

# PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows

Guandao Yang<sup>1,2\*</sup>, Xun Huang<sup>1,2\*</sup>, Zekun Hao<sup>1,2</sup>, Ming-Yu Liu<sup>3</sup>, Serge Belongie<sup>1,2</sup>, Bharath Hariharan<sup>1</sup>  
<sup>1</sup>Cornell University    <sup>2</sup>Cornell Tech    <sup>3</sup>NVIDIA

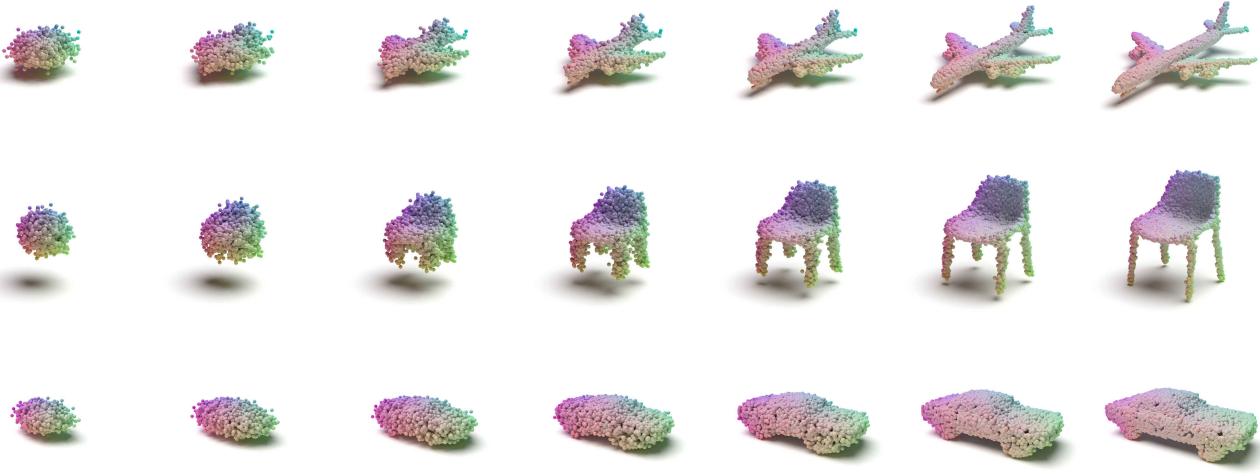


Figure 1: Our model transforms points sampled from a simple prior to realistic point clouds through continuous normalizing flows. The videos of the transformations can be viewed on our project website: <https://www.guandaoyang.com/PointFlow/>.

## Abstract

As 3D point clouds become the representation of choice for multiple vision and graphics applications, the ability to synthesize or reconstruct high-resolution, high-fidelity point clouds becomes crucial. Despite the recent success of deep learning models in discriminative tasks of point clouds, generating point clouds remains challenging. This paper proposes a principled probabilistic framework to generate 3D point clouds by modeling them as a distribution of distributions. Specifically, we learn a two-level hierarchy of distributions where the first level is the distribution of shapes and the second level is the distribution of points given a shape. This formulation allows us to both sample shapes and sample an arbitrary number of points from a shape. Our generative model, named PointFlow, learns each level of the distribution with a continuous normalizing flow. The invertibility of normalizing flows enables the computation of the likelihood during training and allows

us to train our model in the variational inference framework. Empirically, we demonstrate that PointFlow achieves state-of-the-art performance in point cloud generation. We additionally show that our model can faithfully reconstruct point clouds and learn useful representations in an unsupervised manner. The code will be available at <https://github.com/stevenygd/PointFlow>.

## 1. Introduction

Point clouds are becoming popular as a 3D representation because they can capture a much higher resolution than voxel grids and are a stepping stone to more sophisticated representations such as meshes. Learning a generative model of point clouds could benefit a wide range of point cloud synthesis tasks such as reconstruction and super-resolution, by providing a better *prior* of point clouds. However, a major roadblock in generating point clouds is the complexity of the space of point clouds. A cloud of points corresponding to a chair is best thought of as sam-

\*Equal contribution.

ples from a distribution that corresponds to the surface of the chair, and the chair itself is best thought of as a sample from a distribution of chair shapes. As a result, in order to generate a chair according to this formulation, we need to characterize a *distribution of distributions*, which is under-explored by existing generative models.

In this paper, we propose PointFlow, a principled generative model for 3D point clouds that learns a distribution of distributions: the former being the distribution of shapes and the latter being the distribution of points given a shape. Our key insight is that instead of directly parametrizing the distribution of points in a shape, we model this distribution as an *invertible parameterized transformation* of 3D points from a prior distribution (e.g., a 3D Gaussian). Intuitively, under this model, generating points for a given shape involves sampling points from a generic Gaussian prior, and then *moving* them according to this parameterized transformation to their new location in the target shape, as illustrated in Figure 1. In this formulation, a given shape is then simply the variable that parametrizes such transformation, and a category is simply a distribution of this variable. Interestingly, we find that representing this distribution too as a transformation of a prior distribution leads to a more expressive model of shapes. In particular, we use the recently proposed continuous normalizing flow framework to model both kinds of transformations [40, 5, 16].

This parameterization confers several advantages. The invertibility of these transformations allows us to not just sample but also estimate probability densities. The ability to estimate probability densities in turn allows us to train these models in a principled manner using the variational inference framework [27], where we maximize a variational lower bound on the log likelihood of a training set of point clouds. This probabilistic framework for training further lets us avoid the complexities of training GANs or hand-crafting good distance metrics for measuring the difference between two sets of points. Experiments show that PointFlow outperforms previous state-of-the-art generative models of point clouds, and achieves compelling results in point cloud reconstruction and unsupervised feature learning.

## 2. Related work

**Deep learning for point clouds.** Deep learning has been introduced to improve performance in various point cloud discriminative tasks including classification [38, 39, 51, 55], segmentation [38, 43], and critical points sampling [10]. Recently, substantial progress has been made in point cloud synthesis tasks such as auto-encoding [1, 51, 17], single-view 3D reconstruction [12, 22, 29, 31, 13], stereo reconstruction [45], and point cloud completion [54, 53]. Many point cloud synthesis works convert a point distribution to a  $N \times 3$  matrix by sampling  $N$  ( $N$  is pre-defined) points from the distribution so that existing generative models are

readily applicable. For example, Gadelha *et al.* [13] apply variational auto-encoders (VAEs) [27] and Zamorski *et al.* [56] apply adversarial auto-encoders (AAEs) [34] to point cloud generation. Achlioptas *et al.* [1] explore generative adversarial networks (GANs) [15, 2, 19] for point clouds in both raw data space and latent space of a pre-trained auto-encoder. In the above methods, the auto-encoders are trained with heuristic loss functions that measure the distance between two point sets, such as Chamfer distance (CD) or earth mover’s distance (EMD). Sun *et al.* [44] apply auto-regressive models [47] with a discrete point distribution to generate one point at a time, also using a fixed number of points per shape.

However, treating a point cloud as a fixed-dimensional matrix has several drawbacks. First, the model is restricted to generate a fixed number of points. Getting more points for a particular shape requires separate up-sampling models such as [54, 53, 52]. Second, it ignores the permutation invariance property of point sets, which might lead to suboptimal parameter efficiency. Heuristic set distances are also far from ideal objectives from a generative modeling perspective since they make the original probabilistic interpretation of VAE/AAE no longer applicable when used as the reconstruction objective. In addition, exact EMD is slow to compute while approximations could lead to biased or noisy gradients. CD has been shown to incorrectly favor point clouds that are overly concentrated in the mode of the marginal point distribution [1].

Some recent works introduce sophisticated decoders consisting of a cascade [51] or a mixture [17] of smaller decoders to map one (or a mixture of) 2-D uniform distribution(s) to the target point distribution, overcoming the shortcomings of using a fixed number of points. However, they still rely on heuristic set distances that lack a probabilistic guarantee. Also, their methods only learn the distribution of points for each shape, but not the distribution of shapes. Li *et al.* [30] propose a “sandwiching” reconstruction objective that combines a variant of WGAN [2] loss with EMD. They also train another GAN in the latent space to learn shape distribution, similar to Achlioptas *et al.* [1]. In contrast, our method is simply trained end-to-end by maximizing a variational lower bound on the log-likelihood, does not require multi-stage training, and does not have any instability issues common for GAN based methods.

**Generative models.** There are several popular frameworks of deep generative models, including generative adversarial networks [15, 2, 23], variational auto-encoders [27, 41], auto-regressive models [35, 47], and flow-based models [8, 40, 9, 25]. In particular, flow-based models and auto-regressive models can both perform exact likelihood evaluation, while flow-based models are much more efficient to sample from. Flow-based models have been successfully applied to a variety of generation tasks such as

image generation [25, 9, 8], video generation [28], and voice synthesis [37]. Also, there has been recent work that combines flows with other generative models, such as GAN [18, 7], auto-regressive models [20, 36, 26], and VAEs [26, 46, 6, 40, 46, 5, 16].

Most existing deep generative models aim at learning the distribution of fixed-dimensional variables. Learning the *distribution of distributions*, where the data consists of a *set of sets*, is still under-explored. Edwards and Storkey [11] propose a hierarchical VAE named Neural Statistician that consumes a set of sets. They are mostly interested in the few-shot case where each set only has a few samples. Also, they are focused on classifying sets or generating new samples from a given set. While our method is also applicable to these tasks, our focus is on learning the distribution of sets and generating new sets (point clouds in our case). In addition, our model employs a tighter lower bound on the log-likelihood, thanks to the use of normalizing flow in modeling both the reconstruction likelihood and the prior.

### 3. Overview

Consider a set of shapes  $\mathcal{X} = \{X_i\}_{i=1}^N$  from a particular class of object, where each shape is represented as a set of 3D points  $X_i = \{x_j^i\}_{j=1}^{M_i}$ . As discussed in Section 1, each point  $x_j^i \in \mathbb{R}^3$  is best thought of as being sampled from a point distribution  $Q^i(x)$ , usually a uniform distribution over the surface of an object  $X_i$ . Each shape  $X_i$  is itself a sample from a distribution over shapes  $Q(X)$  that captures what shapes in this category look like.

Our goal is to learn the distribution of shapes, each itself being a distribution of points. In other words, our generative model should be able to both sample shapes and sample an arbitrary number of points from a shape.

We propose to use continuous normalizing flows to model the distribution of points given a shape. A continuous normalizing flow can be thought of as a vector field in the 3-D Euclidean space, which induces a distribution of points through transforming a generic prior distribution (*e.g.*, a standard Gaussian). To sample points from the induced distribution, we simply sample points from the prior and move them according to the vector field. Moreover, the continuous normalizing flow is invertible, which means we can move data points back to the prior distribution to compute the exact likelihood. This model is highly intuitive and interpretable, allowing a close inspection of the generative process as shown in Figure 1.

We parametrize each continuous normalizing flow with a latent variable that represents the shape. As a result, modeling the distribution of shapes can be reduced to modeling the distribution of the latent variable. Interestingly, we find continuous normalizing flow also effective in modeling the latent distribution. Our full generative model thus consists

of two levels of continuous normalizing flows, one modeling the shape distribution by modeling the distribution of the latent variable, and the other modeling the point distribution given a shape.

In order to optimize the generative model, we construct a variational lower bound on the log-likelihood by introducing an inference network that infers a latent variable distribution from a point cloud. Here, we benefit from the fact that the invertibility of the continuous normalizing flow enables likelihood computation. This allows us to train our model end-to-end in a stable manner, unlike previous work based on GANs that requires two-stage training [1, 30]. As a side benefit, we find the inference network learns a useful representation of point clouds in an unsupervised manner.

In Section 4 we introduce some background on continuous normalizing flows and variational auto-encoders. We then describe our model and training in detail in Section 5.

## 4. Background

### 4.1. Continuous normalizing flow

A normalizing flow [40] is a series of invertible mappings that transform an initial known distribution to a more complicated one. Formally, let  $f_1, \dots, f_n$  denote a series of invertible transformations we want to apply to a latent variable  $y$  with distribution  $P(y)$ .  $x = f_n \circ f_{n-1} \circ \dots \circ f_1(y)$  is the output variable. Then the density of the output variable is given by the change of variables formula:

$$\log P(x) = \log P(y) - \sum_{i=1}^n \log \left| \det \frac{\partial f_k}{\partial y_{k-1}} \right|, \quad (1)$$

where  $y$  can be computed from  $x$  using the inverse flow:  $y = f_1^{-1} \circ \dots \circ f_n^{-1}(x)$ . In practice,  $f_1, \dots, f_n$  are usually instantiated as neural networks with an architecture that makes the determinant of the Jacobian  $\left| \det \frac{\partial f_k}{\partial y_{k-1}} \right|$  easy to compute. The normalizing flow has been generalized from a discrete sequence to a continuous transformation [16, 5] by defining the transformation  $f$  using a continuous time dynamic  $\frac{\partial y(t)}{\partial t} = f(y(t), t)$ , where  $f$  is a neural network that has an unrestricted architecture. The continuous normalizing flow (CNF) model for  $P(x)$  with a prior distribution  $P(y)$  at the start time can then be written as:

$$x = y(t_0) + \int_{t_0}^{t_1} f(y(t), t) dt, \quad y(t_0) \sim P(y) \\ \log P(x) = \log P(y(t_0)) - \int_{t_0}^{t_1} \text{Tr} \left( \frac{\partial f}{\partial y(t)} \right) dt \quad (2)$$

and  $y(t_0)$  can be computed using the inverse flow  $y(t_0) = x + \int_{t_1}^{t_0} g(y(t), t) dt$ . A black-box ordinary differential equation (ODE) solver can be applied to estimate the outputs and the input gradients of a continuous normalizing flow [16, 5].

## 4.2. Variational auto-encoder

Suppose we have a random variable  $X$  that we are building generative models for. The variational auto-encoder (VAE) is a framework that allows one to learn  $P(X)$  from a dataset of observations of  $X$  [27, 41]. The VAE does the generation via a latent variable  $z$  with a prior distribution  $P_\psi(z)$ , and a decoder  $P_\theta(X|z)$  which captures the (hopefully simpler) distribution of  $X$  given  $z$ . At test time, the latent variable  $z$  is sampled from the prior  $P_\psi(z)$  and then the decoder is used to sample  $X$  conditioned on  $z$ .

The VAE is trained on a set of observations of  $X$ . During training, it additionally learns an inference model (or encoder)  $Q_\phi(z|X)$ . The encoder and decoder are jointly trained to maximize a lower bound on the log-likelihood of the observed variable

$$\begin{aligned} \log P_\theta(X) &\geq \log P_\theta(X) - D_{KL}(Q_\phi(z|X)||P_\theta(z|X)) \\ &= \mathbb{E}_{Q_\phi(z|x)} [\log P_\theta(X|z)] - D_{KL}(Q_\phi(z|X)||P_\psi(z)) \\ &\triangleq \mathcal{L}(X; \phi, \psi, \theta), \end{aligned} \quad (3)$$

which is also called the evidence lower bound (ELBO). One can interpret ELBO as the sum of the negative reconstruction error (the first term) and a latent space regularizer (the second term). In practice,  $Q_\phi(z|X)$  is usually modeled as a diagonal Gaussian  $\mathcal{N}(z|\mu_\phi(X), \sigma_\phi(X))$  whose mean and standard deviation are predicted by a neural network with parameter  $\phi$ . To efficiently optimize the ELBO, sampling from  $Q_\phi(z|X)$  can be done by reparametrizing  $z$  as  $z = \mu_\phi(X) + \sigma_\phi(X) \cdot \epsilon$ , where  $\epsilon \sim \mathcal{N}(0, I)$ .

## 5. Model

We now have the paraphernalia needed to define our generative model of point clouds. Using the terminology of the VAE, we need three modules: the encoder  $Q_\phi(z|X)$  that encodes a point cloud into a shape representation  $z$ , a prior  $P_\psi(z)$  over shape representations, and a decoder  $P_\theta(X|z)$  that models the distribution of points given the shape representation. We use a simple permutation-invariant encoder to predict  $Q_\phi(z|X)$ , following the architecture in Achlioptas *et al.* [1]. We use continuous normalizing flows for both the prior  $P_\psi(z)$  and the generator  $P_\theta(X|z)$ , which are described below.

### 5.1. Flow-based point generation from shape representations

We first decompose the reconstruction log-likelihood of a point set into the sum of log-likelihood of each point

$$\log P_\theta(X|z) = \sum_{x \in X} \log P_\theta(x|z). \quad (4)$$

We propose to model  $P_\theta(x|z)$  using a conditional extension of CNF. Specifically, a point  $x$  in the point set  $X$  is the result

of transforming some point  $y(t_0)$  in the prior distribution  $P(y) = \mathcal{N}(0, I)$  using a CNF conditioned on  $z$ :

$$x = G_\theta(y(t_0); z) \triangleq y(t_0) + \int_{t_0}^{t_1} g_\theta(y(t), t, z) dt, y(t_0) \sim P(y),$$

where  $g_\theta$  is the continuous-time dynamics of the flow  $G_\theta$  conditioned on  $z$ . Note that the inverse of  $G_\theta$  is given by  $G_\theta^{-1}(x; z) = x + \int_{t_1}^{t_0} g_\theta(y(t), t, z) dt$  with  $y(t_1) = x$ . The reconstruction likelihood of  $x$  given  $z$  follows equation (2):

$$\log P_\theta(x|z) = \log P(G_\theta^{-1}(x; z)) - \int_{t_0}^{t_1} \text{Tr} \left( \frac{\partial g_\theta}{\partial y(t)} \right) dt. \quad (5)$$

Note that  $\log P(G_\theta^{-1}(x; z))$  can be computed in closed form with the Gaussian prior.

### 5.2. Flow-based prior over shapes

Although it is possible to use a simple Gaussian prior over shape representations, it has been shown that a restricted prior tends to limit the performance of VAEs [6]. To alleviate this problem, we use another CNF to parametrize a learnable prior. Formally, we rewrite the KL divergence term in Equation 3 as

$$D_{KL}(Q_\phi(z|x)||P_\psi(z)) = -\mathbb{E}_{Q_\phi(z|x)} [\log P_\psi(z)] - H[Q_\phi(z|X)], \quad (6)$$

where  $H$  is the entropy and  $P_\psi(z)$  is the prior distribution with learnable parameters  $\psi$ , obtained by transforming a simple Gaussian  $P(w) = \mathcal{N}(0, I)$  with a CNF:

$$z = F_\psi(w(t_0)) \triangleq w(t_0) + \int_{t_0}^{t_1} f_\psi(w(t), t) dt, w(t_0) \sim P(w),$$

where  $f_\psi$  is the continuous-time dynamics of the flow  $F_\psi$ . Similarly as described above, the inverse of  $F_\psi$  is given by  $F_\psi^{-1}(z) = z + \int_{t_1}^{t_0} f_\psi(w(t), t) dt$  with  $w(t_1) = z$ . The log probability of the prior distribution can be computed by:

$$\log P_\psi(z) = \log P(F_\psi^{-1}(z)) - \int_{t_0}^{t_1} \text{Tr} \left( \frac{\partial f_\psi}{\partial w(t)} \right) dt. \quad (7)$$

### 5.3. Final training objective

Plugging Equation 4, 5, 6, 7 into Equation 3, the ELBO of a point set  $X$  can be finally written as

$$\begin{aligned} \mathcal{L}(X; \phi, \psi, \theta) &= \mathbb{E}_{Q_\phi(z|x)} [\log P_\psi(z) + \log P_\theta(X|z)] + H[Q_\phi(z|X)] \\ &= \mathbb{E}_{Q_\phi(z|x)} [\log P(F_\psi^{-1}(z)) - \int_{t_0}^{t_1} \text{Tr} \left( \frac{\partial f_\psi}{\partial w(t)} \right) dt \\ &\quad + \sum_{x \in X} (\log P(G_\theta^{-1}(x; z)) - \int_{t_0}^{t_1} \text{Tr} \left( \frac{\partial g_\theta}{\partial y(t)} \right) dt)] \\ &\quad + H[Q_\phi(z|X)]. \end{aligned} \quad (8)$$

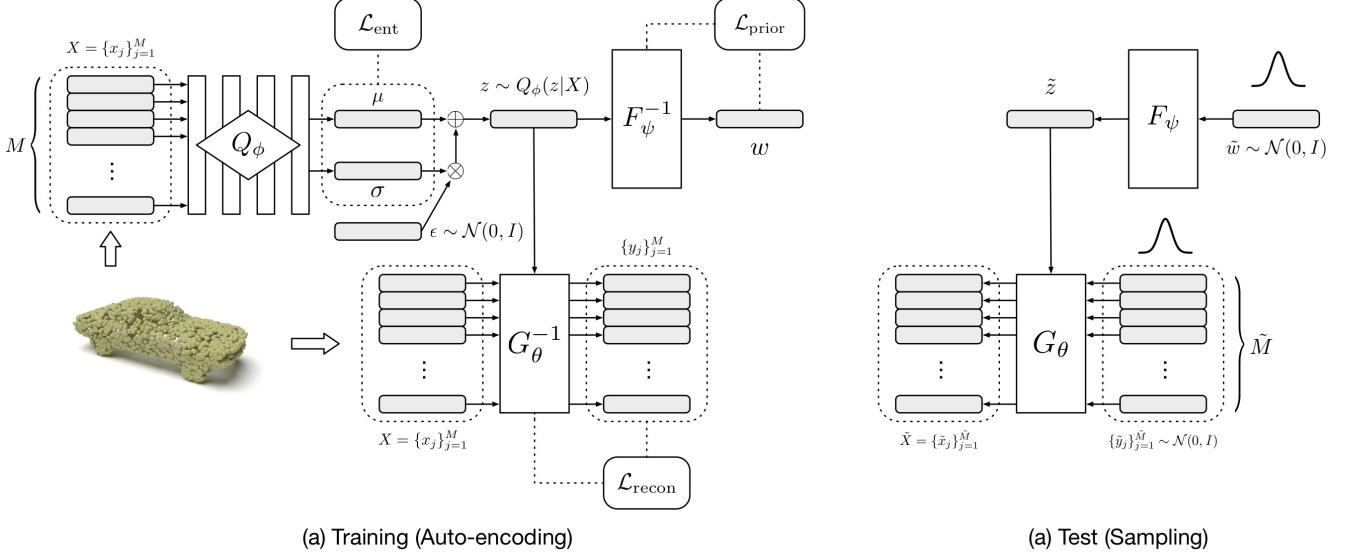


Figure 2: Model architecture. (a) At training time, the encoder  $Q_\phi$  infers a posterior over shape representations given an input point cloud  $X$ , and samples a shape representation  $z$  from it. We then compute the probability of  $z$  in the prior distribution ( $\mathcal{L}_{\text{prior}}$ ) through a inverse CNF  $F_\psi^{-1}$ , and compute the reconstruction likelihood of  $X$  ( $\mathcal{L}_{\text{recon}}$ ) through another inverse CNF  $G_\theta^{-1}$  conditioned on  $z$ . The model is trained end-to-end to maximize the evidence lower bound (ELBO), which is the sum of  $\mathcal{L}_{\text{prior}}$ ,  $\mathcal{L}_{\text{recon}}$ , and  $\mathcal{L}_{\text{ent}}$  (the entropy of the posterior  $Q_\phi(z|X)$ ). (b) At test time, we sample a shape representation  $\tilde{z}$  by sampling  $\tilde{w}$  from a Gaussian prior and transforming it with  $F_\psi$ . To sample points from the shape represented by  $\tilde{z}$ , we first sample points from the 3-D Gaussian prior and then move them according to the CNF parameterized by  $\tilde{z}$ .

Our model is trained end-to-end by maximizing the ELBO of all point sets in the dataset

$$\phi^*, \psi^*, \theta^* = \arg \max_{\phi, \psi, \theta} \sum_{X \in \mathcal{X}} \mathcal{L}(X; \phi, \psi, \theta). \quad (9)$$

We can interpret this objective as the sum of three parts:

1. **Prior:**  $\mathcal{L}_{\text{prior}}(X; \psi, \phi) \triangleq \mathbb{E}_{Q_\phi(z|x)}[\log P_\psi(z)]$  encourages the encoded shape representation to have a high probability under the prior, which is modeled by a CNF as described in Section 5.2. We use the reparameterization trick [27] to enable a differentiable Monte Carlo estimate of the expectation:

$$\mathbb{E}_{Q_\phi(z|x)}[\log P_\psi(z)] \approx \frac{1}{L} \sum_{l=1}^L \log P_\psi(\mu + \epsilon_l \odot \sigma),$$

where  $\mu$  and  $\sigma$  are mean and standard deviation of the isotropic Gaussian posterior  $Q_\phi(z|x)$  and  $L$  is simply set to 1.  $\epsilon_l$  is sampled from the standard Gaussian distribution  $\mathcal{N}(0, I)$ .

2. **Reconstruction likelihood:**  $\mathcal{L}_{\text{recon}}(X; \theta, \phi) \triangleq \mathbb{E}_{Q_\phi(z|x)}[\log P_\theta(X|z)]$  is the reconstruction log-likelihood of the input point set, computed as described in Section 5.1. The expectation is also estimated using Monte Carlo sampling.

3. **Posterior Entropy:**  $\mathcal{L}_{\text{ent}}(X; \phi) \triangleq H[Q_\phi(z|X)]$  is the entropy of the approximated posterior:

$$H[Q_\phi(z|X)] = \frac{d}{2}(1 + \ln(2\pi)) + \sum_{i=1}^d \ln \sigma_i.$$

All the training details (e.g., hyper-parameters, model architectures) are included in Section B of the appendix.

#### 5.4. Sampling

To sample a shape representation, we first draw  $\tilde{w} \sim \mathcal{N}(0, I)$  then pass it through  $F_\psi$  to get  $\tilde{z} = F_\psi(\tilde{w})$ . To generate a point given a shape representation  $\tilde{z}$ , we first sample a point  $\tilde{y} \in \mathbb{R}^3$  from  $\mathcal{N}(0, I)$ , then pass  $\tilde{y}$  through  $G_\theta$  conditioned on  $\tilde{z}$  to produce a point on the shape :  $\tilde{x} = G_\theta(\tilde{w}; \tilde{z})$ . To sample a point cloud with size  $\tilde{M}$ , we simply repeat it for  $\tilde{M}$  times. Combining these two steps allows us to sample a point cloud with  $\tilde{M}$  points from our model:

$$\tilde{X} = \{G_\theta(\tilde{y}_j; F_\psi(\tilde{w}))\}_{1 \leq j \leq \tilde{M}}, \tilde{w} \sim \mathcal{N}(0, I), \forall j, \tilde{y}_j \sim \mathcal{N}(0, I).$$

## 6. Experiments

In this section, we first introduce existing metrics for point cloud generation, discuss their limitations, and introduce a new metric that overcomes these limitations. We then compare the proposed method with previous state-of-the-art generative models of point clouds, using both previous metrics and the proposed one. We additionally evaluate

the reconstruction and representation learning ability of the auto-encoder part of our model.

## 6.1. Evaluation metrics

Following prior work, we use Chamfer distance (CD) and earth mover’s distance (EMD) to measure the similarity between point clouds. Formally, they are defined as follows:

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

$$\text{EMD}(X, Y) = \min_{\phi: X \rightarrow Y} \sum_{x \in X} \|x - \phi(x)\|_2,$$

where  $X$  and  $Y$  are two point clouds with the same number of points and  $\phi$  is a bijection between them. Note that most previous methods use either CD or EMD in their training objectives, which tend to be favored if evaluated under the same metric. Our method, however, do not use CD or EMD during training.

Let  $S_g$  be the set of generated point clouds and  $S_r$  be the set of reference point clouds with  $|S_r| = |S_g|$ . To evaluate generative models, we first consider the three metrics introduced by Achlioptas *et al.* [1]:

- **Jensen-Shannon Divergence (JSD)** are computed between the marginal point distributions:

$$\text{JSD}(P_g, P_r) = \frac{1}{2} D_{KL}(P_r || P_g) + \frac{1}{2} D_{KL}(P_g || P_r),$$

where  $P_r$  and  $P_g$  are marginal distributions of points in the reference and generated sets, approximated by discretizing the space into  $28^3$  voxels and assigning each point to one of them. However, it only considers the marginal point distributions but not the distribution of individual shapes. A model that always outputs the “average shape” can obtain a perfect JSD score without learning any meaningful shape distributions.

- **Coverage (COV)** measures the fraction of point clouds in the reference set that are matched to at least one point cloud in the generated set. For each point cloud in the generated set, its nearest neighbor in the reference set is marked as a match:

$$\text{COV}(S_g, S_r) = \frac{|\{\arg \min_{Y \in S_r} D(X, Y) | X \in S_g\}|}{|S_r|},$$

where  $D(\cdot, \cdot)$  can be either CD or EMD. While coverage is able to detect mode collapse, it does not evaluate the quality of generated point clouds. In fact, it is possible to achieve a perfect coverage score even if the distances between generated and reference point clouds are arbitrarily large.

- **Minimum matching distance (MMD)** is proposed to complement coverage as a metric that measures quality. For each point cloud in the reference set, the distance to its nearest neighbor in the generated set is computed and averaged:

$$\text{MMD}(S_g, S_r) = \frac{1}{|S_r|} \sum_{Y \in S_r} \min_{X \in S_g} D(X, Y),$$

where  $D(\cdot, \cdot)$  can be either CD or EMD. However, MMD is actually very insensitive to low-quality point clouds in  $S_g$ , since they are unlikely to be match to real point clouds in  $S_r$ . In the extreme case, one can imagine that  $S_g$  consists of mostly very low-quality point clouds with one additional point cloud in each mode of  $S_r$ , yet having a reasonably good MMD score.

As discussed above, all existing metrics have their limitations. As will be shown later, we also empirically find all these metrics sometimes give generated point clouds even better scores than real point clouds, further casting doubt on whether they can ensure a fair model comparison. We therefore introduce another metric that we believe is better suited for evaluating generative models of point clouds:

- **1-nearest neighbor accuracy (1-NNA)** is proposed by Lopez-Paz and Oquab [32] for two-sample tests, assessing whether two distributions are identical. It has also been explored as a metric for evaluating GANs [50]. Let  $S_{-X} = S_r \cup S_g - \{X\}$  and  $N_X$  be the nearest neighbor of  $X$  in  $S_{-X}$ . 1-NNA is the leave-one-out accuracy of the 1-NN classifier:

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

where  $\mathbb{I}[\cdot]$  is the indicator function. For each sample, the 1-NN classifier classifies it as coming from  $S_r$  or  $S_g$  according to the label of its nearest sample. If  $S_g$  and  $S_r$  are sampled from the same distribution, the accuracy of such a classifier should converge to 50% given a sufficient number of samples. The closer the accuracy is to 50%, the more similar  $S_g$  and  $S_r$  are, and therefore the better the model is at learning the target distribution. In our setting, the nearest neighbor can be computed using either CD or EMD. Unlike JSD, 1-NNA considers the similarity between shape distributions rather than between marginal point distributions. Unlike COV and MMD, 1-NNA directly measures distributional similarity and takes both diversity and quality into account.

## 6.2. Generation

We compare our method with three existing generative models for point clouds: raw-GAN [1], latent-GAN [1], and

Table 1: Generation results.  $\uparrow$ : the higher the better,  $\downarrow$ : the lower the better. The best scores are highlighted in bold. Scores of the real shapes that are worse than some of the generated shapes are marked in gray. MMD-CD scores are multiplied by  $10^3$ ; MMD-EMD scores are multiplied by  $10^2$ ; JSDs are multiplied by  $10^2$ .

| Category     | Model            | # Parameters (M) |             | JSD ( $\downarrow$ ) | MMD ( $\downarrow$ ) |             | COV (%), $\uparrow$ |              | 1-NNA (%), $\downarrow$ |              |
|--------------|------------------|------------------|-------------|----------------------|----------------------|-------------|---------------------|--------------|-------------------------|--------------|
|              |                  | Full             | Gen         |                      | CD                   | EMD         | CD                  | EMD          | CD                      | EMD          |
| Airplane     | r-GAN            | 7.22             | 6.91        | 7.44                 | 0.261                | 5.47        | 42.72               | 18.02        | 93.58                   | 99.51        |
|              | l-GAN (CD)       | 1.97             | 1.71        | 4.62                 | 0.239                | 4.27        | 43.21               | 21.23        | 86.30                   | 97.28        |
|              | l-GAN (EMD)      | 1.97             | 1.71        | <b>3.61</b>          | 0.269                | 3.29        | <b>47.90</b>        | <b>50.62</b> | 87.65                   | 85.68        |
|              | PC-GAN           | 9.14             | 1.52        | 4.63                 | 0.287                | 3.57        | 36.46               | 40.94        | 94.35                   | 92.32        |
|              | PointFlow (ours) | <b>1.61</b>      | <b>1.06</b> | 4.92                 | <b>0.217</b>         | <b>3.24</b> | 46.91               | 48.40        | <b>75.68</b>            | <b>75.06</b> |
| Training set |                  | -                | -           | 6.61                 | 0.226                | 3.08        | 42.72               | 49.14        | 70.62                   | 67.53        |
| Chair        | r-GAN            | 7.22             | 6.91        | 11.5                 | 2.57                 | 12.8        | 33.99               | 9.97         | 71.75                   | 99.47        |
|              | l-GAN (CD)       | 1.97             | 1.71        | <b>4.59</b>          | 2.46                 | 8.91        | 41.39               | 25.68        | 64.43                   | 85.27        |
|              | l-GAN (EMD)      | 1.97             | 1.71        | 2.27                 | 2.61                 | <b>7.85</b> | 40.79               | 41.69        | 64.73                   | 65.56        |
|              | PC-GAN           | 9.14             | 1.52        | 3.90                 | 2.75                 | 8.20        | 36.50               | 38.98        | 76.03                   | 78.37        |
|              | PointFlow (ours) | <b>1.61</b>      | <b>1.06</b> | <b>1.74</b>          | <b>2.42</b>          | 7.87        | <b>46.83</b>        | <b>46.98</b> | <b>60.88</b>            | <b>59.89</b> |
| Training set |                  | -                | -           | 1.50                 | 1.92                 | 7.38        | 57.25               | 55.44        | 59.67                   | 58.46        |
| Car          | r-GAN            | 7.22             | 6.91        | 12.8                 | 1.27                 | 8.74        | 15.06               | 9.38         | 97.87                   | 99.86        |
|              | l-GAN (CD)       | 1.97             | 1.71        | <b>4.43</b>          | 1.55                 | 6.25        | 38.64               | 18.47        | 63.07                   | 88.07        |
|              | l-GAN (EMD)      | 1.97             | 1.71        | 2.21                 | 1.48                 | 5.43        | 39.20               | 39.77        | 69.74                   | 68.32        |
|              | PC-GAN           | 9.14             | 1.52        | <b>5.85</b>          | 1.12                 | 5.83        | 23.56               | 30.29        | 92.19                   | 90.87        |
|              | PointFlow (ours) | <b>1.61</b>      | <b>1.06</b> | <b>0.87</b>          | <b>0.91</b>          | <b>5.22</b> | <b>44.03</b>        | <b>46.59</b> | <b>60.65</b>            | <b>62.36</b> |
| Training set |                  | -                | -           | 0.86                 | 1.03                 | 5.33        | 48.30               | 51.42        | 57.39                   | 53.27        |

PC-GAN [30], using their official implementations that are either publicly available or obtained by contacting the authors. We train each model using point clouds from one of the three categories in ShapeNet [3] dataset: *airplane*, *chair*, and *car*. The point clouds are obtained by sampling points uniformly from the mesh surface. All points in each category are normalized to be zero-mean per axis and unit-variance globally. Following prior convention [1], we use 2048 points for each shape during both training and testing, although our model is able to sample an arbitrary number of points. We additionally report the performance of point clouds sampled from the training set, which is considered as an upper bound since they are from the target distribution.

In Table 1, we report the performance of different models, as well as their number of parameters in total (full) or in the generative pathways (gen). We first note that all the previous metrics (JSD, MMD, and COV) sometimes assign a better score to point clouds generated by models than those from the training set (marked in gray). The 1-NNA metric does not seem to have this problem and always gives a better score to shapes from the training set. Our model outperforms all baselines across all three categories according to 1-NNA and also obtains the best score in most cases as evaluated by other metrics. Besides, our model has the fewest parameters among compared models. In Section C of the appendix, we perform additional ablation studies to show

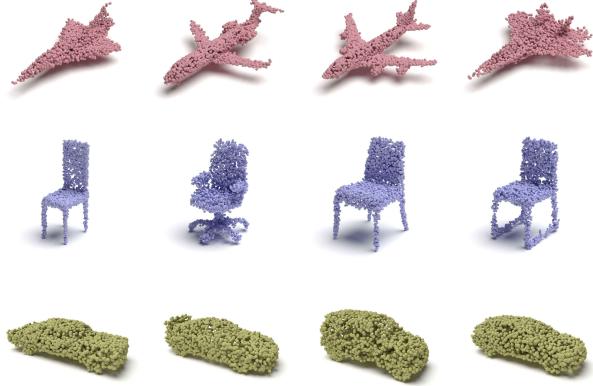


Figure 3: Examples of point clouds generated by our model. From top to bottom: airplane, chair, and car.

the effectiveness of different components of our model. Figure 3 shows some examples of novel point clouds generated by our model. Figure 4 shows examples of point clouds reconstructed from given inputs.

### 6.3. Auto encoding

We further quantitatively compare the reconstruction ability of our flow-based auto-encoder with l-GAN [1] and

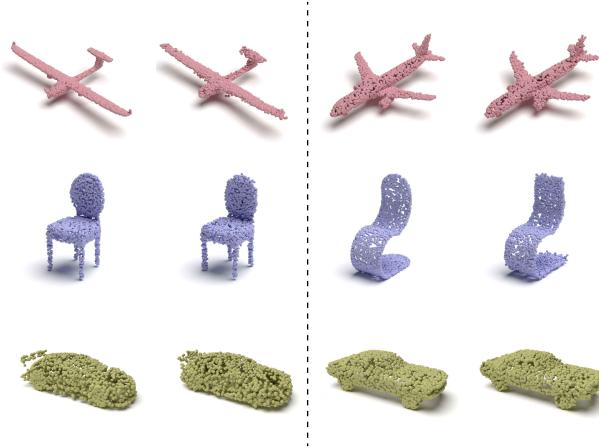


Figure 4: Examples of point clouds reconstructed from inputs. From top to bottom: airplane, chair, and car. On each side of the figure we show the input point cloud on the left and the reconstructed point cloud on the right.

Table 2: Unsupervised feature learning. Models are first trained on ShapeNet to learn shape representations, which are then evaluated on ModelNet40 (MN40) and ModelNet10 (MN10) by comparing the accuracy of off-the-shelf SVMs trained using the learned representations.

| Method                       | MN40 (%)    | MN10 (%)    |
|------------------------------|-------------|-------------|
| SPH [24]                     | 68.2        | 79.8        |
| LFD [4]                      | 75.5        | 79.9        |
| T-L Network [14]             | 74.4        | -           |
| VConv-DAE [42]               | 75.5        | 80.5        |
| 3D-GAN [48]                  | 83.3        | 91.0        |
| I-GAN (EMD) [1]              | 84.0        | <b>95.4</b> |
| I-GAN (CD) [1]               | 84.5        | <b>95.4</b> |
| PointGrow [44]               | 85.7        | -           |
| MRTNet-VAE [13]              | 86.4        | -           |
| FoldingNet [51]              | <b>88.4</b> | 94.4        |
| I-GAN (CD) [1] <sup>†</sup>  | <b>87.0</b> | 92.8        |
| I-GAN (EMD) [1] <sup>†</sup> | 86.7        | 92.2        |
| PointFlow (ours)             | 86.8        | <b>93.7</b> |

<sup>†</sup> We run the official code of I-GAN on our pre-processed dataset using the same encoder architecture as our model.

AtlasNet [17]. Following the setting of AtlasNet, the state of the art in this task, we train our auto-encoder on all shapes in the ShapeNet dataset. The auto-encoder is trained with the reconstruction likelihood objective  $\mathcal{L}_{\text{recon}}$  only. At test time, we sample 4096 points per shape and split them into an input set and a reference set, each consisting of 2048 points. We then compute the distance (CD or EMD) be-

Table 3: Auto-encoding performance evaluated by CD and EMD. AtlasNet is trained with CD and I-GAN is trained on CD or EMD. Our method is not trained on CD or EMD. CD scores are multiplied by  $10^4$ ; EMD scores are multiplied by  $10^2$ .

| Model            | # Parameters (M) | CD          | EMD         |
|------------------|------------------|-------------|-------------|
| I-GAN (CD) [1]   | 1.77             | 7.12        | 7.95        |
| I-GAN (EMD) [1]  | 1.77             | 8.85        | 5.26        |
| AtlasNet [17]    | 44.9             | <b>5.13</b> | 5.97        |
| PointFlow (ours) | <b>1.30</b>      | 7.54        | <b>5.18</b> |

tween the reconstructed input set and the reference set <sup>1</sup>. Although our model is not directly trained with EMD, it obtains the best EMD score, even higher than I-GAN trained with EMD and AtlasNet which has more than 40 times more parameters.

#### 6.4. Unsupervised representation learning

We finally evaluate the representation learning ability of our auto-encoders. Specifically, we extract the latent representations of our auto-encoder trained in the full ShapeNet dataset and train a linear SVM classifier on top of it on ModelNet10 or ModelNet40 [49]. Only for this task, we normalize each individual point cloud to be zero-mean per axis and unit-variance globally, following prior works [55, 1]. We also apply random rotations along the gravity axis when training the auto-encoder.

A problem with this task is that different authors have been using different encoder architectures with a different number of parameters, making it hard to perform an apple-to-apple comparison. In addition, different authors may use different pre-processing protocols (as also noted by Yang *et al.* [51]), which could also affect the numbers.

In Table 2, we still show the numbers reported by previous papers, but also include a comparison with I-GAN [1] trained using the same encoder architecture and the exact same data as our model. On ModelNet10, the accuracy of our model is 1.5% and 0.9% higher than I-GAN (EMD) and I-GAN (CD) respectively. On ModelNet40, the performance of the three models is very close.

## 7. Conclusion and future works

In this paper, we propose PointFlow, a generative model for point clouds consisting of two levels of continuous normalizing flows trained with variational inference. Future work includes applications to other tasks such as point cloud reconstruction from a single image.

<sup>1</sup>We use a separate reference set because we expect the auto-encoder to learn the point distribution. Exactly reproducing the input points is acceptable behavior, but should not be given a higher score than randomly sampling points from the underlying point distribution.

## 8. Acknowledgement

This work was supported in part by a research gift from Magic Leap. Xun Huang was supported by NVIDIA Graduate Fellowship.

## References

- [1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. In *ICML*, 2018. [2](#), [3](#), [4](#), [6](#), [7](#), [8](#), [11](#)
- [2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017. [2](#)
- [3] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. [7](#)
- [4] D.-Y. Chen, X.-P. Tian, E. Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3d model retrieval. *Comput. Graph. Forum*, 22:223–232, 2003. [8](#)
- [5] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018. [2](#), [3](#), [11](#)
- [6] X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel. Variational lossy autoencoder. In *ICLR*, 2016. [3](#), [4](#)
- [7] I. Danihelka, B. Lakshminarayanan, B. Uria, D. Wierstra, and P. Dayan. Comparison of maximum likelihood and gan-based training of real nvp. *arXiv preprint arXiv:1705.05263*, 2017. [3](#)
- [8] L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear independent components estimation. *CoRR*, abs/1410.8516, 2014. [2](#)
- [9] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. In *ICLR*, 2017. [2](#)
- [10] O. Dovrat, I. Lang, and S. Avidan. Learning to sample. *arXiv preprint arXiv:1812.01659*, 2018. [2](#)
- [11] H. A. Edwards and A. J. Storkey. Towards a neural statistician. In *ICLR*, 2017. [3](#), [11](#), [12](#)
- [12] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, 2017. [2](#)
- [13] M. Gadelha, R. Wang, and S. Maji. Multiresolution tree networks for 3d point cloud processing. In *ECCV*, 2018. [2](#), [8](#)
- [14] R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta. Learning a predictable and generative vector representation for objects. In *ECCV*, 2016. [8](#)
- [15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NeurIPS*, 2014. [2](#)
- [16] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *ICLR*, 2019. [2](#), [3](#), [11](#), [12](#)
- [17] T. Groueix, M. Fisher, V. G. Kim, B. Russell, and M. Aubry. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *CVPR*, 2018. [2](#), [8](#)
- [18] A. Grover, M. Dhar, and S. Ermon. Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *AAAI*, 2018. [3](#)
- [19] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *NeurIPS*, 2017. [2](#), [11](#)
- [20] C.-W. Huang, D. Krueger, A. Lacoste, and A. C. Courville. Neural autoregressive flows. In *ICML*, 2018. [3](#)
- [21] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. [11](#)
- [22] L. Jiang, S. Shi, X. Qi, and J. Jia. Gal: Geometric adversarial loss for single-view 3d-object reconstruction. In *ECCV*, 2018. [2](#)
- [23] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019. [2](#)
- [24] M. M. Kazhdan, T. A. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *Symposium on Geometry Processing*, 2003. [8](#)
- [25] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, 2018. [2](#)
- [26] D. P. Kingma, T. Salimans, and M. Welling. Improving variational inference with inverse autoregressive flow. In *NeurIPS*, 2016. [3](#)
- [27] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014. [2](#), [4](#), [5](#)
- [28] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. Kingma. Videoflow: A flow-based generative model for video. *arXiv preprint arXiv:1903.01434*, 2019. [3](#)
- [29] A. Kurenkov, J. Ji, A. Garg, V. Mehta, J. Gwak, C. B. Choy, and S. Savarese. Deformnet: Free-form deformation network for 3d shape reconstruction from a single image. In *WACV*, 2018. [2](#)
- [30] C.-L. Li, M. Zaheer, Y. Zhang, B. Poczos, and R. Salakhutdinov. Point cloud gan. *arXiv preprint arXiv:1810.05795*, 2018. [2](#), [3](#), [7](#)
- [31] K. Li, T. Pham, H. Zhan, and I. D. Reid. Efficient dense point cloud object reconstruction using deformation vector fields. In *ECCV*, 2018. [2](#)
- [32] D. Lopez-Paz and M. Oquab. Revisiting classifier two-sample tests. In *ICLR*, 2017. [6](#)
- [33] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. [12](#)
- [34] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015. [2](#)
- [35] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016. [2](#)
- [36] G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In *NeurIPS*, 2017. [3](#)

- [37] R. Prenger, R. Valle, and B. Catanzaro. Waveglow: A flow-based generative network for speech synthesis. *CoRR*, abs/1811.00002, 2018. 3
- [38] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 2, 11
- [39] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 2
- [40] D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. In *ICML*, 2015. 2, 3
- [41] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014. 2, 4
- [42] A. Sharma, O. Grau, and M. Fritz. Vconv-dae: Deep volumetric shape learning without object labels. In *ECCV Workshops*, 2016. 8
- [43] M. Shoef, S. Fogel, and D. Cohen-Or. Pointwise: An unsupervised point-wise feature learning network. *arXiv preprint arXiv:1901.04544*, 2019. 2
- [44] Y. Sun, Y. Wang, Z. Liu, J. E. Siegel, and S. E. Sarma. Point-grow: Autoregressively learned point cloud generation with self-attention. *arXiv preprint arXiv:1810.05591*, 2018. 2, 8
- [45] V. Usenko, J. Engel, J. Stückler, and D. Cremers. Reconstructing street-scenes in real-time from a driving car. In *3DV*, 2015. 2
- [46] R. van den Berg, L. Hasenclever, J. M. Tomczak, and M. Welling. Sylvester normalizing flows for variational inference. In *UAI*, 2018. 3
- [47] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. In *NeurIPS*, 2016. 2
- [48] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In *NeurIPS*, 2016. 8
- [49] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 8
- [50] Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, and K. Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *arXiv preprint arXiv:1806.07755*, 2018. 6
- [51] Y. Yang, C. Feng, Y. Shen, and D. Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *CVPR*, 2018. 2, 8
- [52] W. Yifan, S. Wu, H. Huang, D. Cohen-Or, and O. Sorkine-Hornung. Patch-based progressive 3d point set upsampling. *arXiv preprint arXiv:1811.11286*, 2018. 2
- [53] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Ec-net: an edge-aware point set consolidation network. In *ECCV*, 2018. 2
- [54] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Pu-net: Point cloud upsampling network. In *CVPR*, 2018. 2
- [55] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *NeurIPS*, 2017. 2, 8
- [56] M. Zamorski, M. Zieba, R. Nowak, W. Stokowiec, and T. Trzciński. Adversarial autoencoders for generating 3d point clouds. *arXiv preprint arXiv:1811.07605*, 2018. 2

## A. Overview

In the appendix, we first describe the detailed hyper-parameters and model architectures for our experiments in Section B. We then compare our model with additional baselines to understand the effect of different model components in Section C. Limitations and typical failure cases are discussed in Section D. Finally, additional visualizations of latent space t-SNE, interpolations and flow transformations are presented in Section E, Section F, and Section G respectively.

## B. Training details

In this section, we provide details about our network architectures and training hyper-parameters. We will release the code to reproduce our experiments.

**Encoder.** The architecture of our encoder follows that of Achlioptas *et al.* [1]. Specifically, we first use 1D Convolution with filter size 128, 128, 256, and 512 to process each point independently and then use max pooling to create a 512-dimension feature as done in PointNet [38]. Such a feature is invariant to the permutation of points due to the max-pooling. Finally, we apply a three-layer MLP with 256 and 128 hidden dimensions to convert the permutation invariant feature to a  $D_z$ -dimension one. For the unsupervised representation learning experiment, we set  $D_z = 512$  following convention. For all other experiments,  $D_z$  is set to 128.

**CNF prior.** The CNF prior models the distribution  $P_\psi(z)$ . We follow FFJORD [16]’s released code to use three concatsquash layers to model the dynamics  $f_\psi$ . A concatsquash layer is defined as:

$$\text{CS}(x, t) = (W_x x + b_x) \sigma(W_t t + b_t) + (W_b t + b_b), \quad (10)$$

where  $W_x$ ,  $b_x$ ,  $W_t$ ,  $b_t$ ,  $W_b$ , and  $b_b$  are all trainable parameters and  $\sigma(\cdot)$  is the sigmoid function.  $f_\psi$  uses three concatsquash layers with a hidden dimension 256. Tanh is used as the non-linearity between layers.

We use a Moving Batch Normalization layer to learn the scale of each dimension before and after the CNF, following FFJORD’s released code [16]. Specifically, Moving Batch Normalization is defined as

$$\text{MBN}(x) = \frac{x - \mu}{\sigma} \gamma + \beta, \quad (11)$$

where  $\gamma$  and  $\beta$  are trainable parameters. Different from batch normalization proposed by Ioffe and Szegedy [21],  $\mu$  and  $\sigma$  are running averages of the batch mean and standard deviation. MovingBatchNorm is invertible :  $\text{MBN}^{-1}(y) = \frac{y - \beta}{\gamma} \sigma + \mu$ . Its log determinant is given as:

$$\log \det \left| \frac{\partial \text{MBN}(x)}{\partial x} \right| = \sum_i \log |\gamma_i| - \log |\sigma_i|. \quad (12)$$

**CNF decoder.** The CNF decoder models the reconstruction likelihood  $P_\theta(X|z)$ . We extend the concatsquash layer to condition on the latent vector  $z$ :

$$\begin{aligned} \text{CCS}(x, z, t) &= (W_x x + b_x) \sigma(W_{tt} t + W_{tz} z + b_t) \\ &\quad + (W_{bt} t + W_{bz} z + b_b), \end{aligned} \quad (13)$$

where  $W_x, W_{tt}, W_{tz}, W_{bt}, W_{bz}, b_t, b_b$  are all learnable parameters. The CNF decoder uses four conditional concatsquash layers with a hidden dimension 512 to model the dynamic  $g_\theta$ . The non-linearity between layers is Tanh. Similar to the CNF prior model, we also add a Moving Batch Normalization layer before and after the CNF. In this case, all 3D points (from different shapes) from a batch are used to compute the batch statistics.

**Hyper-parameters.** We use an Adam optimizer with an initial learning rate 0.002,  $\beta_1 = 0.9$ , and  $\beta_2 = 0.999$ . The learning rate decays linearly to 0 starting at the 2000<sup>th</sup> epoch and ends at the 4000<sup>th</sup> epoch. We do not use any weight decay. We also learn the integration time  $t_1$  during training by back-propagation [5].

## C. Additional comparisons

In this section, we compare our model to more baselines to show the effectiveness of the model design. The first baseline is Neural Statistician (NS) [11], a state-of-the-art generative model for sets. We modify its official code for generating 2D spatial coordinates of MNIST digits to make it work with 3D point cloud coordinates. We use the same encoder architecture as our model, and use the VAE decoder provided by authors with the input dimension changed from 2 to 3. It differs from our model mainly in 1) using VAEs instead of CNFs to model the reconstruction likelihood, and 2) using a simple Gaussian prior instead of a flow-based one. The second baseline is VAECNF, where we use the CNF to model the reconstruction likelihood but not prior. Specifically, the VAECNF optimizes ELBO in the following form:

$$\begin{aligned} \mathcal{L}(X; \phi, \theta) &= \sum_{x \in X} \left( \log P(G_\theta^{-1}(x; z)) - \int_{t_0}^{t_1} \text{Tr} \left( \frac{\partial g_\theta}{\partial y(t)} \right) dt \right) \\ &\quad + D_{KL}(Q_\phi(z|X) || P(z)), \end{aligned} \quad (14)$$

where  $P(z)$  is a standard Gaussian  $\mathcal{N}(0, I)$  and  $D_{KL}$  is the KL-divergence. As another baseline, we follow 1-GAN [1] to train a WGAN [19] in the latent space of our pretrained auto-encoder. Both the discriminator and the generator are MLP with batch normalization between layers. The generator has three layers with hidden dimensions 256. The discriminator has three layers with hidden dimensions 512.

The results are presented in Table 4. Neural Statistician [11] is able to learn the marginal point distribution but fails to learn the correct shape distribution, as it obtains the best marginal JSD but very poor scores according to metrics that measure similarities between shape distributions. Also,

Table 4: Ablation studies.  $\uparrow$ : the higher the better,  $\downarrow$ : the lower the better. The best scores are highlighted in bold. MMD-CD scores are multiplied by  $10^3$ ; MMD-EMD scores are multiplied by  $10^2$ ; JSDs are multiplied by  $10^2$ .

| Category | Model            | # Parameters (M) |             | JSD ( $\downarrow$ ) | MMD ( $\downarrow$ ) |             | COV (%), $\uparrow$ |              | 1-NNA (%), $\downarrow$ |              |
|----------|------------------|------------------|-------------|----------------------|----------------------|-------------|---------------------|--------------|-------------------------|--------------|
|          |                  | Full             | Gen         |                      | CD                   | EMD         | CD                  | EMD          | CD                      | EMD          |
| Airplane | NS [11]          | 2.29             | 1.00        | <b>1.74</b>          | 0.655                | 4.51        | 7.81                | 4.51         | 99.61                   | 99.61        |
|          | VAECNF           | <b>1.47</b>      | <b>0.92</b> | 6.30                 | 0.261                | 3.35        | 41.98               | 46.17        | 88.64                   | 82.72        |
|          | WGANGCNF         | 1.75             | 1.06        | 4.29                 | 0.254                | <b>3.23</b> | 42.47               | <b>48.40</b> | 75.80                   | 75.68        |
|          | PointFlow (ours) | 1.61             | 1.06        | 4.92                 | <b>0.217</b>         | 3.24        | <b>46.91</b>        | <b>48.40</b> | <b>75.68</b>            | <b>75.06</b> |
|          | Training set     | -                | -           | 6.61                 | 0.226                | 3.08        | 42.72               | 49.14        | 70.62                   | 67.53        |

using a flexible prior parameterized by a CNF (PointFlow) is better than using a simple Gaussian prior (VAECNF) or a prior learned with a latent GAN (WGANGCNF) that requires two-stage training.

## D. Limitation and failure cases

In this section, we discuss the limitation of our model and present visualizations of difficult cases where our model fails. As mentioned in FFJORD [16], each integration requires evaluating the neural networks modeling the dynamics multiple times. The number of function evaluations tends to increase as the training proceeds since the dynamic becomes more complex and more function evaluations are needed to achieve the same numerical precision. This issue limits our model size and makes the convergence slow. Grathwohl *et al.* indicate that using regularization such as weight decay could alleviate such an issue, but we empirically find that using regularization tends to hurt performance. Future advances in invertible models like CNF might help improve this issue. Typical failure case appears when reconstructing or generating the rare shape or shapes with many thin structures as presented in Figure 5.

## E. Latent space visualizations

We provide visualization of the sampled latent vectors  $z \in \mathbb{R}^{128}$  in Figure 6. We sample 1000 latent vectors and run t-SNE [33] to visualize these latent vectors in 2D. Shapes with similar styles are close in the latent space.

## F. Interpolation

In this section, we present interpolation between two different shapes using our model. For two shapes  $X_1$  and  $X_2$ , we first compute the mean of the posterior distribution using  $Q_\theta(z|X)$ . Let  $\mu_1$  and  $\mu_2$  be the means of the posterior distribution for  $X_1$  and  $X_2$  respectively. We use  $\mu_1$  and  $\mu_2$  as the latent representation for these two shapes. We then use the inverse prior flow  $F_\psi^{-1}$  to transform  $\mu_1$  and  $\mu_2$  back to the prior space. Let  $w_1 = F_\psi^{-1}(\mu_1)$  and  $w_2 = F_\psi^{-1}(\mu_2)$  be the corresponding vectors for  $\mu_1$  and  $\mu_2$  in the prior space.

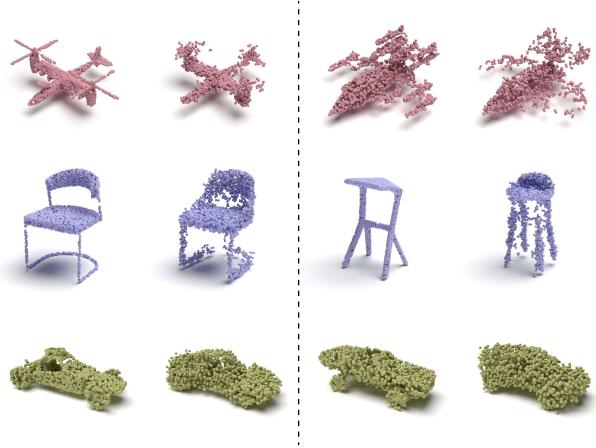


Figure 5: Difficult cases for our model. Rare shapes or shapes that contain many thin structures are usually hard to reconstruct in high quality.

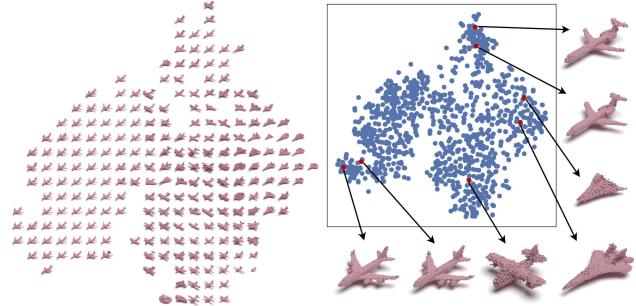


Figure 6: Visualization of latent space.

We use spherical interpolation between  $w_1$  and  $w_2$  to retrieve a series of vectors  $w_i$ . For each  $w_i$ , we use the CNF prior  $F_\psi$  and the CNF decoder  $G_\theta$  to generate the corresponding shape  $X_i$ . Figure 7 contains examples of the interpolation.

## G. More flow transformation

Figure 8 presents more examples of flow transformations from the Gaussian prior to different shapes.

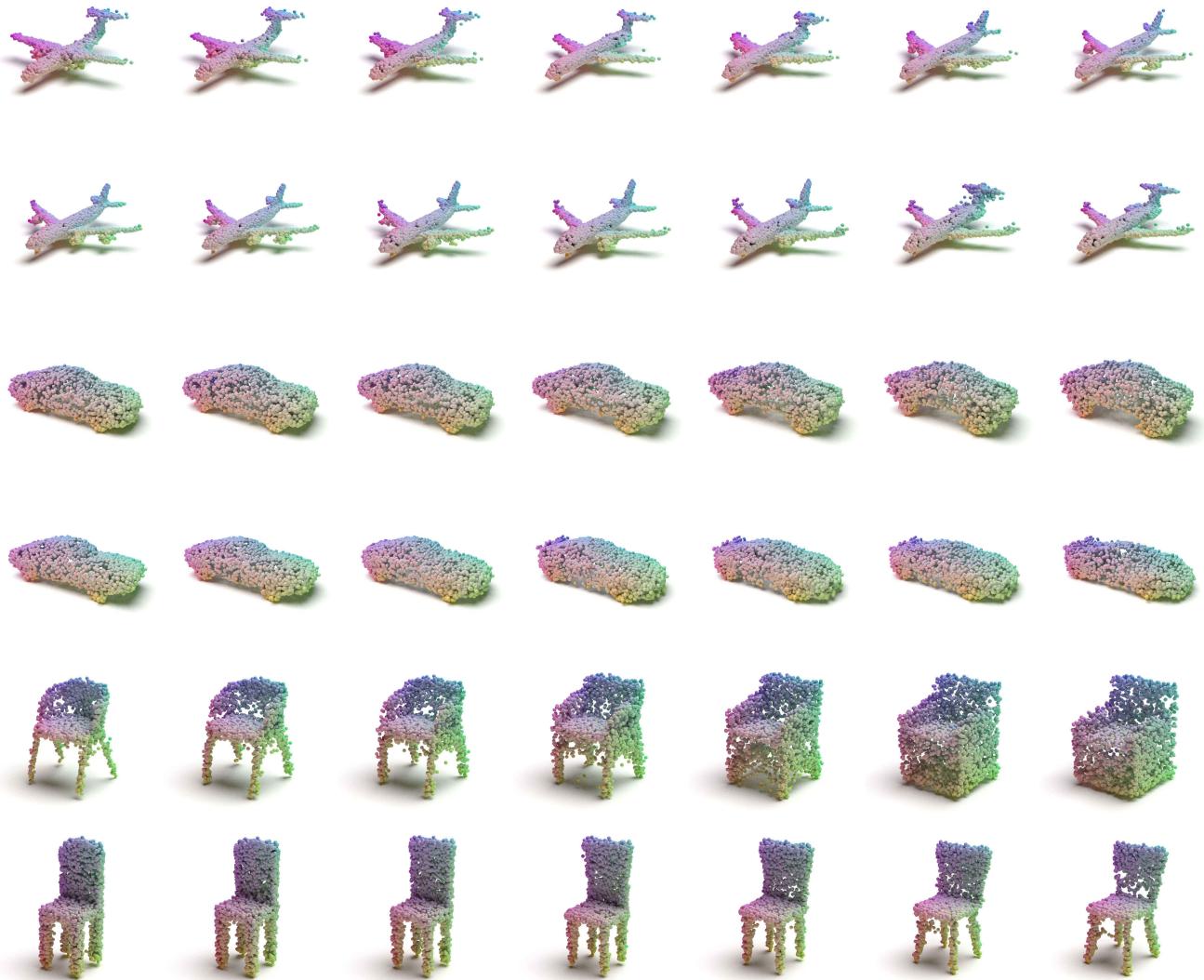


Figure 7: Feature space interpolation. The left-most and the right-most shapes are sampled from scratch. The shapes in between are generated by interpolating the two shapes in the prior space.



Figure 8: Additional visualizations on the process of transforming prior to point cloud.