电子科技大学
University of Electronic Science and Technology of China

UltraLiDAR

Learning **Compact Representations** for LiDAR Completion and Generation

Related Work

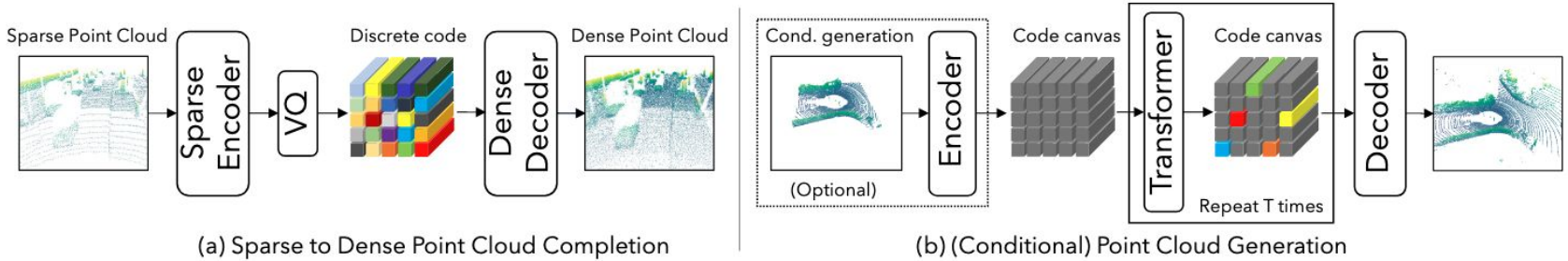
- VQ-VAE
  - learn discrete representations by compressing images into discrete latent space
- VQ-GAN
  - based on the learned codebook and decoder in VQ-VAE, make a use of transformer model for generation
- MaskGIT
  - use mask modeling with **bidirectional** transformers

Existing works focus on **2D natural images**

First work that conducts **discrete representation** in the 3D domain

Main Contribution

- Present a compact LiDAR representation that can effectively capture data priors
- Propose a sparse-to-dense LiDAR completion pipeline
- Develop an (un)conditional LiDAR generative model



Methodology

- Discrete Representations for LiDAR
- LiDAR Completion
- LiDAR Generation
  - Unconditional Generation
  - Conditional Generation
  - Free Space Suppression Sampling
  - Iterative Denoising

Discrete Representations for LiDAR

- VQ-VAE revisit
  - $\mathcal{L}_{\text{vq}} = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 + \|\text{sg}[E(\mathbf{x})] - \hat{\mathbf{z}}\|_2^2 + \|\text{sg}[\hat{\mathbf{z}}] - E(\mathbf{x})\|_2^2$
  - Limited number of discrete codes stabilizes the input distribution of the decoder, forces the codes to capture **meaningful**, **re-usable** information
- VQ-VAE for LiDAR
  - Directly applying VQ-VAE for LiDAR is challenging
  - **Voxelization**: voxelize the point clouds and instead infer whether **each voxel** is **occupied or not**
  - Ground the point clouds with a **pre-defined grid** where the size of each voxel is 15.625 × 15.625 × 15 cm for x, y, z dimensions
  - Convert the input to **Bird's-Eye View (BEV)** images: treat the height dimension of the voxel grid as feature channel C, and process 3D LiDAR data just like 2D images

Discrete Representations for LiDAR

Details of VQ-VAE for LiDAR:

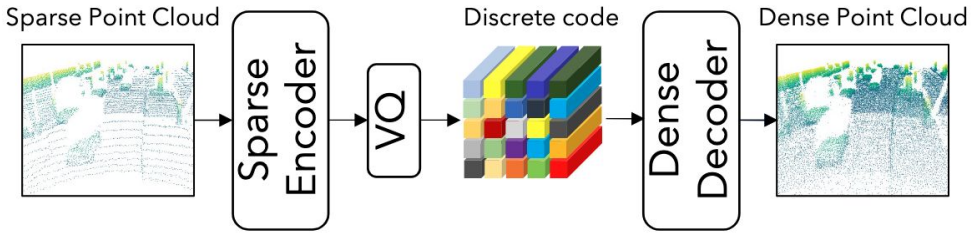
- Encoder E and Decoder G: both **Swin Transformer**s with 12 layers (8 heads and the embedding dimension is 512)
- Codebook size is 1024, with 1024 hidden dimensions for each code
- Output of decoder is a logit grid $\hat{\mathbf{x}} \in \mathbb{R}^{H \times W \times C}$
- Convert to binary voxel grid $\hat{\mathbf{x}}^{\text{bin}} \in \{0, 1\}^{H \times W \times C}$
- Replace the ℓ_2 reconstruction loss with a binary (occupied or not) cross-entropy loss $\mathcal{L}_{\text{vq}} = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 + \|\text{sg}[E(\mathbf{x})] - \hat{\mathbf{z}}\|_2^2 + \|\text{sg}[\hat{\mathbf{z}}] - E(\mathbf{x})\|_2^2$
- Adopt a pre-trained voxel-based detector V to compute perceptual loss

$$\mathcal{L}_{\text{feat}} = \mathcal{L}_{\text{vq}} + \|V_{\text{b}}(\mathbf{x}) - V_{\text{b}}(\hat{\mathbf{x}}^{\text{bin}})\|_2^2$$

V_{b} denotes the feature from the last backbone layer of V

LiDAR Completion

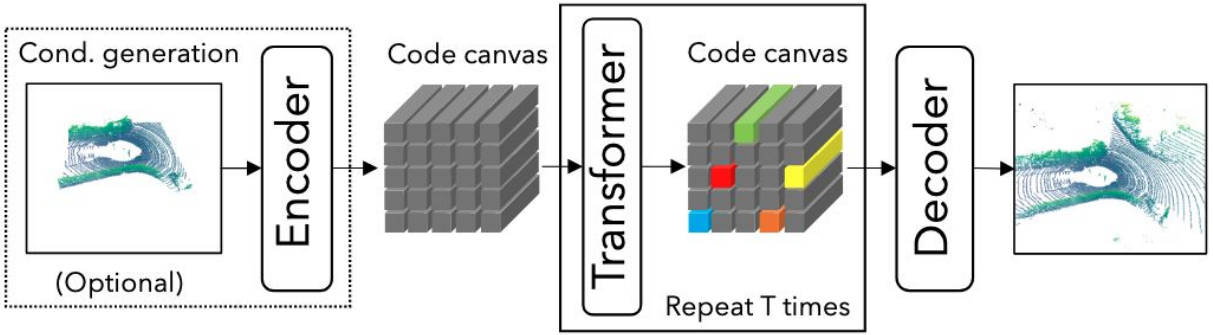
- Given a dataset of paired, voxelized LiDAR point clouds $\{(\mathbf{x}_1^{\text{sp}}, \mathbf{x}_1^{\text{den}}), ..., (\mathbf{x}_N^{\text{sp}}, \mathbf{x}_N^{\text{den}})\}$
- For dense LiDAR point clouds $\mathbf{x}^{\text{den}}$, learn a discrete codebook $\{\mathbf{e}_1^{\text{den}}, ..., \mathbf{e}_K^{\text{den}}\}$, an encoder E^{den} and a decoder G^{den}
- Learn a separate encoder E^{sp} to map each sparse LiDAR point cloud $\mathbf{x}^{\text{sp}}$ to the same feature space $\mathbf{z}^{\text{sp}} = E^{\text{sp}}(\mathbf{x}^{\text{sp}})$
- Quantise the sparse LiDAR point cloud with the dense discrete representation $\mathbf{e}^{\text{den}}$, $\hat{\mathbf{z}}^{\text{sp}} = q(\mathbf{z}^{\text{sp}})$
- Decode with the dense decoder $\hat{\mathbf{x}}^{\text{sp-den}} = G^{\text{den}}(\hat{\mathbf{z}}^{\text{sp}})$



(a) Sparse to Dense Point Cloud Completion

LiDAR Unconditional Generation

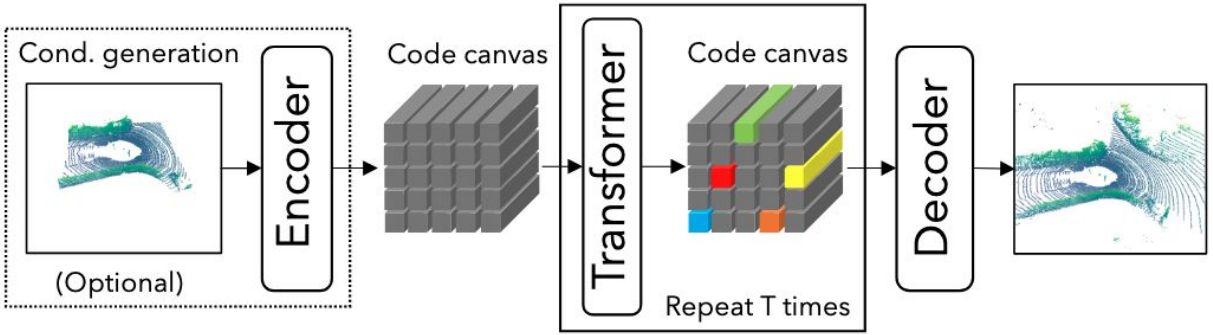
- Generate discrete code maps in the form of code indices
- Following MaskGIT, adopt a bi-directional self-attention Transformer to iteratively predict the code map
- For unconditional generation, start from a blank canvas
- At each iteration, select a subset of the predicted codes with top confidence scores and update the canvas accordingly



(b) (Conditional) Point Cloud Generation

LiDAR Conditional Generation

- For conditional generation, instead of start with a blank canvas, one can start with a partially filled code map
- For LiDAR manipulation, one can place special codes at ROI (Region of Interest), e.g. [CAR] codes, and run the model multiple times



(b) (Conditional) Point Cloud Generation

Free Space Suppression Sampling

- MaskGIT iterative procedure can be viewed as a **coarse-to-fine** generation
  - Codes generated during early iterations determin the overall structure
  - While the ones generated at the end are in charge of fine-grained details
- Lead to degenerated results when generating LiDAR point clouds
  - LiDAR point clouds are sparse, a large portion of the scene is represented by the same code: [BLANK]
  - Transformer tends to predict [BLANK] codes with high scores since they occur frequently
  - May fill most of the canvas with them, and little structure will remain
- Suppress the [BLANK] code during the early generation stages by setting their probability to 0
  - Identify the [BLANK] codes by looking at the occurrence statistic of all codes across the whole dataset, empirically select the top as [BLANK] codes.

Iterative Denoising

- Generated point clouds contain high-frequency noise
- Randomly **mask out** different regions of the output LiDAR point clouds and re-generate them
  - If the masked region is a structured region, Transformer can still recover it through the context
  - If the masked region is a pure noise, then it will likely be removed after multiple trials, since the model can’t infer it from the context

Experiment

- Dataset: KITTI-360
- Since UltraLiDAR generates points based on voxels, the number of points may differ from the real point cloud, points from the same voxel will only count once

Method	$\text{MMD}_{\text{BEV}} \downarrow$	$\text{JSD}_{\text{BEV}} \downarrow$
LiDAR VAE [3]	1.18×10^{-3}	0.256
LiDAR GAN [3]	2.07×10^{-3}	0.275
Projected GAN [38]	1.25×10^{-3}	0.190
LiDARGen [56]	4.80×10^{-4}	0.140
Ours	$\mathbf{9.67 \times 10^{-5}}$	**0.132**

Table 4. **Quantitative results on KITTI-360.** Our results show better statistical alignment with the real data.