the accuracy ranges between 50% and 100%.

4
Results

In this chapter, we will present the primary experimental results to demonstrate the effectiveness of our method: Point Transformer Diffusion (PTD). The chapter is divided into three sections. The first two sections respectively present the experimental results of Point-Voxel Diffusion and Point Transformer, both of which serve as cornerstones for PTD. The third section, Section 4.3, includes the generated results of PTD's models, extensive ablation studies covering various settings, as well as an in-depth analysis and discussion.

4.1 Point-Voxel Diffusion

This section presents the reproduced results of PVD on the ShapeNet dataset. Replicating PVD's results is essential before implementing our own method due to two key factors. Firstly, PVD represents one of the state-of-the-art results in 3D point cloud generation, serving as the baseline for our project. Secondly, our project utilizes PVD's implementation of DDPM tailored for 3D point clouds, which emphasizes the importance of validating the effectiveness of this implementation.

For the task of unconditional shape generation, PVD was evaluated on three categories of ShapeNet: airplane, chair, and car. The corresponding results in terms of CD and EMD are shown in Table 4.1. We reproduced the results using two methods. The first method involved directly utilizing the pre-trained models provided by PVD, which will be described in Section 4.1.1. The second approach was to train the models from scratch and then use them for generation and evaluation, as detailed in Section 4.1.2. Additionally, we visualized the generated point clouds in both methods to assess whether PVD works properly.

Table 4.1: Generation results presented in PVD's paper [74] using 1-NN as the metric. Both CD and EMD as the distance measure are calculated. Lower scores indicate better quality and diversity.

Category	CD	EMD
airplane	73.82	64.81
chair	56.26	53.32
car	54.55	53.83

4.1.1 Pre-trained models

Three pre-trained models are available for download from PVD's GitHub page. These models are capable of generating airplane, chair, and car shapes, respectively. The evaluation results for each model are recorded in Table 4.2, and several generated shapes are visualized in Figure 4.1.

Table 4.2: Evaluation results of PVD's pre-trained models.

Category	Model	CD	EMD
airplane	airplane_2899.pth	73.08	66.41
chair	chair_1799.pth	58.08	53.85
car	car_3999.pth	58.94	53.69

* airplane_2899.pth represents the pre-trained model for the airplane category at epoch 2899. The interpretations of the other two models follow the same rule.

Compared to the results in Table 4.1, the evaluation of the pre-trained models demonstrates better performance in two metrics: CD for the airplane category and EMD for the car category. Meanwhile, it shows worse performance in the other four metrics. Since the evaluation results of the trained models exhibit a similar phenomenon, the discussion with respect to those differences will be presented in Section 4.1.3.

(a) airplanes **(b)** chairs **(c)** cars

Figure 4.1: Shape generations with 2,048 points using PVD's pre-trained models. In (a), the two airplanes seem to be a passenger plane and an aerospace aircraft, respectively. In (b), the right chair has armrests while the left one does not. In (c), the two cars seem to be an SUV and a sedan, respectively.

4.1.2 Trained models

In addition to simply using the pre-trained models, we also trained the models from scratch and subsequently evaluated them. This approach is of greater significance as it provides a direct assessment of the effectiveness of PVD's implementation. The evaluation results for each model are recorded in Table 4.3, and several generated shapes are visualized in Figure 4.1.

Table 4.3: Evaluation results of the trained models of PVD.

Category	Model	CD	EMD
airplane	airplane_2399.pth	73.20	62.59
chair	chair_1899.pth	57.40	55.06
car	car_3899.pth	56.39	52.41

In comparison to the results in Table 4.1, the evaluation of the trained models shows improved performance in three metrics: CD and EMD for the airplane category, as well as EMD for the car category. Meanwhile, it exhibits poorer performance in the other three metrics.

(a) airplanes **(b)** chairs **(c)** cars

Figure 4.2: Shape generations with 2,048 points using the trained models of PVD. In (a), the two airplanes appear to be a passenger plane and a military aircraft, respectively. In (b), the left chair resembles a crossbar chair, while the right one resembles a lounge chair. In (c), the two cars seem to be a van and a race car, respectively.

4.1.3 Analysis and discussion

As depicted in Figure 4.1 and 4.2, both the pre-trained and trained models are capable of generating realistic and high-quality shapes of airplanes, chairs, and cars. Furthermore, the evaluation results of these models align closely with the corresponding results presented in PVD's paper. Therefore, the experiments have effectively verified the effectiveness of PVD and its implementation.

As mentioned in Section 4.1.1, there are differences among the evaluation results presented in PVD's paper [74], those of PVD's pre-trained models, and those of PVD's trained models. These differences can be attributed to three factors. Firstly, slight discrepancies may arise due to variations in the versions of GPU, CUDA, PyTorch, and other software packages used. Secondly, the training dataset contains 10,000 points for each object, while the test dataset contains 5,000 points. During each iteration, the program randomly selects 2,048 points for each object in the batch, introducing randomness to both training and evaluation processes. Lastly, the frequency of evaluation plays a significant role. While PVD's paper does not specify the evaluation frequency, the pre-trained models were obtained at epochs that are multiples of 100. Similarly, when we evaluated the models, the evaluations

were also conducted at intervals of 100 epochs, which translates to an evaluation frequency of $\frac{1}{100}$. Due to the computational expense, evaluating all models at every epoch is impractical, making it challenging to achieve identical results at such a low evaluation frequency. Despite these differences, the corresponding results in Table 4.1, 4.2, and 4.3 consistently demonstrate good performance, effectively validating the effectiveness of PVD and its implementation.

After successfully reproducing the results of PVD, we can confidently utilize its implementation of DDPM for 3D point clouds in the project.

4.2 Point Transformer

In this section, we will present the reproduced results of Point Transformer. In our method, we replace the Point-Voxel CNN within the diffusion model with the Point Transformer model. Therefore, it is of utmost importance to validate both the effectiveness of the Point Transformer model and the performance of our implementation before integrating it into the diffusion model. To accomplish this, we conducted experiments on two specific tasks: classification using the ModelNet40 dataset and 3D object part segmentation using the ShapeNet-Part dataset.

4.2.1 Classification task

For the classification task on the ModelNet40 dataset, Table 4.4 presents the results of our Point Transformer implementation and compares them with the corresponding results in Point Transformer's paper [72]. In our implementation, the trained model achieved a slightly higher mAcc score, but a marginally lower OA score compared to the results reported in Point Transformer's paper.

Table 4.4: Shape classification results on the ModelNet40 dataset.

Method	mAcc	OA
Point Transformer's paper	90.6	93.7
our implementation	90.7	93.1

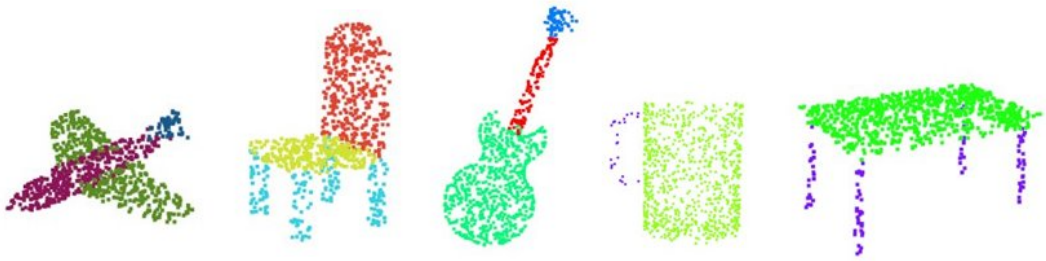
4.2.2 3D object part segmentation task

For the 3D object part segmentation task on the ShapeNet-Part dataset, Table 4.5 presents the results of our Point Transformer implementation and compares them with the corresponding results in Point Transformer's paper. Additionally, Figure 4.3 visualizes several object part segmentation results, providing a straightforward way to assess the effectiveness of our implementation.

Table 4.5: Object part segmentation results on the ShapeNet-Part dataset.

Method	cat. mIoU	ins. mIoU
Point Transformer's paper	83.7	86.6
our implementation	83.0	86.2

(a) ground truth



(b) prediction

Figure 4.3: Visualization of object part segmentation results on the ShapeNet-Part dataset. Among the 16 categories in the ShapeNet-Part dataset, five of them are visualized, which are airplane, chair, guitar, mug, and table, from left to right.

4.2.3 Analysis and discussion

As illustrated in Figure 4.3, the visualization of the predictions generated by the trained model exhibits a substantial resemblance to that of the ground truth labels. This observation indicates that the trained model in our implementation exhibits a notable level of proficiency in understanding 3D point clouds and executing accurate object part segmentation.

As shown in Tables 4.4 and 4.5, our implementation shows slight differences compared to the results in Point Transformer's paper, with the latter performing slightly better overall. Although we implemented Point Transformer following the paper's description, certain discrepancies may exist at a more detailed level. Moreover, since these two tasks are not the primary focus of our project, we only conducted limited tuning of learning rate hyperparameters. In conclusion, while not perfect, our implementation of Point Transformer has achieved satisfactory results and demonstrates competence in 3D point cloud learning.

4.3 Point Transformer Diffusion

In this section, we will present the experimental results of our method: Point Transformer Diffusion (PTD). Similar to the approach taken with PVD, we evaluated PTD's performance across three categories of the ShapeNet dataset: airplane, chair, and car. In Section 4.3.1, we will present the evaluation results of PTD and compare them with the outcomes produced by various alternative methods. Furthermore, in order to provide a clear assessment of PTD's functionality, we have visualized the point clouds generated by the models. In Section 4.3.2, we will present the results of several ablation studies. These experiments quantitatively validate the structure decisions made for PTD, the choices of activation functions, and the setup of the learning rate warm-up mechanism.

4.3.1 Performance evaluation

As shown in Table 4.6, similar to PVD, PTD outperforms other methods in terms of generation results. Upon comparing PVD and PTD, it becomes evident that PTD surpasses PVD in three entries: EMD for airplanes, CD for chairs, and EMD for cars. In contrast, PVD demonstrates superior results in the remaining three values. Overall, the disparity between the results of PVD and PTD is minimal. Consequently, we can conclude that PTD achieves excellent generation results comparable to those of PVD.

Table 4.6: Generation results on Airplane, Chair, and Car compared with baselines using 1-NNA as the metric. Both CD and EMD serve as distance measures, with lower scores indicating higher quality and greater diversity. (Data source: [74])

	Airplane		Chair		Car	
	CD	EMD	CD	EMD	CD	EMD
r-GAN [1]	98.40	96.79	83.69	99.70	94.46	99.01
l-GAN (CD) [1]	87.30	93.95	68.58	83.84	66.49	88.78
l-GAN (EMD) [1]	89.49	76.91	71.90	64.65	71.16	66.19
PointFlow [67]	75.68	70.74	62.84	60.57	58.10	56.25
SoftFlow [21]	76.05	65.80	59.21	60.05	64.77	60.09
DPF-Net [25]	75.18	65.55	62.00	58.53	62.35	54.48
Shape-GF [5]	80.00	76.17	68.96	65.48	63.20	56.53
PVD [74]	**73.82**	64.81	56.26	**53.32**	**54.55**	53.83
PTD (ours)	74.19	**61.48**	**56.11**	53.39	56.39	**52.69**

Furthermore, Table 4.7 records the epochs at which the three optimal models of PTD were attained. When compared to the epoch values in Tables 4.2 and 4.3, it is evident that the PTD models can also converge to the optimal state within

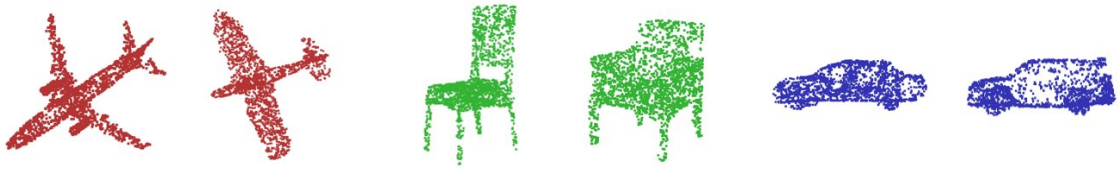
a reasonable number of training epochs, and they may even demonstrate a faster convergence rate than the pre-trained and trained models of PVD.

Table 4.7: Evaluation results of the trained models of PTD.

Category	Model	CD	EMD
airplane	airplane_2799.pth	74.19	61.48
chair	chair_1499.pth	56.11	53.39
car	car_2799.pth	56.39	52.69

* airplane_2799.pth indicates that the optimal model for the airplane category was obtained at epoch 2799. The same naming convention applies to the other two models.

Figure 4.4 demonstrates the PTD models' ability to generate realistic and faithful shapes of airplanes, chairs, and cars. Further visualizations highlighting the diversity of the generated results are available in Figure B.1 in Appendix B.



(a) airplanes **(b)** chairs **(c)** cars

Figure 4.4: Shape generations with 2,048 points using the trained models of PTD. In (a), the two airplanes appear to be a twin-engine jet and a fighter aircraft, respectively. In (b), the left chair is a hollow chair while the right one is an armchair. In (c), the two cars seem to be a sedan and an SUV, respectively.

4.3.2 Ablation study

In the project, we conducted a series of controlled experiments to investigate specific design and application decisions in PTD.

Table 4.8: Ablation study: The number of neighbors k in the definition of local neighborhoods.

k	CD	EMD
16	58.30	54.75
32	**56.11**	**53.39**
64	58.91	58.83

Table 4.9: Ablation study: Downsampling rates for the five stages of the encoder within the Point Transformer model.

Downsampling rates	CD	EMD
[1, 4, 4, 4, 4]	58.45	55.28
[1, 2, 4, 4, 4]	**56.11**	**53.39**

* Setting the downsampling rate to 4 implies that the transition down block in the corresponding stage retains a quarter of the input points. Similarly, a rate of 2 retains half, while a rate of 1 indicates no downsampling in the current stage.

Table 4.10: Ablation study: The numbers of Point Transformer layers in the model.

Structure name	Point Transformer layers	CD	EMD
seg26	encoder:[1, 1, 1, 1, 1] decoder:[1, 1, 1, 1, 1]	60.27	59.13
seg39	encoder:[1, 2, 2, 2, 2] decoder:[1, 1, 2, 2, 1]	58.91	56.34
seg42	encoder:[1, 1, 2, 2, 1] decoder:[1, 2, 2, 2, 2]	**56.11**	**53.39**
seg44	encoder:[1, 2, 2, 2, 2] decoder:[1, 2, 2, 2, 2]	58.15	59.29

Table 4.11: Ablation study: Utilization of the activation function Swish in either the input head, the output head, or in both.

Utilization of Swish	CD	EMD
none	59.13	56.41
input head	58.30	54.98
output head	58.15	54.75
input + output head	**56.11**	**53.39**

Number of neighbors. We begin by investigating the choice of k that plays a role in defining the local neighborhood around each point. The outcomes are presented in Table 4.8, showcasing the optimal performance achieved when k is set to 32. A smaller neighborhood ($k = 16$) might result in inadequate context for accurate predictions. Conversely, a larger neighborhood ($k = 64$) provides numerous data points to each self-attention layer, potentially including distant and less relevant points that could introduce excessive noise and compromise the model's accuracy.

Downsampling rates. We now examine the setting of downsampling rates for the five stages of the encoder in the Point Transformer model. The results are presented in Table 4.9. Keeping the downsampling rates constant for the other four stages, the only variable is the downsampling rate in the first transition down block of the Point Transformer model. A rate of 2 yields better results than 4, emphasizing the importance of preserving more information at this stage.

In the original Point Transformer paper [72], the optimal setting is [1, 4, 4, 4, 4]. However, in our project, this setting has been adjusted to [1, 2, 4, 4, 4]. The variation can be attributed to the distinct characteristics of the tasks at hand. The original paper evaluated Point Transformer models within the context of classification and segmentation tasks. In classification, the model only predicts a single label, resulting in shared predictions across all points. In segmentation, although the model must predict a part label for each point, the number of distinct labels is significantly fewer than the number of points. Consequently, these tasks can tolerate some information loss without degrading performance. However, our project involves training the model for a generation task, where predictions vary significantly between individual points. In this scenario, it becomes crucial to retain more information, especially when the model is learning fine-grained features.

Model structure. We now investigate the model structure, with a focus on the numbers of PT layers employed in the model. Table 4.10 presents the results for four typical structures. The structure *seg26* represents the simplest configuration, where each stage utilizes only one PT layer. Conversely, the *seg44* structure is the most complex among those, employing two successive PT layers at each stage. Notably, both of these structures yield unsatisfactory results. In our analysis, *seg26* is overly simplistic, struggling to capture the intricate patterns and relationships within the data, thus resulting in underfitting. Conversely, *seg44* is overly complex, causing the model to learn noise and random fluctuations in the training data rather than capturing the fundamental patterns, thereby leading to poor generalization ability.

The structures *seg39* and *seg42* possess a moderate level of complexity, lying between that of *seg26* and *seg44*. Both *seg39* and *seg42* employ a total of 16 PT layers, making them equally complex in terms of layer count. The key distinction between these structures is that *seg42* allocates a greater number of PT layers to the decoder, while *seg39* allocates a larger portion to the encoder. Consequently, *seg42* demonstrates superior generation results. This comparison implies that the decoder has a more pronounced positive influence than the encoder within the Point Transformer model for our shape generation task.

Activation function. Continuing with the analysis of the model structure, we explore the impact of replacing ReLU with Swish, either in the input head, the output head, or in both. The outcomes of these investigations are presented in Table 4.11, clearly demonstrating that incorporating Swish in both the input and output heads significantly enhances the model's performance. These controlled experiments further solidify the advantages of Swish over ReLU, as elucidated in Section 3.3.2,

within the specific context of our shape generation task.

The ablation studies discussed above were conducted on the chair category of the ShapeNet dataset. However, it is important to note that the models trained for airplanes or cars both adhere to the same settings. This decision is grounded in the understanding that these model-related hyperparameters possess generalization ability across diverse datasets. By maintaining consistent settings, we aim to ascertain the broader applicability of our findings and conclusions beyond a single category.

Warm-up period. In contrast, separate ablation studies were conducted for the learning rate warm-up mechanism in the chair, airplane, and car categories due to two main factors. Firstly, these categories have different sample counts in the training dataset, resulting in distinct iteration numbers per epoch with the same batch size. For instance, the chair category comprises 4,612 training samples, while the airplane category has 2,832. With a batch size of 16, this translates to 288 iterations per epoch for chairs and 177 for airplanes, probably leading to varying optimal warm-up periods. Secondly, these categories have unique learning rate settings. For example, as seen in Table 4.7, the model for the car category requires more epochs to converge compared to the chair category. Consequently, the learning rate schedule for the car model is adjusted to decay more slowly for effective learning at later epochs. Given different learning rates during training, it is expected that the optimal warm-up periods for these models will also differ.

Table 4.12: Ablation study: Learning rate warm-up period on the chair category.

Warm-up period	CD	EMD
0 (not used)	58.31	55.51
5	**56.11**	**53.39**
10	57.32	53.47
25	58.38	55.51
50	57.85	56.43

Table 4.13: Ablation study: Learning rate warm-up period on the airplane category.

Warm-up period	CD	EMD
0 (not used)	75.55	61.48
5	75.67	63.33
10	**74.19**	61.48
15	75.67	63.82
20	75.80	**61.23**

Table 4.14: Ablation study: Learning rate warm-up period on the car category.

Warm-up period	CD	EMD
0 (not used)	57.10	55.82
5	**56.39**	**52.69**
10	58.09	54.68
15	60.36	54.40
20	56.96	55.96

As shown in Tables 4.12, 4.13, and 4.14, the warm-up period mechanism enhances the models across all three categories, each requiring a few warm-up epochs. The optimal warm-up periods for both the chair model and the car model are 5. The situation for the airplane model is slightly more complex: the optimal CD and EMD are not achieved simultaneously. Specifically, the optimal CD is attained with a warm-up period of 10, while the optimal EMD is reached with a warm-up period of 20. By averaging the CD and EMD values, an overall better performance is observed with a warm-up period of 10 compared to 20. Consequently, the airplane model's optimal warm-up period is determined to be 10.

4.3.3 Analysis and discussion

As illustrated in Figure 4.4, the trained PTD models demonstrate the capability to generate realistic and high-quality shapes of airplanes, chairs, and cars. Furthermore, the evaluation results of these models align with those of PVD, as indicated in Table 4.6. This consistency enables us to confidently assert that PTD stands as a successful approach for 3D point cloud shape generation. Moreover, we will delve into the method's potential and address the limitations observed in the experiments.

Despite the excellence of the generated results from PTD being comparable to those of PVD, they do not attain theoretical optimality due to two main factors. Firstly, the evaluations of the models are performed at 100-epoch intervals, resulting in an evaluation frequency of $\frac{1}{100}$. By increasing the evaluation frequency, the possibility of identifying a model with superior results is heightened, given the larger pool of potential optimal models. Secondly, the ablation studies are not comprehensive enough. For example, apart from the four model structures presented in Table 4.10, many other structures remain unexplored. If we design and conduct more ablation studies, it could be possible to discover better settings for the PTD models.

Meanwhile, it is important to consider the limitations when drawing conclusions from the experimental results. In other words, some conclusions based on the experimental results may not always be universally applicable but are contingent on certain preconditions. Let us take the ablation study regarding the learning rate warm-up mechanism as an example. As demonstrated in Tables 4.12, 4.13, and 4.14, the learning rate warm-up mechanism enhances the performance of the three PTD models. While we may initially conclude that the warm-up mechanism has

a positive effect on improving PTD models, this conclusion lacks thorough rigor. This is because our evaluations are conducted at 100-epoch intervals. Taking this constraint into account, a more rigorous conclusion would state that the warm-up mechanism exhibits a positive effect, given the evaluation frequency is $\frac{1}{100}$.

5
Conclusion

Diffusion models are a novel class of generative models that have already been successfully applied to numerous generation tasks. Meanwhile, Transformer-based methods have revolutionized natural language processing and have also made remarkable progress in 2D image analysis and 3D point cloud understanding. However, the exploration of combining a diffusion model with a local self-attention Transformer for 3D point cloud generation remains uncharted. In this thesis, we introduce PTD, which refines a Point Transformer model and integrates it with a DDPM formulated in 3D space. Through experimentation, our method has proven capable of generating realistic and diverse shapes of airplanes, chairs, and cars in 3D point clouds. Moreover, the evaluation results are comparable, and in some cases, even slightly superior to those achieved by PVD.

Beyond combining a Point Transformer model with a DDPM, we have adopted several techniques, including adjustments to the model structure and hyperparameters, the utilization of Swish, and the implementation of the learning rate warm-up mechanism. All of these techniques have contributed to improving the performance of PTD models. The reasons behind the success of these techniques vary. It could be that the technique is more advanced than the original corresponding component, as seen in the comparison between Swish and ReLU. Alternatively, the success could be attributed to the distinct characteristics of the tasks at hand. For instance, a deeper decoder proves to be more effective than a deeper encoder in the shape generation task in our project.

PTD currently focuses on the task of unconditional shape generation. While it has achieved success in this domain, further validation of its performance is required across other 3D generation tasks, including conditional shape generation, shape completion, and the generation of colored or textured shapes. Moreover, the adjustments and techniques applied to PTD primarily center around refining the Point Transformer model to better suit the diffusion model or the specific generation task at hand. It would be interesting to consider certain modifications from the perspective of the diffusion model to further enhance PTD's performance. We anticipate that our work will inspire future investigations into architectures that combine diffusion models and Transformers, along with their application to a wide range of 3D generation tasks.

Bibliography

[1] Panos Achlioptas et al. "Learning representations and generative models for 3d point clouds". In: *International conference on machine learning*. PMLR. 2018, pp. 40–49.

[2] Yang You et al. *Pytorch Implementation of Various Point Transformers*. `https://github.com/qq456cvb/Point-Transformers`. 2021.

[3] Jianmin Bao et al. "CVAE-GAN: fine-grained image generation through asymmetric training". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2745–2754.

[4] John Bridle. "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters". In: *Advances in neural information processing systems* 2 (1989).

[5] Ruojin Cai et al. "Learning gradient fields for shape generation". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer. 2020, pp. 364–381.

[6] Angel X Chang et al. "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012* (2015).

[7] Pointcept Contributors. *Pointcept: A Codebase for Point Cloud Perception Research*. `https://github.com/Pointcept/Pointcept`. 2023.

[8] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[9] Prafulla Dhariwal and Alexander Nichol. "Diffusion models beat gans on image synthesis". In: *Advances in neural information processing systems* 34 (2021), pp. 8780–8794.

[10] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[11] Zhaoxin Fan et al. "Svt-net: Super light-weight sparse voxel transformer for large scale place recognition". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 1. 2022, pp. 551–560.

[12] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[13] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[14] Chenhang He et al. "Voxel set transformer: A set-to-set approach to 3d object detection from point clouds". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 8417–8427.

[15] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[16] Kaiming He et al. “Masked autoencoders are scalable vision learners”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 16000–16009.

[17] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

[18] Jonathan Ho et al. “Cascaded diffusion models for high fidelity image generation”. In: *The Journal of Machine Learning Research* 23.1 (2022), pp. 2249–2281.

[19] Ka-Hei Hui et al. “Neural wavelet-domain diffusion for 3d shape generation”. In: *SIGGRAPH Asia 2022 Conference Papers*. 2022, pp. 1–9.

[20] Tero Karras, Samuli Laine, and Timo Aila. “A style-based generator architecture for generative adversarial networks”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

[21] Hyeongju Kim et al. “Softflow: Probabilistic framework for normalizing flow on manifolds”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 16388–16397.

[22] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[23] Diederik P Kingma, Max Welling, et al. “An introduction to variational autoencoders”. In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[24] Durk P Kingma and Prafulla Dhariwal. “Glow: Generative flow with invertible 1x1 convolutions”. In: *Advances in neural information processing systems* 31 (2018).

[25] Roman Klokov, Edmond Boyer, and Jakob Verbeek. “Discrete point flow networks for efficient point cloud generation”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 694–710.

[26] Roman Klokov and Victor Lempitsky. “Escape from cells: Deep kd-networks for the recognition of 3d point cloud models”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 863–872.

[27] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[28] Xin Lai et al. “Stratified transformer for 3d point cloud segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 8500–8509.

[29] Chun-Liang Li et al. “Point cloud gan”. In: *arXiv preprint arXiv:1810.05795* (2018).

[30] Ze Liu et al. “Swin transformer: Hierarchical vision transformer using shifted windows”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10012–10022.

[31] Zhijian Liu et al. “Point-voxel cnn for efficient 3d deep learning”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[32] David Lopez-Paz and Maxime Oquab. “Revisiting classifier two-sample tests”. In: *arXiv preprint arXiv:1610.06545* (2016).

[33] David G Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[34] Calvin Luo. “Understanding diffusion models: A unified perspective”. In: *arXiv preprint arXiv:2208.11970* (2022).

[35] Shitong Luo and Wei Hu. “Diffusion probabilistic models for 3d point cloud generation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2837–2845.

[36] Simian Luo et al. “Learning Versatile 3D Shape Generation with Improved AR Models”. In: *arXiv preprint arXiv:2303.14700* (2023).

[37] Zhaoyang Lyu et al. “A conditional point diffusion-refinement paradigm for 3d point cloud completion”. In: *arXiv preprint arXiv:2112.03530* (2021).

[38] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[39] Jiageng Mao et al. “Voxel transformer for 3d object detection”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 3164–3173.

[40] Alex Nichol et al. “Glide: Towards photorealistic image generation and editing with text-guided diffusion models”. In: *arXiv preprint arXiv:2112.10741* (2021).

[41] Alex Nichol et al. “Point-e: A system for generating 3d point clouds from complex prompts”. In: *arXiv preprint arXiv:2212.08751* (2022).

[42] Alexander Quinn Nichol and Prafulla Dhariwal. “Improved denoising diffusion probabilistic models”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8162–8171.

[43] Yatian Pang et al. “Masked autoencoders for point cloud self-supervised learning”. In: *European conference on computer vision*. Springer. 2022, pp. 604–621.

[44] Charles R Qi et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[45] Charles Ruizhongtai Qi et al. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space”. In: *Advances in neural information processing systems* 30 (2017).

[46] Prajit Ramachandran, Barret Zoph, and Quoc V Le. “Swish: a self-gated activation function”. In: *arXiv preprint arXiv:1710.05941* 7.1 (2017), p. 5.

[47] Sebastian Raschka and Vahid Mirjalili. *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

[48] Danilo Rezende and Shakir Mohamed. “Variational inference with normalizing flows”. In: *International conference on machine learning*. PMLR. 2015, pp. 1530–1538.

[49] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. “Stochastic backpropagation and approximate inference in deep generative models”. In: *International conference on machine learning*. PMLR. 2014, pp. 1278–1286.

[50] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. "Octnet: Learning deep 3d representations at high resolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3577–3586.

[51] Robin Rombach et al. "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[52] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241.

[53] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. "The earth mover's distance as a metric for image retrieval". In: *International journal of computer vision* 40 (2000), pp. 99–121.

[54] Jascha Sohl-Dickstein et al. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 2256–2265.

[55] Hugues Thomas et al. "Kpconv: Flexible and deformable convolution for point clouds". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6411–6420.

[56] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[57] Lei Wang et al. "Graph attention convolution for point cloud semantic segmentation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 10296–10305.

[58] Peng-Shuai Wang et al. "O-cnn: Octree-based convolutional neural networks for 3d shape analysis". In: *ACM Transactions On Graphics (TOG)* 36.4 (2017), pp. 1–11.

[59] Shenlong Wang et al. "Deep parametric continuous convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2589–2597.

[60] Yue Wang et al. "Dynamic graph cnn for learning on point clouds". In: *ACM Transactions on Graphics (tog)* 38.5 (2019), pp. 1–12.

[61] Zhirong Wu et al. "3d shapenets: A deep representation for volumetric shapes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[62] Chong Xiang, Charles R Qi, and Bo Li. "Generating 3d adversarial point clouds". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9136–9144.

[63] Yifan Xu et al. "Spidercnn: Deep learning on point sets with parameterized convolutional filters". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 87–102.

[64] Yusheng Xu, Xiaohua Tong, and Uwe Stilla. "Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry". In: *Automation in Construction* 126 (2021), p. 103675.

[65] Xu Yan. *Pointnet/Pointnet++ Pytorch*. `https://github.com/yanx27/Pointnet_Pointnet2_pytorch`. 2019.

[66] Xu Yan. *Pointnet/Pointnet++ Pytorch*. 2019. URL: `https://github.com/yanx27/Pointnet_Pointnet2_pytorch` (visited on 07/09/2023).

[67] Guandao Yang et al. "Pointflow: 3d point cloud generation with continuous normalizing flows". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 4541–4550.

[68] Li Yi et al. "A scalable active framework for region annotation in 3d shape collections". In: *ACM Transactions on Graphics (ToG)* 35.6 (2016), pp. 1–12.

[69] Xiaohui Zeng et al. "LION: Latent point diffusion models for 3D shape generation". In: *arXiv preprint arXiv:2210.06978* (2022).

[70] Cheng Zhang et al. "PVT: Point-voxel transformer for point cloud learning". In: *International Journal of Intelligent Systems* 37.12 (2022), pp. 11985–12008.

[71] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. "Exploring self-attention for image recognition". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10076–10085.

[72] Hengshuang Zhao et al. "Point transformer". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 16259–16268.

[73] Hengshuang Zhao et al. "PointWeb: Enhancing Local Neighborhood Features for Point Cloud Processing". In: *CVPR*. 2019.

[74] Linqi Zhou, Yilun Du, and Jiajun Wu. "3d shape generation and completion through point-voxel diffusion". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5826–5835.

A

Extended derivations

Below, you will find the complete version of Eq. (2.25):

$$
\begin{aligned}
\mathcal{L} &= \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \\
&= \mathbb{E}_q\left[-\log \frac{p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{\prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] \\
&= \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t>1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} - \log \frac{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)}\right] \\
&= \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t>1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)} - \log \frac{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)}\right] \\
&= \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t>1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)} \cdot \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} - \log \frac{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)}\right] \\
&= \mathbb{E}_q\left[-\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_0)} - \sum_{t>1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)} - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] \\
&= \mathbb{E}_q\left[D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \,||\, p(\mathbf{x}_T)) + \sum_{t>1} D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right],
\end{aligned}
\tag{A.1}
$$

which represents the reduced variance variational bound specifically designed for diffusion models. The derivation and the corresponding insights are sourced from the work of Sohl-Dickstein et al. [54].

Below, you will find a detailed derivation of Eq. (2.26):

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0)q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \tag{A.2}$$

$$= \frac{\mathcal{N}(\mathbf{x}_t;\sqrt{\alpha_t}\mathbf{x}_{t-1},(1-\alpha_t)\mathbf{I})\mathcal{N}(\mathbf{x}_{t-1};\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0,(1-\bar{\alpha}_{t-1})\mathbf{I})}{\mathcal{N}(\mathbf{x}_t;\sqrt{\bar{\alpha}_t}\mathbf{x}_0,(1-\bar{\alpha}_t)\mathbf{I})}$$

$$\propto \exp\left\{-\frac{1}{2}\left[\frac{(\mathbf{x}_t-\sqrt{\alpha_t}\mathbf{x}_{t-1})^2}{1-\alpha_t}+\frac{(\mathbf{x}_{t-1}-\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0)^2}{1-\bar{\alpha}_{t-1}}-\frac{(\mathbf{x}_t-\sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t}\right]\right\}$$

$$= \exp\left\{-\frac{1}{2}\left[\frac{-2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1}+\alpha_t\mathbf{x}_{t-1}^2}{1-\alpha_t}+\frac{\mathbf{x}_{t-1}^2-2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_{t-1}\mathbf{x}_0}{1-\bar{\alpha}_{t-1}}+C(\mathbf{x}_t,\mathbf{x}_0)\right]\right\} \tag{A.3}$$

$$\propto \exp\left\{-\frac{1}{2}\left[-\frac{2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1}}{1-\alpha_t}+\frac{\alpha_t\mathbf{x}_{t-1}^2}{1-\alpha_t}+\frac{\mathbf{x}_{t-1}^2}{1-\bar{\alpha}_{t-1}}-\frac{2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_{t-1}\mathbf{x}_0}{1-\bar{\alpha}_{t-1}}\right]\right\}$$

$$= \exp\left\{-\frac{1}{2}\left[\left(\frac{\alpha_t}{1-\alpha_t}+\frac{1}{1-\bar{\alpha}_{t-1}}\right)\mathbf{x}_{t-1}^2-2\left(\frac{\sqrt{\alpha_t}\mathbf{x}_t}{1-\alpha_t}+\frac{\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0}{1-\bar{\alpha}_{t-1}}\right)\mathbf{x}_{t-1}\right]\right\}$$

$$= \exp\left\{-\frac{1}{2}\left[\frac{1-\bar{\alpha}_t}{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}\mathbf{x}_{t-1}^2-2\left(\frac{\sqrt{\alpha_t}\mathbf{x}_t}{1-\alpha_t}+\frac{\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0}{1-\bar{\alpha}_{t-1}}\right)\mathbf{x}_{t-1}\right]\right\}$$

$$= \exp\left\{-\frac{1}{2}\left(\frac{1-\bar{\alpha}_t}{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}\right)\left[\mathbf{x}_{t-1}^2-2\,\frac{\frac{\sqrt{\alpha_t}\mathbf{x}_t}{1-\alpha_t}+\frac{\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0}{1-\bar{\alpha}_{t-1}}}{\frac{1-\bar{\alpha}_t}{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}}\,\mathbf{x}_{t-1}\right]\right\}$$

$$= \exp\left\{-\frac{1}{2}\left(\frac{1}{\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}}\right)\left[\mathbf{x}_{t-1}^2-2\,\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{x}_t+\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\mathbf{x}_0}{1-\bar{\alpha}_t}\,\mathbf{x}_{t-1}\right]\right\}$$

$$\propto \mathcal{N}(\mathbf{x}_{t-1};\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{x}_t+\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\mathbf{x}_0}{1-\bar{\alpha}_t},\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{I}) \tag{A.4}$$

$$= \mathcal{N}(\mathbf{x}_{t-1};\boldsymbol{\mu}_q(\mathbf{x}_t,\mathbf{x}_0),\tilde{\beta}_t\mathbf{I}), \tag{A.5}$$

which is based on the work by Luo [34]. The term $C(\mathbf{x}_t,\mathbf{x}_0)$ shown in Eq. (A.3) is a constant that depends only on $\mathbf{x}_t$, $\mathbf{x}_0$, and α values. This term is implicitly returned in Eq. (A.4) to facilitate the completion of the square.

Below, you will find a thorough derivation of Eq. (2.30):

$$
\begin{aligned}
L_{t-1} &= D_{\mathrm{KL}}(\mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}) \,||\, \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})) \\
&= \frac{1}{2}\Bigg[\log \frac{\sigma_t^2}{\tilde{\beta}_t} - d + \mathrm{tr}\Big((\sigma_t^2 \mathbf{I})^{-1} \tilde{\beta}_t \mathbf{I}\Big) \\
&\qquad + (\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) - \boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0))^T (\sigma_t^2 \mathbf{I})^{-1} (\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) - \boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0))\Bigg] \\
&= \frac{1}{2}\Bigg[(1 - d + d + C) \\
&\qquad + (\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) - \boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0))^T (\sigma_t^2 \mathbf{I})^{-1} (\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) - \boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0))\Bigg] \\
&= \frac{1}{2}\Big[(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) - \boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0))^T (\sigma_t^2 \mathbf{I})^{-1} (\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) - \boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0))\Big] + C \\
&= \frac{1}{2\sigma_t^2} ||\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)||^2 + C \\
&\propto \frac{1}{2\sigma_t^2} ||\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)||^2,
\end{aligned}
\tag{A.6}
$$

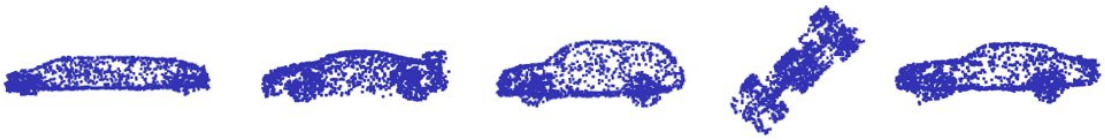
where C is a constant term that does not rely on $\boldsymbol{\theta}$.

B

Visualizations

(a) airplanes

(b) chairs



(c) cars

Figure B.1: Shape generations with 2,048 points using the trained models of PTD.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY