S2Net: Stochastic Sequential Pointcloud Forecasting

Xinshuo Weng[1], Junyu Nan[1], Kuan-Hui Lee[2], Rowan McAllister[2], Adrien Gaidon[2], Nicholas Rhinehart[3], and Kris Kitani[1]

[1] Robotics Institute, Carnegie Mellon University
{xinshuow,jnan1,kkitani}@andrew.cmu.edu
[2] Toyota Research Institute
{kuan.lee,rowan.mcallister,adrien.gaidon}@tri.global
[3] Berkeley Artificial Intelligence Research Lab, University of California, Berkeley
nrhinehart@berkeley.edu

Abstract. Predicting futures of surrounding agents is critical for autonomous systems such as self-driving cars. Instead of requiring accurate detection and tracking prior to trajectory prediction, an object agnostic Sequential Pointcloud Forecasting (SPF) task was proposed [28], which enables a forecast-then-detect pipeline effective for downstream detection and trajectory prediction. One limitation of prior work is that it forecasts only a deterministic sequence of future point clouds, despite the inherent uncertainty of dynamic scenes. In this work, we tackle the stochastic SPF problem by proposing a generative model with two main components: (1) a conditional variational recurrent neural network that models a temporally-dependent latent space; (2) a pyramid-LSTM that increases the fidelity of predictions with temporally-aligned skip connections. Through experiments on real-world autonomous driving datasets, our stochastic SPF model produces higher-fidelity predictions, reducing Chamfer distances by up to 56.6% compared to its deterministic counterpart. In addition, our model can estimate the uncertainty of predicted points, which can be helpful to downstream tasks.

Keywords: Sequential point cloud forecasting; variational recurrent neural network; future uncertainty; self-driving cars.

1 Introduction

Uncertainty is an inherent challenge associated with different future prediction tasks such as video prediction [4,30,31,23], trajectory prediction [33,15,18,29] and sequential pointcloud forecasting (SPF) [28,21]. For SPF [28], given a sequence of past point clouds captured by the LiDAR sensor, future point clouds are uncertain because the future behaviors of entities in the scene are uncertain. Although prior work in video prediction [7,20] and trajectory prediction [25,33] has made significant advancements in developing generative models to handle

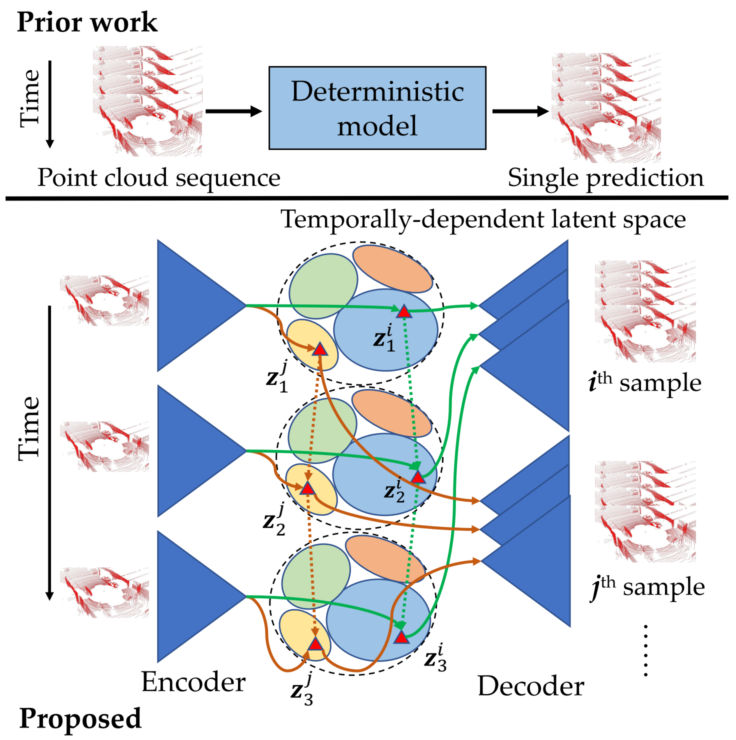

Fig. 1. Different from prior work (top) that uses a deterministic model to predict a single future of point clouds, our S2Net (bottom) can sample (*e.g.*, green and brown) sequences of latent variables to tackle future uncertainty. Also, each sequence of latent variables is temporally-dependent to ensure consistency of forecasting.

future uncertainty, prior work in SPF is limited by predicting a deterministic sequence of future point clouds.

In this work, we propose S2Net[4], a *Stochastic* SPFNet that, for the first time, enables prediction of stochastic future point clouds. Specifically, we follow the LSTM (Long Short-Term Memory) encoder-decoder style as in [28] but extend it with a conditional variational LSTM. At each frame, the LSTM hidden state propagates past information up to the current frame, which we use as the context and map to a prior distribution. Then, we can sample from this prior distribution to predict the point cloud in the next frame through a decoder. As future point clouds are supposed to be temporally-consistent, the uncertainty at each frame should depend on the last frame. Therefore, we enforce the temporal dependency between sampled latent variables across time as shown in Fig. 1.

[4] Our project website is at `https://www.xinshuoweng.com/projects/S2Net`.

Another limitation of prior work is blurry predictions or excessive smoothness in the predicted point clouds. We alleviate this issue and improve the fidelity of predicted point clouds by adding skip connections between encoders and decoders. Specifically, a pyramid of features at different levels of encoders are fed into the decoders, which reduces the complexity of training the decoder because now it only needs to learn residuals of features. However, naively adding skip connections performs poorly in the SPF task as the timestamp of features in the encoder is not aligned with the timestamp of features in the decoder. To tackle this issue, we add additional LSTMs at every level of the pyramids before skip connection so that we fuse temporally-aligned features in the decoder.

Through experiments on KITTI and nuScenes, we show that the pyramid LSTMs is a key to produce sharper and more accurate point clouds than the simple LSTM encoder-decoder framework in [28]. Combining with the conditional variational LSTM, we reduce predictions errors by 56.6% relative to its deterministic counterpart when sampling five predictions. By computing the standard deviation of every point across five samples, we can measure how certain the network is about every predicted point, which can be useful to downstream tasks. Also, compared to [28], the addition of conditional variational LSTM and temporally-aligned skip connections does not add significant computational overhead, so our method is still efficient and able to predict a sequence of large-scale LiDAR point clouds (*e.g.*, 122,880 points per frame) for a time horizon of 3 seconds. Our contributions are summarized below:

1. a *stochastic* SPF model (S2Net) with temporally-dependent latent space that can sample predictions of point clouds to tackle future uncertainty;
2. a pyramid-LSTM module that can increase the fidelity and sharpness of the predicted future point clouds over prior work;
3. an efficient network design that allows prediction of a sequence of large-scale LiDAR point clouds over a long horizon.

2 Related Work

Sequential Pointcloud Forecasting was proposed in [28] for large-scale LiDAR point clouds. It has been shown that, by scaling up the learning of SPF in a fully unsupervised manner, the downstream trajectory prediction task can be improved. Instead of using 1D-LSTM, [27] proposed to use ConvLSTM for SPF. Instead of directly predicting point locations, [9] proposes to predict flow which can be added to the input point cloud to generate future point locations. However, [9] only predicts the next frame of point cloud, which does not strictly follow the design of sequence prediction. Recently, [21] proposes to use 3D convolutions to jointly encode spatial and temporal information for SPF, which shows improved performance over [28]. However, all these prior works propose deterministic models, whereas our method can predict stochastic futures of point cloud sequences to tackle future uncertainty.

Sequence Point Cloud Processing is closely related to SPF as it also processes a sequence of point clouds. The difference to SPF is that, the term "point cloud" here is loose and does not necessarily mean large-scale LiDAR point cloud. For example, it can be processing of image pixels such as moving MNIST [12] with $64 \times 64 = 4096$ pixels as in [14], or processing severely downsampled LiDAR point clouds of about $1,024$ points as in [12], or processing mobile traffic with about $5,000$ points as in [34], or processing human hand joints with less than 50 points as in [22]. Because it does not necessarily process large-scale LiDAR point clouds, network efficiency is not a critical concern in prior work of sequence point cloud processing. As a result, these works may not be directly transferred to SPF and process full LiDAR point cloud data, although we can draw inspiration from these works in designing our method for SPF. In contrast, our method is designed specifically for SPF, and can operate on a sequence of $\geq 100,000$ points, *i.e.*, a total of $\geq 1M$ points for 10 frames.

Variational Models in Sequence Modeling. Stochastic recurrent networks [3,8,6] are commonly used to model variations and uncertainty in sequential data, which is significantly different from standard VAE/CVAE[16] that is designed to model variations for a single frame of data. STORNs [3] incorporates a latent variable into every timestamp of the recurrent network, where the latent variables are independent across time. VRNN [8] models the dependency between the RNN's hidden state at the current timestamp and the latent variable in the last timestamp. As the latent variable in the current timestamp also depends on the RNN's hidden state, there is an indirect temporal dependency between latent variables. [6] further improves VRNN by allowing direct temporal dependency between latent variables[5]. However, the use of these stochastic recurrent networks in large-scale future prediction tasks is still under-explored, and in this work we explore how we can adapt them to stochastic SPF.

Point Cloud Generation. Most prior work focuses on generating a point cloud of a single object at a single frame. [11] proposes to optimize object point cloud generation using Chamfer distance [11] and Earth Mover's distance [24]. [19] projects the point cloud into 2d images from multiple views and optimizes for the re-projection error. Other works focus on stochastic object point cloud generation, where normalizing flow [32], generative adversarial networks (GANs)[26], or variational autoencoders (VAEs) [17] can be used. Different from these works which are designed to generate the point cloud for an object or a scene at a single frame, our work aims to *predict* a *long horizon* of *large-scale* point clouds.

3 Approach: S2Net

The goal of stochastic SPF (Sequential Pointcloud Forecasting) is to learn a function that can sample different sequences of future point clouds given a sequence of observed past point clouds. Let $\mathcal{S}_p = \{\boldsymbol{S}_{1-M}, \ldots, \boldsymbol{S}_0\}$ denote M frames of past point clouds of a scene and $\mathcal{S}_f = \{\boldsymbol{S}_1, \ldots, \boldsymbol{S}_N\}$ denote N frames of future point clouds. Each frame of point cloud $\boldsymbol{S}_t = \{(x, y, z)_j\}_{j=1}^{K_t} = \{\boldsymbol{u}_j\}_{j=1}^{K_t}$ contains a set of unordered points, where $t \in [1\text{-}M, \cdots, N]$ denotes the frame index,

[5] An illustrative comparison between these works can be found in [6].

$j \in [1, \cdots, K_t]$ denotes the index of point and K_t denotes the number of points at frame t. Then, we aim to learn a function $\mathcal{F}$ that can map $\mathcal{S}_p$ and sampled latent variables $\mathcal{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ to a sequence of future point clouds $\mathcal{S}_f$:

$$\mathcal{S}_f = \mathcal{F}(\mathcal{S}_p, \mathcal{Z}). \tag{1}$$

In this work, we use LiDAR point clouds as $\mathcal{S}$ to validate our approach, but our approach can also be applied to other types of point clouds such as those captured by indoor RGB-D cameras or generated by stereo images. The main challenge of working with LiDAR point clouds is that the data is typically large-scale, that is more than $100,000$ points per frame or 1 million of points for a sequence of 10 frames, so efficiency is the key in approach design.

The network architecture of our approach is illustrated in Fig. 2, which contains the following parts: 1) a shared encoder to obtain features $\boldsymbol{r}_t^l$ at different levels of the pyramid and at all frames, where $t \in [1-M, \cdots, N]$ denotes the frame index and $l \in [1, \cdots, L]$ denotes the pyramid level; 2) a pyramid of LSTMs to propagate features from the past frame $\boldsymbol{h}_{t-1}^l$ to the future $\boldsymbol{h}_t^l$; 3) a temporally-dependent latent model that computes the prior distribution and sample latent variable $\boldsymbol{z}_t$ conditioned on previous latent $\boldsymbol{z}_{t-1}$ and current hidden state $\boldsymbol{h}_t^1$; 4) a shared decoder to predict the next frame's point cloud $\mathcal{S}_t'$ given $\boldsymbol{z}_t$, $\boldsymbol{h}_t^l$, where $l \in [1, \cdots, L]$. Here we omit the details about how to compute the prior and posterior distribution to sample latent variables during training and testing, which can be found in Sec. 3.4. During testing, our model performs auto-regressive prediction, which uses the predicted point cloud as input for the next frame.

3.1 Input/Output Representation: Range Map

To represent a point cloud $\boldsymbol{S} = \{(x, y, z)_j\}_{j=1}^K$, we follow prior work [21,28] and use the range map representation $\mathcal{R} \in \mathbb{R}^{H \times W \times 1}$. Each range map can be viewed as a 1-channel image, with every pixel corresponding to a point in 3D and the pixel value storing the Euclidean distance $d = \sqrt{x^2 + y^2 + z^2}$ of the point to the LiDAR sensor. To convert a point cloud to a range map, we use spherical projection. Specifically, for a point $p = (x, y, z)$ in the Cartesian coordinate (z is the height), its coordinate in the range map $q = (\theta, \phi)$ can be computed as:

$$\theta = \arctan(y/x), \qquad \phi = \arcsin(z/\sqrt{x^2 + y^2 + z^2}) = \arcsin(z/d), \tag{2}$$

where θ, ϕ are the azimuth, elevation angle. Since d is stored in the range map, one can also apply inverse spherical projection to recover the $p = (x, y, z)$ as:

$$z = \sin(\phi) \times d, \qquad d_{\text{ground}} = \cos(\phi) \times d, \tag{3}$$

where d_{ground} is the projected distance on the x-y ground plane. Then,

$$x = \cos(\theta) \times d_{\text{ground}}, \qquad y = \sin(\theta) \times d_{\text{ground}}. \tag{4}$$

Although range map is an image representation, it is very different from other image representations which often involves loss of information during projection

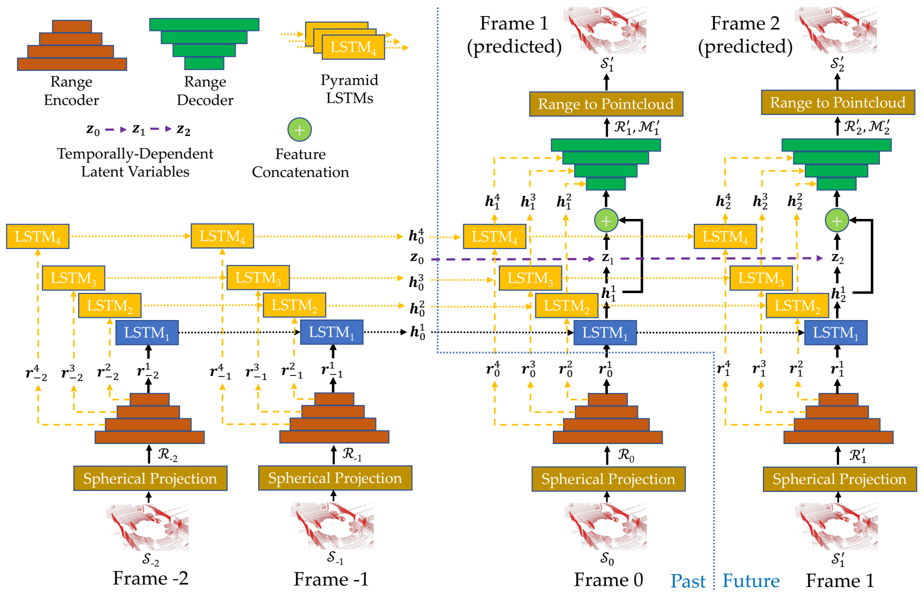

Fig. 2. (Left) **Extracting features from past frames**. Given a point cloud $\mathcal{S}_t$ in the past, we first convert it into a 2D range map $\mathcal{R}_t$ (spherical projection) and then extract features $\boldsymbol{r}_t^l = \{\boldsymbol{r}_t^1, \boldsymbol{r}_t^2, \cdots, \boldsymbol{r}_t^L\}$ at different layers with a 2D CNN (range encoder). To learn temporal dynamics, features at different layers $\boldsymbol{r}_t^l$ are propagated through time via pyramid LSTMs and output hidden states $\boldsymbol{h}_0^l = \{\boldsymbol{h}_0^1, \boldsymbol{h}_0^2, \cdots, \boldsymbol{h}_0^L\}$. (Right) **Autoregressive future prediction.** To begin at input frame 0, given propagated hidden states $\boldsymbol{h}_0^l$ and extracted feature pyramids $\boldsymbol{r}_0^l$, we further propagate the LSTM hidden states into frame 1. Combining $\boldsymbol{h}_1^1$ with the initialized latent variable $\boldsymbol{z}_0$, a prior distribution is computed during testing where we can sample multiple $\boldsymbol{z}_1$. We then predict the range map $\mathcal{R}'_1$ and range mask $\mathcal{M}'_1$ via the range decoder, conditioned on the propagated hidden states $(\boldsymbol{h}_1^1, \boldsymbol{h}_1^2, \cdots, \boldsymbol{h}_1^L)$ and sampled latent variable $\boldsymbol{z}_1$. For illustration, we only show using three past frames of inputs $(\mathcal{S}_{-2}, \mathcal{S}_{-1}, \mathcal{S}_0)$ to predict two frames of future $(\mathcal{S}'_1, \mathcal{S}'_2)$. Also, only one branch of the encoder-decoder is shown while we have two sets of encoders and decoders for range map and range mask separately.

such as the camera image with perspective projection (losing depth information) or the Bird's eye view projection (losing height information). In contrast, there is no loss of information in theory[6] when converting a point cloud to range map since the range map preserves full LiDAR data. Also, it is very efficient to process range maps with CNNs (Convolutional Neural Networks) due to the nature of image representation, which is the key to our efficient network design.

[6] There might be still a small amount of information loss due to discretization unless one uses very high-resolution range map as we do in the experiments.

3.2 Range Encoder

To process the range map $\mathcal{R}_t$ converted from the point cloud $\mathcal{S}_t$, we follow [28] and use a 2D CNNs as shown in reddish-brown in Fig. 2. This 2D CNNs contains a series of blocks, with each block having a 2D convolution, batch normalization (BN) and LeakyReLU activation, except for the last block which does not have BN and LeakyReLU. The output of the encoder is a pyramid of features $[\boldsymbol{r}_t^1, \cdots, \boldsymbol{r}_t^L]$ after each layer of convolution. The range encoder is shared for all past frames $(\mathcal{R}_{1-M}, \cdots, \mathcal{R}_0)$ and also used to extract features from the predicted range maps $(\mathcal{R}'_1, \cdots, \mathcal{R}'_{N-1})$.

In addition to the range map encoder, we have another range mask encoder that has the exactly same architecture as the range map encoder, which we omit in Fig. 2 for simplicity. The goal of the range mask encoder is to learn features to predict the range mask $\mathcal{M}'_t$, which masks out "empty" pixels[7] in the predicted range map $\mathcal{R}'_t$. The inputs to the range mask encoder are the same as the range map encoder, *i.e.*, $\mathcal{R}_t$, while the weights of two encoders are not shared.

3.3 Pyramid LSTMs

After extracting features at different levels $\boldsymbol{r}_t^l$ for all past frames, we propagate them through time using LSTMs to learn temporal dynamics of the point cloud sequence. As the encoder CNNs output a vector in the last layer $\boldsymbol{r}_t^1$, we use standard LSTM as shown in blue in Fig. 2 to propagate $\boldsymbol{r}_t^1$ through time. However, for features at other levels except for the last layer, we use Convolutional LSTMs (ConvLSTM) as shown in orangish-yellow in Fig. 2 because $\boldsymbol{r}_t^l$ is a 2D feature map when $l > 1$. The outputs of pyramid LSTMs at frame $t-1$ are propagated to obtain hidden states $\boldsymbol{h}_t^l$ (with a shift of one timestep), where $l \in [1, \cdots, L]$.

The motivation to keep other levels of features (besides $\boldsymbol{r}_t^1$) being propagated through time is to prepare for temporally-aligned skip connection in the decoder at multiple pyramid levels, which we realize is a key to increase the prediction accuracy compared to using only one level of features as shown in the ablation in Table 3 (right). Also, if only extracting feature pyramids for naive skip connection without propagating them through time, *e.g.*, directly feeding $\boldsymbol{r}_0^2, \boldsymbol{r}_0^3, \boldsymbol{r}_0^4$ (rather than $\boldsymbol{h}_1^2, \boldsymbol{h}_1^3, \boldsymbol{h}_1^4$) to the decoder at frame 0, the timestamps of features in the decoder are not consistent which turns out can mess up the training.

3.4 CVAE Prior and Posterior

To enable prediction of stochastic futures, we adapt standard CVAE (Conditional Variational Auto-encoder) framework to our sequence-to-sequence problem. As shown in related stochastic video prediction work [10], learnable prior $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of the Gaussian are conditioned on context features, works better

[7] Unlike RGB images, not every pixel in the range map is valid. This is because there are pixels without corresponding points in 3D due to no return of the LiDAR beam in that direction, *e.g.*, when the LiDAR beam shots to the sky.

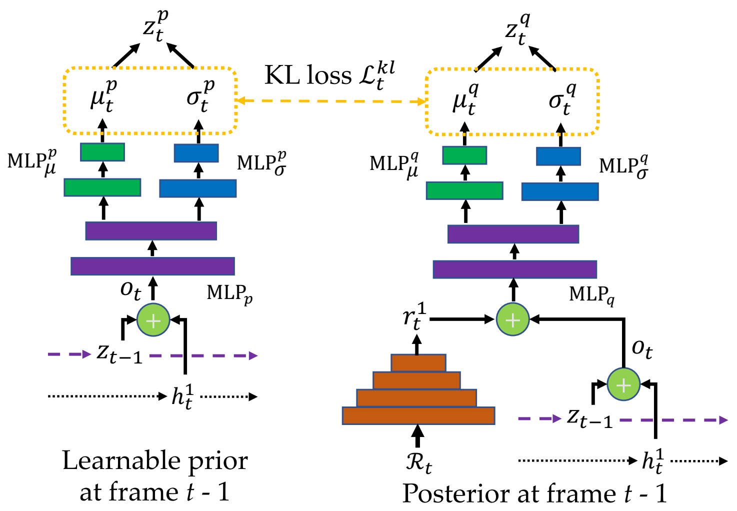

Fig. 3. (Left) **Computing the prior**. To better accommodate prediction at a long horizon, we learn the prior $\mathcal{N}(\boldsymbol{\mu}_t^p, \boldsymbol{\sigma}_t^{p2}) = \mathcal{N}(\mu_{\phi_1}(\mathcal{S}_{1-M:t-1}), \sigma_{\phi_2}(\mathcal{S}_{1-M:t-1}))$, instead of using a fixed Gaussian $\mathcal{N}(0, \mathbf{I})$. The inputs are the concatenated feature of $\boldsymbol{z}_{t-1}$ and $\boldsymbol{h}_t^1$, where $\boldsymbol{h}_t^1$ is the context feature while $\boldsymbol{z}_{t-1}$ enforces temporal dependency in the latent space. We then compute the mean $\boldsymbol{\mu}_t^p$ and variance $\boldsymbol{\sigma}_t^p$ via MLPs so we can sample latent variable $\boldsymbol{z}_t^p$. (Right) **Computing the posterior**. Except for one additional input, that is the future range map $\mathcal{R}_t$ and its extracted feature $\boldsymbol{r}_t^1$, we use the same architecture as the learnable prior to compute the posterior, although weights are not shared between posterior and prior computation. A KL divergence loss $\mathcal{L}_t^{\text{kl}}$ is applied between the computed prior and posterior distributions during training.

than fixed Gaussian prior $\mathcal{N}(0, \mathbf{I})$ in the sequence-to-sequence problem, so we also use learnable prior for our S2Net.

As shown in Fig. 3 (left), the inputs to compute the prior at frame $t-1$ are sampled latent variable $\boldsymbol{z}_{t-1}$ and the context feature $\boldsymbol{h}_t^1$ (propagated LSTM hidden state). Two inputs are first concatenated to obtain an intermediate feature $\boldsymbol{o}_t$, which is then fed into a joint MLP$_p$ (Multi-Layer Perceptron) followed by two separate MLP$_\mu^p$ and MLP$_\sigma^p$ to compute the mean $\boldsymbol{\mu}_t^p$ and variance $\boldsymbol{\sigma}_t^p$ of the prior distribution at frame $t-1$. During testing time, we can then randomly sample latent variables $\boldsymbol{z}_t^p$ from the prior distribution $\mathcal{N}(\boldsymbol{\mu}_t^p, \boldsymbol{\sigma}_t^{p2}) = \mathcal{N}(\mu_{\phi_1}(\mathcal{S}_{1-M:t-1}), \sigma_{\phi_2}(\mathcal{S}_{1-M:t-1}))$ to enable stochastic prediction, where the MLPs are parametrized by ϕ_1 and ϕ_2. The process can be summarized below:

$$\boldsymbol{o_t} = \boldsymbol{z}_{t-1} \oplus \boldsymbol{h}_t^1, \tag{5}$$

$$\boldsymbol{\mu}_t^p = \text{MLP}_\mu^p(\text{MLP}_p(\boldsymbol{o}_t)), \;\; \boldsymbol{\sigma}_t^p = \text{MLP}_\sigma^p(\text{MLP}_p(\boldsymbol{o}_t)), \tag{6}$$

$$\boldsymbol{z}_t \sim \mathcal{N}(\boldsymbol{\mu}_t^p, \boldsymbol{\sigma}_t^{p2}). \tag{7}$$

Similar to computing the prior distribution during testing, we compute the posterior distribution $\mathcal{N}(\boldsymbol{\mu}_t^q, \boldsymbol{\sigma}_t^{q2})$ (Fig. 3 right) during training for the KL loss.

The main difference when computing the posterior at frame $t-1$ is the use of future range map $\mathcal{R}_t$ as input, in addition to $\boldsymbol{z}_{t-1}$ and the context $\boldsymbol{h}_t^1$. Here, a future range encoder that has the same architecture as the range encoder (Sec. 3.2) is used to extract features from the future range data $\mathcal{R}_t$. Similarly, we can sample latent variable $\boldsymbol{z_t^q} \sim \mathcal{N}(\boldsymbol{\mu}_t^q, \boldsymbol{\sigma}_t^{q2}) = \mathcal{N}(\mu_{\phi_3}(\mathcal{S}_{1-M:t}), \sigma_{\phi_4}(\mathcal{S}_{1-M:t}))$, where ϕ_3 and ϕ_4 summarize parameters of the $\mathrm{MLP}_q, \mathrm{MLP}_\mu^q, \mathrm{MLP}_\sigma^q$ and the future range encoder. Then, the latent variable $\boldsymbol{z_t^q}$ is passed to the decoder during training. Note that during testing, we only compute the prior as in the standard CVAE formulation because we do not need the KL loss and also the future range data (ground truth) $\mathcal{R}_t$ is not available at testing.

3.5 Temporally-Dependent Latent Space

Given the prior and posterior, we can sample the latent variable at every frame. However, if we sample latent variables independently at every frame, it is not likely to generate temporally-consistent variations in the predicted point clouds as the uncertainty is drawn from independent samples. This motivates us to enforce temporal dependency in the latent space, that is the latent variable $\boldsymbol{z}_t$ is directly conditioned on the latent variable $\boldsymbol{z}_{t-1}$ sampled in the previous frame as shown in Fig. 3. We realize that this temporally-dependent latent space is especially useful to improve the performance of point cloud prediction for a long time horizon. The first latent variable $\boldsymbol{z}_0$ is initialized with all zeros.

3.6 Range Decoder

To obtain predicted point clouds at every future frame, we first predict the range map $\mathcal{R}'_t$ and range mask $\mathcal{M}'_t$. Then, the range map $\mathcal{R}'_t$ is used to generate point clouds via inverse spherical projection as explained in Eq. 3 and 4, and the range mask is used in a mask operation to prevent "empty" pixels in the range map to generate "bad" points. The mask decoder has the architecture as the range map decoder except that the weights of decoders are different.

Now we describe how we predict the range map or mask. As we formulate the stochastic SPF in a CVAE framework, the range decoder at frame $t-1$ is conditioned on the context feature (LSTM hidden state $\boldsymbol{h}_t^1$), in addition to the sampled latent variable $\boldsymbol{z}_t$. Two inputs are concatenated before feeding into the range decoder, as shown in Figure 2. To reduce the complexity of predicting full-resolution range map $\mathcal{R}'_t$ from only two low-dimensional features ($\boldsymbol{h}_t^1$ and $\boldsymbol{z}_t$), we add the temporally-aligned skip connections from pyramid LSTMs. As higher-resolution feature maps are added as inputs to the decoder, we only need to learn the residual of features for small local changes after one frame, which increases the fidelity of point cloud prediction. In terms of the architecture, the range decoder has the inverse structure as the range encoder introduced in Sec. 3.2 except that the input channel of the first layer is increased to accommodate the added dimension of the latent variable $\boldsymbol{z}_t$.

3.7 Training Objective

Since we formulate our stochastic SPF in a sequential CVAE framework, we use the negative evidence lower bound (ELBO) $\mathcal{L}_{\text{elbo}}$ as our loss function:

$$\mathcal{L}_{\text{elbo}} = \mathbb{E}_{q(\boldsymbol{z}_{\leq N}|\mathcal{S}_{\leq N})}\left[\sum\nolimits_{t=1}^{N}\Big(-\log p(\mathcal{S}_t|\boldsymbol{z}_{\leq t}, \mathcal{S}_{<t}) + \text{KL}(\ q(\boldsymbol{z}_t|\mathcal{S}_{\leq t}, \boldsymbol{z}_{<t}) \parallel p(\boldsymbol{z}_t|\mathcal{S}_{<t}, \boldsymbol{z}_{<t})\ )\Big)\right], \quad (8)$$

where the KL loss aims to minimize the difference between the prior distribution $p(\boldsymbol{z}_t|\mathcal{S}_{<t}, \boldsymbol{z}_{<t})$ and posterior distribution $q(\boldsymbol{z}_t|\mathcal{S}_{\leq t}, \boldsymbol{z}_{<t})$. Also, to maximize the log-likelihood $\log p(\mathcal{S}_t|\boldsymbol{z}_{\leq t}, \mathcal{S}_{<t})$, we use the following reconstruction losses:

1. Chamfer distance [11] $\mathcal{L}_t^{\text{cd}}$ between the predicted point cloud $\mathcal{S}'_t$ (after points masking) and ground truth (GT) point cloud $\mathcal{S}_t$:

$$\mathcal{L}_t^{\text{cd}} = \sum\nolimits_{\boldsymbol{u}' \in \mathcal{S}'_t} \min_{\boldsymbol{u} \in \mathcal{S}_t} \|\boldsymbol{u} - \boldsymbol{u}'\|_2^2 + \sum\nolimits_{\boldsymbol{u} \in \mathcal{S}_t} \min_{\boldsymbol{u}' \in \mathcal{S}'_t} \|\boldsymbol{u} - \boldsymbol{u}'\|_2^2, \quad (9)$$

2. L_1 distance $\mathcal{L}_t^{L1}$ between the predicted range map $\mathcal{R}'_t$ and GT range map $\mathcal{R}_t \in \mathbb{R}^{H \times W \times 2}$ at every valid pixel:

$$\mathcal{L}_t^{\text{L1}} = \sum\nolimits_i^H \sum\nolimits_j^W \|\mathcal{M}_{t\,i,j}(\mathcal{R}'_{t\,i,j} - \mathcal{R}_{t\,i,j})\|_1, \quad (10)$$

3. A binary cross-entropy loss $\mathcal{L}_t^{bce}$ between the predicted mask $\mathcal{M}'_t$ and GT mask $\mathcal{M}_t$ at every pixel:

$$\mathcal{L}_t^{\text{bce}} = \sum\nolimits_i^H \sum\nolimits_j^W -\mathcal{M}_{t\,i,j} \log(\mathcal{M}'_{t\,i,j}) - (1 - \mathcal{M}_{t\,i,j}) \log(1 - \mathcal{M}'_{t\,i,j}), \quad (11)$$

Combining the reconstruction losses with the KL loss, our full loss function is:

$$\mathcal{L} = \sum\nolimits_{t=1}^{N} (\mathcal{L}_t^{\text{kl}} + \mathcal{L}_t^{\text{reconstruction}}) = \sum\nolimits_{t=1}^{N} (\beta\mathcal{L}_t^{\text{kl}} + \alpha\mathcal{L}_t^{\text{cd}} + \kappa\mathcal{L}_t^{\text{L1}} + \omega\mathcal{L}_t^{\text{bce}}). \quad (12)$$

We note that our $\mathcal{L}_{\text{elbo}}$ is different from many CVAE-based methods [25,33] because our reconstruction loss has three components (rather than only a L2 loss for regression). But we will show in the supplementary that all three components are necessary to achieve strong practical performance.

3.8 Training and Testing Details

To stabilize the LSTM training, especially for large-scale data over a long time horizon, we apply teacher forcing [9]. This means that, at the beginning of the training when prediction is of low quality, we use GT point clouds $(\mathcal{S}_1, \mathcal{S}_2, \cdots)$ as inputs at future frames to ease the training. Then, we gradually increase the probability of using predicted point clouds $(\mathcal{S}'_1, \mathcal{S}'_2, \cdots)$ as inputs, which matches the style of autoregressive prediction during testing.

When generating GT range mask for training, we let all pixels have a value of 1 if they have corresponding 3D points after spherical projection, while let all pixels have a value of 0 if they do not have corresponding projected points. During testing, as the predicted range mask $\mathcal{M}'_t$ contains scalars between 0 and 1 after the softmax function, we use a threshold T_m to determine whether to mask out the point generated from every pixel of the predicted range map.

4 Experiments

Hyper-parameters. For KITTI [13], we use $H = 60$ and $W = 2048$ for the input range map, which results in a total of $K = 122880$ points per frame. For nuScenes [5], we use $H = 28$ and $W = 1024$ for the input range map, which results in a total of $K = 28672$ points per frame. We use $\alpha = \kappa = \omega = 1$ and $\beta = 3e-5$ for the loss weights, and $T_m = 0.05$ as the threshold of the mask.

Network details. To extract features from the range map, we use a 2D CNN with 8 blocks of convolution, BN and LeakyReLU by increasing the channel dimensions from the input of 2 to the output of 512 ($2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 512$). Then, we use output features at all layers as inputs to our pyramid LSTMs so $L = 8$. The feature dimension of latent variables $\boldsymbol{z}_t$ is 32 and the feature dimension of hidden states $\boldsymbol{h}_t^1$ is 512.

4.1 Evaluation Methodology

Datasets. We use real-world autonomous driving datasets (KITTI [13] and nuScenes [5]) for training and evaluation. For nuScenes, we use the official data split [2] of the nuScenes prediction challenge (500, 200, 150 sequences for train, val, test). We evaluate the prediction horizon of 1 and 3 seconds. For KITTI, we use the raw dataset [1], which is a superset of the KITTI odometry, tracking, detection datasets, *etc.* As there is no official data split for the KITTI raw dataset, we created our own balanced data split.

Metrics. To evaluate the accuracy of predicted point clouds, we use standard Chamfer Distance (CD) [11] (Sec. 3.7) and Earth Mover's Distance (EMD) [24] to measure the distance between a sequence of GT future point clouds $\mathcal{S}_f$ and a sequence of predicted future point clouds $\mathcal{S}'_f$:

$$\mathrm{EMD}(\mathcal{S}_f, \mathcal{S}'_f) = \frac{1}{N} \sum\nolimits_{t=1}^{N} \min_{\phi: \mathcal{S}'_t \rightarrow \mathcal{S}_t} \sum\nolimits_{\boldsymbol{u}' \in \mathcal{S}'_t} \|\boldsymbol{u}' - \phi(\boldsymbol{u}')\|_2. \tag{13}$$

To adapt standard EMD and CD to evaluate our stochastic model, we follow similar idea in the domain of trajectory prediction by computing the best of k CD and EMD ($\mathrm{minCD}_k, \mathrm{minEMD}_k$) over K_{sa} predicted samples, where each sample ${\mathcal{S}_f}'_k$ is a sequence of predicted point clouds:

$$\mathrm{minCD}_k = \min\nolimits_{k \in (1, \cdots, K_{\mathrm{sa}})} \mathrm{CD}(\mathcal{S}_f, {\mathcal{S}_f}'_k), \; \mathrm{minEMD}_k = \min\nolimits_{k \in (1, \cdots, K_{\mathrm{sa}})} \mathrm{EMD}(\mathcal{S}_f, {\mathcal{S}_f}'_k), \tag{14}$$

where we use $K_{\mathrm{sa}} = 5$ for evaluation on both nuScenes and KITTI. To measure the accuracy of predicted range maps, we use standard L_1, L_2 distance and also the perceptual metrics including Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR).

Baselines. As SPF is still under-explored, there are mainly two baselines to our knowledge which strictly follows the SPF design, *i.e.*, [28] and [21]. We evaluate

Table 1. Comparison on the test split of the KITTI raw dataset.

Prediction horizon	Methods	Range Map				Point Cloud	
		L1↓	L2↓	SSIM↑	PSNR↑	CD↓	EMD↓
1 second	SPFNet [28]	0.865	3.108	0.845	24.610	0.341	123.826
	[21]	0.707	3.603	**0.873**	**30.453**	0.202	96.426
	S2Net (w/ Pyramid LSTMs only)	0.723	2.745	0.754	25.801	0.203	106.830
	S2Net (w/ Pyramid LSTMs + Temporally-dependent latent)	**0.675**	**2.433**	0.868	30.294	**0.148**	**84.418**
3 seconds	SPFNet [28]	1.086	4.586	0.827	27.092	0.562	135.690
	[21]	1.187	6.069	0.795	25.935	0.379	143.194
	S2Net (w/ Pyramid LSTMs only)	1.008	4.413	0.869	29.668	0.360	130.325
	S2Net (w/ Pyramid LSTMs + Temporally-dependent latent)	**0.983**	**4.374**	**0.894**	**31.647**	**0.293**	**120.874**

Table 2. Comparison on the test split of the nuScenes prediction dataset.

Prediction horizon	Methods	Range Map				Point Cloud	
		L1↓	L2↓	SSIM↑	PSNR↑	CD↓	EMD↓
1 second	SPFNet [28]	0.669	1.406	0.708	**23.545**	0.456	239.798
	S2Net (w/ Pyramid LSTMs only)	0.622	**1.196**	0.759	21.118	0.366	189.336
	S2Net (w/ Pyramid LSTMs + Temporally-dependent latent)	**0.545**	1.285	**0.785**	23.274	**0.321**	**172.069**
3 seconds	SPFNet [28]	0.706	1.869	**0.717**	**23.587**	0.522	238.239
	S2Net (w/ Pyramid LSTMs only)	0.712	1.644	0.702	21.452	0.421	**205.269**
	S2Net (w/ Pyramid LSTMs + Temporally-dependent latent)	**0.560**	**1.261**	0.685	20.870	**0.381**	209.267

SPFNet [28][8] on both KITTI/nuScenes, while we evaluate [21] only on the KITTI dataset because [21][9] did not provide method description or configuration files on the nuScenes or other datasets.

4.2 Results and Analysis

Results on KITTI. Results on the test set are summarized in Table 1. Unsurprisingly, our method outperforms SPFNet [28] and the most recent work [21] in all settings (1 or 3 seconds of horizon). Also, it is important to note that both of the key innovations of our method (Pyramid LSTMs and temporally-dependent latent space) are effective for the SPF task! For example, our method reduces the CD in the 1-second setting by 40.5% (from 0.341 to 0.203) after adding Pyramid LSTMs, and then reduces the CD in the 1-second setting by 27.1% (from 0.203 to 0.148) after adding temporally-dependent latent space, with an overall of 56.6% error reduction (from 0.341 to 0.148) compared to SPFNet.

Results on nuScenes. We summarize the results in Table 2. Similar to the trend in the KITTI experiments, S2Net also outperforms the SPFNet baseline for both 1-second and 3-seconds settings. One interesting finding is that the overall magnitude of CD and EMD on nuScenes is higher than those on KITTI. We believe that this is due to sparser point clouds[10] in nuScenes. In other words,

[8] Note that the original numbers reported in [28] are different due to various changes including improved network architectures, balanced data split, improved implementation of the metrics, *etc.*

[9] Even for KITTI, [21] has only evaluated on the odometry dataset with only 22 sequences, *i.e.*, a subset of the raw dataset, so we have re-trained their model on the full KITTI raw dataset using the official code and KITTI configuration files.

[10] We did not aggregate point clouds along the temporal dimension.

Table 3. Ablation study. **(Left)**: Effect of temporal dependency of the latent space. **(Right)**: Effect of pyramid LSTMs on the KITTI dataset.

Datasets	Horizon	Methods	CD	EMD
KITTI	1 second	No dependency	0.171	87.564
		Indirect dependency	0.155	92.586
		Direct dependency	**0.148**	**84.418**
	3 seconds	No dependency	0.357	**120.391**
		Indirect dependency	0.316	126.498
		Direct dependency	**0.293**	120.874
nuScenes	1 second	No dependency	0.352	194.640
		Indirect dependency	0.334	173.642
		Direct dependency	**0.321**	**172.069**
	3 seconds	No dependency	0.485	212.755
		Indirect dependency	0.482	222.665
		Direct dependency	**0.381**	**209.267**

Horizon	Pyramid LSTMs	No. of Pyramids	CD	EMD
1 second		7	0.366	130.586
	✓	1	0.196	110.593
	✓	3	0.176	100.439
	✓	5	0.163	94.427
	✓	7	**0.148**	**84.418**
3 seconds		7	0.472	169.110
	✓	1	0.381	130.898
	✓	3	0.345	123.225
	✓	5	0.349	124.010
	✓	7	**0.293**	**120.874**

for each point, its nearest neighbor point in the GT point cloud generally has a larger distance in nuScenes.

Effect of Temporally-Dependent Latent Space. To validate this key innovation of our stochastic SPF approach, we experiment with two other variants of our method: 1) No dependency. We remove the dependency between $\boldsymbol{z}_{t-1}$ and $\boldsymbol{z}_t$ when computing the prior and posterior; 2) Indirect dependency. Instead of using $\boldsymbol{z}_{t-1}$ as a conditioning variable to $\boldsymbol{z}_t$, we now let $\boldsymbol{h}_t^1$ depends on $\boldsymbol{z}_{t-1}$. As $\boldsymbol{z}_t$ depends on the context $\boldsymbol{h}_t^1$, so there is an indirect dependency between $\boldsymbol{z}_{t-1}$ and $\boldsymbol{z}_t$. The results for three variants are summarized in Table 3 (left), where our method with direct dependency between latent variables achieves the best performance consistently, compared to indirect dependency and no dependency.

Effect of Pyramid LSTMs. To validate the effectiveness of another key innovation of our S2Net, *i.e.*, pyramid LSTMs, we experiment with the following variants of our method: 1) Decrease the number of pyramids we use in pyramid LSTMs, *e.g.*, $L = 1, 3, 5$, compared to $L = 7$ we use in our full model; 2) Keep $L = 7$ but remove the temporal alignment of the pyramid features, *i.e.*, removing the orangish-yellow ConvLSTM as shown in Fig. 2.

Results are summarized in Table 3 (right), where we use the temporally-dependent latent space for all variants. First, when we compare the 5th row with the 2nd, 3rd, 4th rows for both 1-second and 3-second settings, we can see that using all levels of feature pyramids $L = 7$ achieves the best performance compared to using fewer number of pyramids. This shows that adding the pyramid features is indeed useful. Also, when we compare the 1st row and all other rows, we can see that the prediction performance drops significantly if we remove the LSTMs between feature pyramids across timestamps. This confirms that naively adding skip connection to our prediction model is not sufficient and our design of pyramid LSTMs is reasonable.

Uncertainty Measurement. As our S2Net is a generative model, we can compute the standard deviation (SD) of the pixel value from multiple samples of

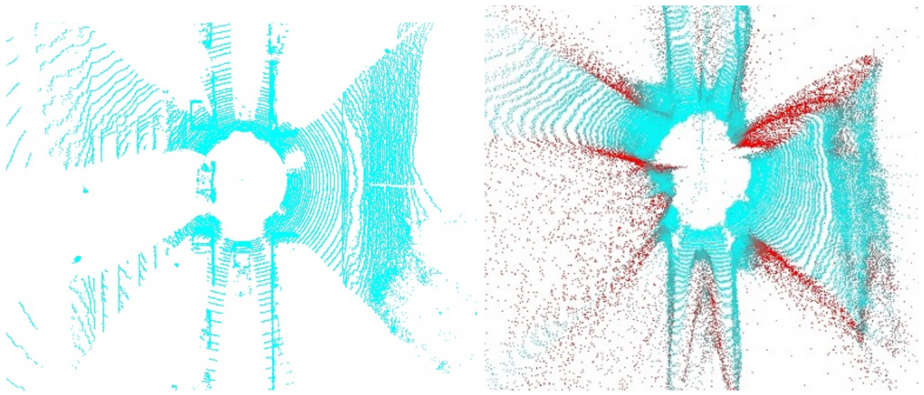

Fig. 4. (Left) GT point cloud. (Right) Predicted point cloud at the same frame colorized with computer point-wise uncertainty. The more red, the higher uncertainty.

predictions. We use this SD as the uncertainty measurement for each point. To understand which predicted points our S2Net is uncertain about, we visualize the predicted point cloud (1-second horizon) in Fig. 4 (right) where each point is colorized with the SD computed from 5 samples. We can see that points with high uncertainty (red) are mostly around the object boundary, which is reasonable since object's future motion is inherently uncertain.

Qualitative Results. Due to limited space, we refer readers to the supplementary for high-resolution visualization. The takeaway is that, with the addition of pyramid LSTMs and temporally-dependent latent space, our S2Net can predict higher-fidelity point clouds (better global structure, more clear freespace and sharper object boundary) compared to SPFNet on different scenes with pedestrians and cars. Also, we provide more visualization of uncertainty measurements in the supplementary, to show the inherent future uncertainty and the potential benefits to the downstream tasks.

5 Conclusion

Our work opened a new direction of predicting stochastic futures of large-scale LiDAR point clouds, which can deal with the inherent future uncertainty and strengthen the forecast-then-detect pipeline proposed in [28] for autonomous systems. We showed that both the addition of temporally-dependent latent space and pyramid LSTMs are critical to obtaining significantly better performance. In future work, we plan to investigate how this new stochastic SPF model can be applied to downstream tasks such as tracking and planning.

References

1. KITTI Raw Dataset, http://www.cvlibs.net/datasets/kitti/raw_data.php 11

2. nuScenes Prediction Data Split, https://github.com/nutonomy/nuscenes-devkit/blob/master/python-sdk/nuscenes/eval/prediction/splits.py 11
3. Bayer, J., Osendorfer, C.: Learning Stochastic Recurrent Networks. arXiv:1411.7610 (2014) 4
4. Bei, X., Yang, Y., Soatto, S.: Learning Semantic-Aware Dynamics for Video Prediction. CVPR (2021) 1
5. Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q.: nuScenes: A Multimodal Dataset for Autonomous Driving. CVPR (2020) 11
6. Castrejón, L., Ballas, N., Courville, A.C.: Improved conditional vrnns for video prediction. 2019 IEEE/CVF International Conference on Computer Vision (ICCV) pp. 7607–7616 (2019) 4
7. Chatterjee, M., Ahuja, N., Cherian, A.: A Hierarchical Variational Neural Uncertainty Model for Stochastic Video Prediction. ICCV (2021) 1
8. Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A.C., Bengio, Y.: A Recurrent Latent Variable Model for Sequential Data. NIPS (2015) 4
9. Deng, D., Zakhor, A.: Temporal LiDAR Frame Prediction for Autonomous Driving. 3DV (2020) 3, 10
10. Denton, E., Fergus, R.: Stochastic Video Generation with a Learned Prior. ICML (2018) 7
11. Fan, H., Su, H., Guibas, L.: A Point Set Generation Network for 3D Object Reconstruction from a Single Image. CVPR (2017) 4, 10, 11
12. Fan, H., Yang, Y.: PointRNN: Point Recurrent Neural Network for Moving Point Cloud Processing. arXiv:1910.08287 (2019) 4
13. Geiger, A., Lenz, P., Urtasun, R.: Are We Ready for Autonomous Driving? the KITTI Vision Benchmark Suite. CVPR (2012) 11
14. Gomes, P., Rossi, S., Toni, L.: Spatio-Temporal Graph-RNN for Point Cloud Prediction. ICIP (2021) 4
15. Gupta, A., Johnson, J., Fei-Fei, L., Savarese, S., Alahi, A.: Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks. CVPR (2018) 1
16. Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes. arXiv:1312.6114 (2013) 4
17. Klokov, R., Verbeek, J.J., Boyer, E.: Probabilistic reconstruction networks for 3d shape inference from a single image. In: BMVC (2019) 4
18. Liang, J., Jiang, L., Murphy, K., Yu, T., Hauptmann, A.: The Garden of Forking Paths: Towards Multi-Future Trajectory Prediction. CVPR (2020) 1
19. Lin, C.H., Kong, C., Lucey, S.: Learning efficient point cloud generation for dense 3d object reconstruction. In: AAAI (2018) 4
20. Liu, B., Chen, Y., Liu, S., Kim, H.S.: Deep Learning in Latent Space for Video Prediction and Compression. CVPR (2021) 1
21. Mersch, B., Chen, X., Behley, J., Stachniss, C.: Self-Supervised Point Cloud Prediction Using 3D Spatio-temporal Convolutional Networks. CoRL (2021) 1, 3, 5, 11, 12
22. Min, Y., Zhang, Y., Chai, X., Chen, X.: An Efficient PointLSTM for Point Clouds Based Gesture Recognition. CVPR (2020) 4
23. Nair, S., Savarese, S., Finn, C.: Goal-Aware Prediction: Learning to Model What Matters. ICML (2020) 1
24. Rubner, Y., Tomasi, C., Guibas, L.J.: The Earth Mover's Distance as a Metric for Image Retrieval. IJCV (2000) 4, 11

25. Salzmann, T., Ivanovic, B., Chakravarty, P., Pavone, M.: Trajectron++: Dynamically-Feasible Trajectory Forecasting With Heterogeneous Data. ECCV (2020) 1, 10
26. Shu, D.W., Park, S.W., Kwon, J.: 3d point cloud generative adversarial network based on tree structured graph convolutions. 2019 IEEE/CVF International Conference on Computer Vision (ICCV) pp. 3858–3867 (2019) 4
27. Sun, X., Wang, S., Wang, M., Wang, Z., Liu, M.: A Novel Coding Architecture for LiDAR Point Cloud Sequence. RA-L (2020) 3
28. Weng, X., Wang, J., Levine, S., Kitani, K., Rhinehart, N.: Inverting the Pose Forecasting Pipeline with SPF2: Sequential Pointcloud Forecasting for Sequential Pose Forecasting. CoRL (2020) 1, 2, 3, 5, 7, 11, 12, 14
29. Weng, X., Yuan, Y., Kitani, K.: PTP: Parallelized Tracking and Prediction with Graph Neural Networks and Diversity Sampling. Robotics and Automation Letters (2021) 1
30. Wu, B., Nair, S., Martin-Martin, R., Fei-Fei, L., Finn, C.: Greedy Hierarchical Variational Autoencoders for Large-Scale Video Prediction. CVPR (2021) 1
31. Wu, Y., Gao, R., Park, J., Chen, Q.: Future Video Synthesis with Object Motion Prediction. CVPR (2020) 1
32. Yang, G., Huang, X., Hao, Z., Liu, M.Y., Belongie, S.J., Hariharan, B.: Pointflow: 3d point cloud generation with continuous normalizing flows. 2019 IEEE/CVF International Conference on Computer Vision (ICCV) pp. 4540–4549 (2019) 4
33. Yuan, Y., Weng, X., Ou, Y., Kitani, K.: AgentFormer: Agent-Aware Transformers for Socio-Temporal Multi-Agent Forecasting. ICCV (2021) 1, 10
34. Zhang, C., Fiore, M., Murray, I., Patras, P.: CloudLSTM: A Recurrent Neural Model for Spatiotemporal Point-cloud Stream Forecasting. AAAI (2021) 4