Learning Compact Representations for LiDAR Completion and Generation

Yuwen Xiong[1,2] Wei-Chiu Ma[1,3] Jingkang Wang[1,2] Raquel Urtasun[1,2]
[1]Waabi [2]University of Toronto [3]Massachusetts Institute of Technology
{yxiong,wma,jwang,urtasun}@waabi.ai

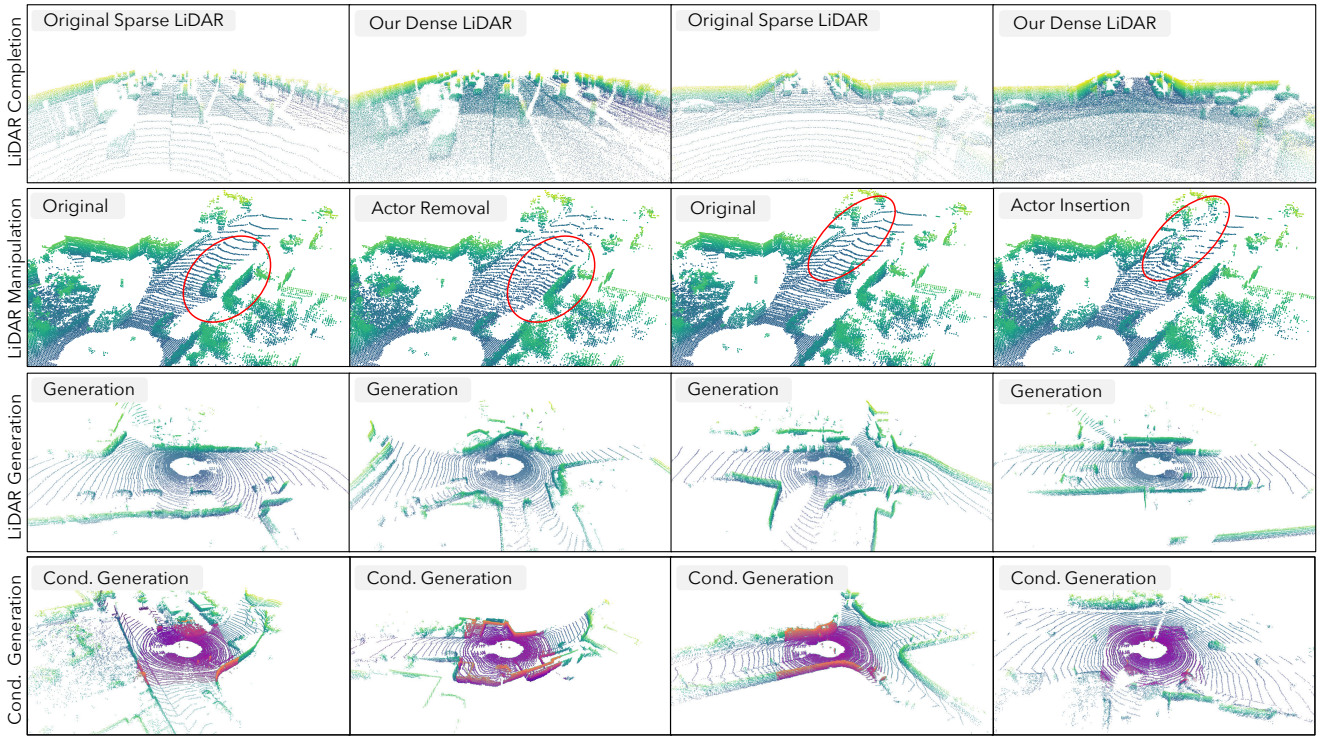

Figure 1. UltraLiDAR learns discrete representations from large-scale LiDAR point clouds and conduct realistic, scalable and controllable LiDAR completion and generation. **Top row:** Sparse-to-dense LiDAR completion; **Second row:** Controllable manipulation of real LiDAR with actor removal and insertion; **Third row:** Diverse LiDAR generation with realistic global structure and fine-grained details; **Bottom row:** Conditional scene generation with partially observed point clouds (highlighted in red). Please see supp. for more examples.

Abstract

LiDAR provides accurate geometric measurements of the 3D world. Unfortunately, dense LiDARs are very expensive and the point clouds captured by low-beam LiDAR are often sparse. To address these issues, we present UltraLiDAR, a data-driven framework for scene-level LiDAR completion, LiDAR generation, and LiDAR manipulation. The crux of UltraLiDAR is a compact, discrete representation that encodes the point cloud's geometric structure, is robust to noise, and is easy to manipulate. We show that by aligning the representation of a sparse point cloud to that of a dense point cloud, we can densify the sparse point clouds as if they were captured by a real high-density LiDAR, drastically reducing the cost. Furthermore, by learning a prior over the discrete codebook, we can generate diverse, realistic LiDAR point clouds for self-driving. We evaluate the effectiveness of UltraLiDAR on sparse-to-dense LiDAR completion and LiDAR generation. Experiments show that densifying real-world point clouds with our approach can significantly improve the performance of downstream perception systems. Compared to prior art on LiDAR generation, our approach generates much more realistic point clouds. According to A/B test, over 98.5% of the time human participants prefer our results over those of previous methods. Please refer to project page https://waabi.ai/research/ultralidar/ for more information.

1. Introduction

Building a robust 3D perception system is key to bringing self-driving vehicles into our daily lives. To effectively

perceive their surroundings, existing autonomous systems primarily exploit LiDAR as the major sensing modality, since it can capture well the 3D geometry of the world. However, while LiDAR provides accurate geometric measurements, it comes with two major limitations: (i) the captured point clouds are inherently sparse; and (ii) the data collection process is difficult to scale up.

Sparsity: Most popular self-driving LiDARs are time-of-flight and scan the environment by rotating emitter-detector pairs (*i.e.*, beams) around the azimuth. At every time step, each emitter emits a light pulse which travels until it hits a target, gets reflected, and is received by the detector. Distance is measured by calculating the time of travel. Due to the design, the captured point cloud density inherently decreases as the distance to the sensor increases. For distant objects, it is often that only a few LiDAR points are captured, which greatly increases the difficulty for 3D perception. The sparsity problem becomes even more severe under poor weather conditions [47], or when LiDAR sensors have fewer beams [4]. One "simple" strategy is to increase the number of LiDAR beams. However, 128-beam LiDAR sensors are much more expensive than their 64/32-beam counterparts, not to mention that 512-beam LiDAR does not exist yet.

Scalability: Training and testing perception systems in diverse situations are crucial for developing robust autonomous systems. However, due to their intricate design, LiDARs are much more expensive than cameras. A commercial 64-beam LiDAR usually costs over 25K USD. The price barrier makes LiDAR less accessible to the general public and restricts data collection to a small fraction of vehicles that populate our roads, significantly hindering scaling up. One way to circumvent the issue is to leverage existing LiDAR simulation suites to generate more data. While the simulated point clouds are realistic, these systems typically require one to manually create the scene or rely on multiple scans of the real world in advance, making such a solution less desirable.

With these challenges in mind, we propose UltraLiDAR, a novel framework for LiDAR completion and generation. The key idea is to learn a *compact, discrete* 3D representation (codebook) of LiDAR point clouds that encodes the geometric structure of the scene and the physical rules of our world (*e.g.*, occlusion). Then, by aligning the representation of a sparse point cloud to that of a dense point cloud, we can densify the sparse point cloud as if it were captured by a high-density LiDAR (*e.g.*, 512-beam LiDAR). Furthermore, we can learn a prior over the discrete codebook and generate novel, realistic driving scenes by sampling from it; we can also manipulate the discrete code of the scene and produce counterfactual scenarios, both of which can drastically improve the diversity and amount of LiDAR data. Fig. 1 shows some example outputs of UltraLiDAR.

We demonstrate the effectiveness of UltraLiDAR on two tasks: sparse-to-dense LiDAR completion and LiDAR generation. For LiDAR completion, since there is no ground truth for high-density LiDAR, we exploit the performance of downstream perception tasks as a "proxy" measurement. Specifically, we evaluate 3D detection models on both sparse and densified point clouds and measure the performance difference. Experiments on Pandaset [46] and KITTI-360 [27] show that our completion model can generalize across datasets and the densified results can significantly improve the performance of 3D detection models. As for LiDAR generation, we compare our results with state-of-the-art LiDAR generative models [3, 53]. Our generated point clouds better match the statistics of those of ground truth data. We also conducted a human study where participants prefer our method over prior art over 98.5% (best 100%) of the time; comparing to ground truth, our results were selected 32% of the time (best 50%).

To summarize, the main contributions of this paper are:

1. We present a LiDAR representation that can effectively capture data priors and enable various downstream applications, *e.g.*, LiDAR completion and generation.
2. We propose a sparse-to-dense LiDAR completion pipeline that can accurately densify sparse point clouds and improve the performance and generalization ability of trained detection models across datasets.
3. We develop an (un)conditional LiDAR generative model that can sythesize high-quality LiDAR point clouds and supports various manipulations.

2. Related Work

LiDAR / Depth Completion: Sparse LiDAR/depth completion is a beneficial task for many robotic applications as it can provide more accurate geometry for holistic scenes compared to images. LiDAR scene completion has been studied and attracted the computer vision community's attention in recent years [2,9,34,35,42,48], while the progress has been pushing forward, most of them focus on directly predicting semantic labels for the completed coarse voxels without fine-grained geometry information, thus no other downstream tasks are benefited. More recently, SPG [47] was proposed for unsupervised domain adaptation for 3D object detection via foreground point completion. However, its completion target is generated by heuristics, thus an improved holistic understanding of the scene is hard to achieve. Depth completion [43] instead solves the problem by relying on a sparse lidar sweep and a paired RGB image to compute a 2.5D pseudo-depth image. However, this problem is ill-posed as monocular images contain ambiguities and the field of view is also limited. Thus, the accuracy is hard to improve, and it's difficult to reconstruct whole scenes at scale. Our model instead tackles this problem

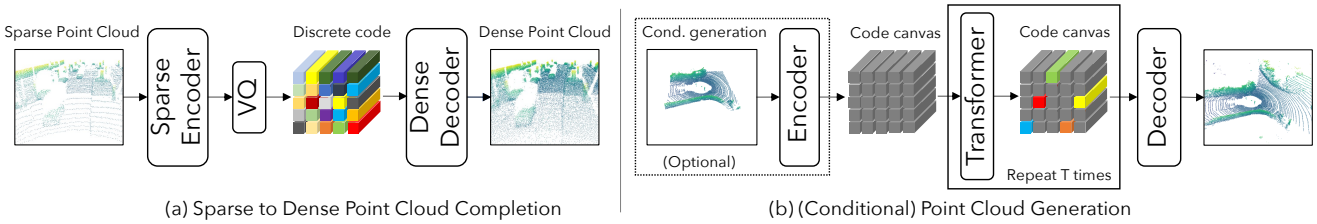

Figure 2. **Overview of UltraLiDAR pipeline.** For (a) LiDAR completion, the sparse encoder maps the sparse point cloud to discrete codes, and the dense decoder reconstructs dense data from them; For (b) LiDAR generation, the transformer model starts from a blank canvas or canvas with codes mapped from the partial observations; and iteratively predicts and updates the missing parts. The decoder produces the LiDAR output given the predicted code as the generation results.

from a representation learning perspective. It captures the data prior in advance with robust discrete representations thus generalizing better. It also completes the full scene with a higher-beam LiDAR pattern, which is generic to various downstream tasks.

3D Generation: Inspired by the tremendous progress in 2D image generation [11, 18, 25], 3D content generation has attracted more and more attention in recent years. Existing works demonstrated the high-quality generation in different representations including point cloud [1, 3, 50, 52], voxel grid [16, 21, 29, 41, 45], mesh [14, 17, 19, 39] or implicit geometry [5, 6, 38]. However, these works usually focus on the object level and cannot handle large scenes (due to non-compact representation, large memory footprint and limited model capacity). To enable larger-scale scene synthesis, recent works either define a more compact output space or procedurally generate the scenes [31, 32]. Unfortunately, the generated scenes are still quite limited (*e.g.*, indoor environments) and there are only few works focusing on large-scale scene generation for self-driving. In this paper, we focus on the generation of LiDAR point clouds that are highly unstructured, sparse, and non-uniform. [3] and [36] use the variational autoencoder (VAE) or generative adversarial network (GAN) on LiDAR point clouds but the generation realism is quite limited. Most recently, researchers propose a novel score-matching diffusion model for higher-quality LiDAR generation [53] but keep only limited local geometry details and global scenario structures. In contrast, we present a generic framework to learn discrete representations to better maintain the structure and semantic information for a more realistic and controllable generation.

Learning Discrete Representations: VQ-VAE [44] proposed to learn discrete representations by compressing images into discrete latent space. It primarily consists of three components: an encoder that learns to map the input image into latent feature maps; a codebook that contains a set of learnable latent embeddings where the feature maps are quantized via nearest neighbor lookup; and a decoder that reconstructs the input image from the quantized code. [33] used a multi-scale hierarchical architecture to capture local and global information in separate codebooks. Based on the learned codebook and decoder in VQ-VAE, VQ-GAN [13] proposed to use of a transformer model for image generation. More recently, MaskGIT [7] shows that the learned codebook and the decoder are capable of generating impressive realistic RGB images by using mask modeling with bi-directional transformers. However, all existing works focus on 2D natural images and it is non-trivial to handle sparse, unstructured LiDAR points. To our best knowledge, this paper is the first work that conducts discrete representation in the 3D domain and achieves state-of-the-art performance in large-scale LiDAR scene completion and generation.

3. UltraLiDAR

In this paper, we seek to learn a compact 3D representation of scene-level LiDAR point clouds to enable a series of downstream applications, such as sparse-to-dense LiDAR completion, (un)conditional LiDAR generation, and LiDAR manipulation. Based on the observation that vector quantized (VQ) representations [13, 33, 44] are robust to noise, easy to manipulate, and naturally compatible with generative models, we propose to learn a discrete codebook for LiDAR point clouds and build our model on top of it.

We start by reviewing the basics of vector-quantized variational autoencoder (VQ-VAE) [44], a building block of our approach. Then we showcase how to exploit similar concepts to encode 3D point clouds into a discrete codebook. Finally, we discuss how to exploit the discrete representation for different tasks.

3.1. Discrete Representations for LiDAR

VQ-VAE revisit: The goal of VQ-VAE is to learn a discrete latent representation that is expressive, robust to noise, and compatible with generative models. VQ-VAE consists of three parts: (i) an encoder E that encodes the input signal, which for simplicity we assume to be an image, $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$ to a continuous embedding map $\mathbf{z} = E(\mathbf{x}) \in \mathbb{R}^{h \times w \times D}$, (ii) an element-wise quantization function q that maps each embedding to its closest learnable latent code $\mathbf{e}_k \in \mathbb{R}^D$, with $k = 1, ..., K$, and (iii) a decoder G that takes as input the quantized representation $\hat{\mathbf{z}} = q(\mathbf{z})$ and outputs the reconstructed image $\hat{\mathbf{x}} = G(\hat{\mathbf{z}})$. The whole model can be trained end-to-end by minimizing:

$$\mathcal{L}_{\text{vq}} = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 + \|\text{sg}[E(\mathbf{x})] - \hat{\mathbf{z}}\|_2^2 + \|\text{sg}[\hat{\mathbf{z}}] - E(\mathbf{x})\|_2^2, \quad (1)$$

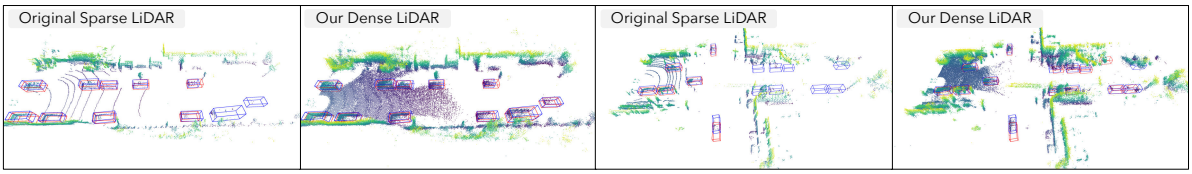

Figure 3. **LiDAR completion and 3D detection on Pandaset.** With densified point clouds, the detection model can identify more objects and reduce false negatives. We show detection results in red boxes and ground truth in blue. The missing area far away from ego vehicle in the densified results is caused by uphill; we refer the reader to the supp. material for the explanation with camera visualization.

where sg[·] denotes the stop-gradient operation.

The limited number of discrete codes $\mathbf{e}$ stabilizes the input distribution of the decoder during training; it also forces the codes to capture meaningful, re-usable information as the decoder can no longer "seek shortcut" from the continuous signals for the reconstruction task. VQ-VAE has enjoyed great success across different modalities, such as natural images [7, 13], audio [44] and 2.5D images [40]. In this work, we further extend it to learn discrete representations for 3D LiDAR point clouds.

VQ-VAE for LiDAR: We aim to learn a discrete codebook that can effectively represent a set of LiDAR point clouds. Directly applying VQ-VAE, however, is challenging, since the fixed set of discrete latents will have to model point clouds that live in a continuous 3D space, and deal with the fact that each point cloud may have a different number of points. To address these issues, we propose to voxelize the point clouds and instead infer whether each voxel is occupied or not. By grounding the point clouds with a pre-defined grid (similar to the 2D pixel grid of images), we can foster the discrete codebook to learn the overall structure rather than the minor 3D positional variations. This representation also can naturally handle the varying number of points. While we may sacrifice some precision during the voxelization process, the impact is negligible for both LiDAR completion and generation (see Sec. 4).

Now that we have a voxelized point cloud, the next step is to design the encoder E and the decoder G. For large scenes with high resolution, 3D convolution becomes very expensive since we need to infer the occupancy of each voxel densely. We thus convert the input to Birds-Eye-View (BEV) images by treating the height dimension of the voxel grids as feature channel C and then adopt 2D convolutions instead. In this case, we can process 3D LiDAR data just like 2D images; we can also exploit existing model architectures designed for 2D images directly. We note that such BEV images have been widely adopted in the context of self-driving perception [8, 51], since they encode rich geometric information. The output of the decoder is a logit grid $\hat{\mathbf{x}} \in \mathbb{R}^{H \times W \times C}$. It can be further converted to a binary voxel grid $\hat{\mathbf{x}}^{\text{bin}} \in \{0, 1\}^{H \times W \times C}$ through gumbel softmax [23].

Finally, we train our LiDAR VQ-VAE model with Eq. 1, except that we replace the ℓ_2 reconstruction loss with a binary (occupied or not) cross-entropy loss. To improve the realism of the reconstruction, we further adopt a pre-trained voxel-based detector V and measure the feature difference, similar to perpectual loss [24]. Our full loss is:

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

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

LiDAR manipulation: Once we train the model, we can easily manipulate arbitrary LiDAR point clouds by editing their corresponding latent codes. Since objects are spatially apart in 3D, the model can dedicate specific codes for them. We can thus identify the codes for objects of interest (*e.g.*, vehicles) and insert/remove them into/from the scene. As shown in Sec. 4, we can populate many vehicles on the street and create counterfactual scenarios.

3.2. LiDAR Completion

Given a dataset of paired, voxelized LiDAR point clouds $\{(\mathbf{x}_1^{\text{sp}}, \mathbf{x}_1^{\text{den}}), ..., (\mathbf{x}_N^{\text{sp}}, \mathbf{x}_N^{\text{den}})\}$, the goal of LiDAR completion is to learn a function f that maps a sparse LiDAR point cloud $\mathbf{x}^{\text{sp}}$ to its dense counterpart $\mathbf{x}^{\text{den}}$.

One straightforward strategy is to leverage the pairwise supervision and directly learn a pix2pix-style network [22] but on voxels. While this model performs well within the same data distribution, it degrades significantly when the input distribution shifts (*e.g.*, when the sparse LiDAR point clouds come from different datasets). We conjecture that this is because there is no restriction on the learned representation, and the model learns to "cheat" by relying on non-generalizable details rather than the overall structure.

In this section, we investigate the use of discrete representations to alleviate the issue. We first learn a discrete codebook $\{\mathbf{e}_1^{\text{den}}, ..., \mathbf{e}_K^{\text{den}}\}$, an encoder E^{den}, and a decoder G^{den} for each dense LiDAR point cloud $\mathbf{x}^{\text{den}}$ using the approach in Sec. 3.1. Then we learn a separate encoder E^{sp} that maps each sparse LiDAR point cloud $\mathbf{x}^{\text{sp}}$ to the same feature space $\mathbf{z}^{\text{sp}} = E^{\text{sp}}(\mathbf{x}^{\text{sp}})$ and quantize it with the dense discrete representation $\mathbf{e}^{\text{den}}$. Finally, we decode the quantized representation $\hat{\mathbf{z}}^{\text{sp}} = q(\mathbf{z}^{\text{sp}})$ with the dense decoder G^{den} and obtain a densified point cloud $\hat{\mathbf{x}}^{\text{sp-den}} = G^{\text{den}}(\hat{\mathbf{z}}^{\text{sp}})$. With the help of the learned codebook, we can ensure the input to the dense decoder G^{den} is in-domain and the reconstructed point clouds are high-quality; we can also denoise the embedding from the encoder and better handle the variations and distribution shift within the input data.

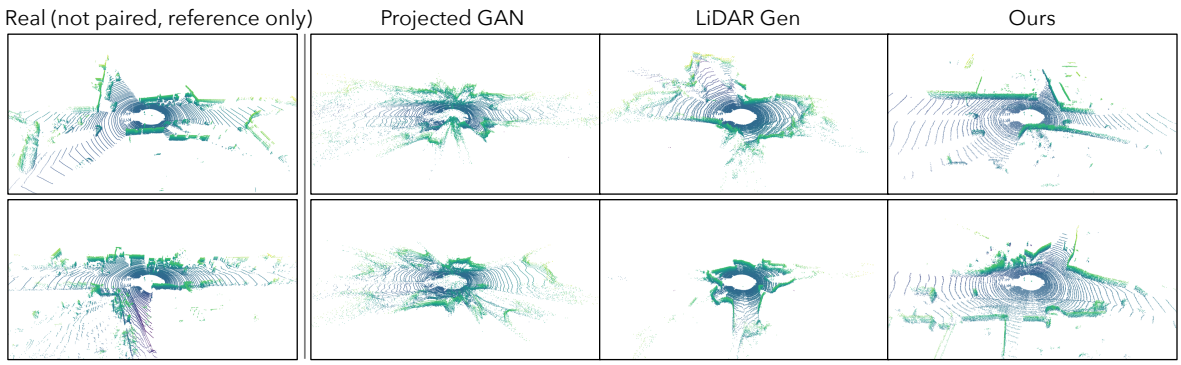

Figure 4. **Qualitative comparison against baselines on unconditional LiDAR generation.** We compare with two state-of-the-art LiDAR generation methods Projected GAN [37] and LiDARGen [53] and include real data for reference. Our model can generate results with more structured layouts and clearer beam patterns.

Training: In practice, we jointly train the sparse encoder E_{sp} and the dense VQ-VAE model, since it achieves slightly better performance than performing two-stage training. We hypothesize this is because joint training allows the model to learn a codebook that is easy to decode, and achieves low quantization error for both encoders E_{sp} and E_{den}. We use the same loss functions in Sec. 3.1 to train the whole model, except that the reconstructed target is always the dense point cloud (which we can obtain from the paired data).

3.3. LiDAR Generation

As we have alluded to earlier, the learned discrete representations can be naturally combined with generative models. In this section, we describe in detail how we exploit the discrete codebook to generate high-fidelity LiDAR point clouds, both unconditionally and conditionally.

Unconditional generation: Given the learned codebook $\mathbf{e}$ and the decoder G, the problem of LiDAR generation can be formulated as code map generation. Instead of directly generating LiDAR point clouds, we first generate discrete code maps in the form of code indices. Then we map the indices to discrete features by querying the codebook and decoding them back to LiDAR point clouds with the decoder. Following Chang *et al*. [7], we adopt a bi-directional self-attention Transformer [7, 20] to *iteratively* predict the code map. Specifically, we start from a blank canvas. At each iteration, we select a subset of the predicted codes with top confidence scores and update the canvas accordingly. With the help of the Transformer, we can aggregate context from the whole map and predict missing parts based on already predicted codes. In the end, the canvas will be filled with predicted code indices, from which we can decode LiDAR point clouds. We refer the reader to [7] for more details.

Conditional generation: Our unconditional LiDAR generation pipeline can be easily extended to perform conditional generation. Instead of starting the generation process from an empty canvas, one can simply start with a partially filled code map and let the Transformer predict the rest. For instance, we can place `[CAR]` codes at regions of interest; and run the model multiple times. We can then obtain different traffic scenarios with the pre-defined cars untouched. Please refer to supp. material for how we identify the codes.

Free space suppression sampling: Our iterative generation procedure can be viewed as a variant of coarse-to-fine generation. The codes generated during early iterations determine the overall structure, while the ones generated at the end are in charge of fine-grained details. While this pipeline is effective for image generation [7], it may lead to degenerated results when generating LiDAR point clouds. One critical reason is that LiDAR point clouds are sparse, and a large portion of the scene is represented by the same `[BLANK]` codes. Since the `[BLANK]` codes occur frequently, the Transformer tends to predict them with high scores. If we naively sample the codes based on the output of the Transformer, we may fill most of the canvas with `[BLANK]` codes, and little structure will remain. To address this issue, we suppress the `[BLANK]` codes during the early generation stages by setting their probability to 0. This ensures the model generates meaningful structures in the beginning. We identify the `[BLANK]` codes by looking at the occurrence statistic of all codes across the whole dataset. We empirically select the top as `[BLANK]` codes.

Iterative denoising: With free space suppression sampling, we can already obtain good results. However, the generated point clouds sometimes still contain high-frequency noise (*e.g*., there might be some floating points in the very far range). To mitigate this issue, we randomly mask out different regions of the output LiDAR point clouds and re-generate them. The intuition is that if we mask out a structured region, we can still recover it through the neighborhood context. However, if the masked region corresponds to pure noise that is irrelevant to the surrounding, it will likely be removed after multiple trials (since the model cannot infer it from the context).

Training: We first encode all LiDAR point clouds into frozen discrete representations (code maps) learned in Sec. 3.1. Then, at each training iteration, we randomly mask out a subset of codes. Finally, we adopt the bi-directional

Model	Sparse to Dense	Two-stage PIXOR		PointPillar	
		AP_{BEV}	AP_{3D}	AP_{BEV}	AP_{3D}
Real / 64	-	79.3	62.2	75.5	62.3
Sim / 512	-	78.1	57.7	70.0	55.5
Sim / 512	ContComp	79.7	62.4	75.1	59.8
Sim / 512	Ours	**80.3**	**64.3**	**76.0**	**62.8**

Table 1. **Results on PandaSet.** We evaluate all models on PandaSet real 64-beam data, and perform sparse-to-dense during inference optionally with ContComp and our model.

Transformer to predict the correct code for those masked regions. Since we have GT code map, we supervise the model with cross-entropy loss. See [7] for more details.

3.4. Implementation Details

In this section, we discuss implementation details that are crucial for learning discrete LiDAR representations.

Voxel sizes matter: We set the input voxel size to be $15.625 \times 15.625 \times 15$ cm for x, y, z dimensions. We find that the downsampling ratio when mapping the BEV image to the discrete code has a huge impact on both reconstruction and generation performance. If the patch size that each discrete code represents is too large, a single fixed code does not have enough capacity to model the variations (e.g., a small position/rotation shift of a car inside the patch). We empirically find that downsampling $8\times$ achieves a good trade-off between preserving high-frequency information in the LiDAR data and maintaining high-level semantic meaning. Thus the patch size is 8×8, leading to 1.25×1.25 m on the spatial dimension that each code represents.

Model hyperparameters: As a typical BEV image usually has a large spatial range, the resolution of the image and the number of codes are high. We use Swin Transformer [28] instead of the vanilla Vision Transformer [12] to reduce the computational cost for our generative Transformer model. It has 24 layers and 8 heads, and the embedding dimension is set to 512. All other training hyperparameters like optimizer settings and label smoothing are kept the same as in [7]. For simplicity, we use the same architecture in our VQ-VAE learning. The encoder and decoder are both Swin Transformers with 12 layers, respectively. We set the codebook size to 1024 with 1024 hidden dimensions for each code.

Codebook (re)-initialization: We empirically find that the codebook can easily collapse (only a few codes are used) during training. For better codebook learning, we use data-dependent codebook initialization. Specifically, we use a memory bank to store the continuous embedding output from the encoders at each iteration; and use K-Means centroids of the memory bank to initialize/reinitialize the codebook if the code utilization percentage is lower than a threshold (empirically, we define a code is not used for 256 iterations as "dead code" and set the threshold to be 50%). During the first 2000 iterations of training, we gradually shifted the decoder input from continuous to quantized embeddings as a warmup. We find these strategies help achieve good codebook learning and utilization.

Model	Sparse to Dense	Two-stage PIXOR		PointPillar	
		AP_{BEV}	AP_{3D}	AP_{BEV}	AP_{3D}
Real / 64	-	71.7	32.8	60.9	28.1
Sim / 512	-	66.9	33.2	58.5	28.0
Sim / 512	ContComp	74.9	41.5	67.7	36.9
Sim / 512	Ours	**76.7**	**46.3**	**73.0**	**40.9**

Table 2. **PandaSet $\rightarrow$ KITTI Cross-Dataset Evaluation.** The detection and LiDAR completion models are trained on PandaSet, and no KITTI data is seen/used before evaluation. Our LiDAR completion model can do reliable completion on unseen data and significantly improve the detector performance in zero-shot.

4. Experiments

4.1. Perception with LiDAR Completion

We first conduct experiments to verify that UltraLiDAR can boost the performance of a 3D detection model performance by performing LiDAR completion and using the densified results as detection model input. To show that our model generalizes well and can generate reliable sparse-to-dense results across datasets with the learned discrete representations, we further test the models on KITTI [15] that are trained on PandaSet [46]. We also evaluate our model on the SemanticKITTI scene completion benchmark [2] and show that our model with a generic design can be on par with models on the leaderboard that are heavily tuned for this task.

Dataset: We train UltraLiDAR and the 3D detection models on PandaSet [46]. We reimplement LiDARsim [30] to generate the 512-beam LiDAR data paired with the real 64-beam LiDAR sweeps (*i.e.*, the actor placement and background are identical) and obtain the sparse-to-dense supervision. We use a custom train/validation split since there is no official split; we refer the readers to the supp. materials for more details. We use KITTI [15] dataset for cross-dataset generalization evaluation; the validation set is used to obtain results, and no model is trained on it. For SemanticKITTI, we use the official train and test split.

Evaluation metrics: To evaluate 3D detection we use the KITTI metrics and report AP_{BEV} and AP_{3D} of the car category at IoU = 0.7. For the SemanticKITTI scene completion benchmark, we report Intersection-over-Union (IoU), which classifies a voxel as occupied or empty.

Training and implementation details: We use two base detectors for the 3D detection benchmark: PointPillar [26] from mmdetection3d [10], and a BEV detector which we denote Two-stage PIXOR, which improves over [49] by adding a second stage RoI refinement branch; more details of Two-stage PIXOR can be found in the supp. materials. We also modified PointPillar to make it use binary voxels

	SCCNet [42]	JS3C-Net [48]	Local-DIFs [34]	Ours
IoU	29.8	56.6	57.7	56.0

Table 3. **SemanticKITTI semantic scene completion results.** Our model with generic design can achieve on-par performance with state-of-the-art methods that employ architectures/modules heavily tuned for this task.

instead of pillar features as input; so that the LiDAR completion results from our model can be used. The voxel size for detectors is set to be the same as UltraLiDAR. For SemanticKITTI, the input data is preprocessed as voxel data, and we directly use it without modification.

Baselines: We first train detectors on PandaSet real data with 64-beam and simulated data with 512-beam, denoted as Real/64 and Sim/512, respectively; during inference, we perform sparse-to-dense LiDAR completion to densify the sparse real data for the Sim/512 model. We also train a LiDAR completion model without the codebook and quantization function (the model thus behaves like an AutoEncoder with sparse-to-dense supervision) as another baseline, denoted as ContComp. We evaluate all models with PandaSet real data; when testing the generalization ability on KITTI dataset, we directly evaluate all models trained on PandaSet in zero-shot (*i.e.*, no KITTI data is seen/used before the evaluation) with KITTI real data.

PandaSet results: From the first two rows in Tab. 1, we can see that the Sim/512 models perform worse than their Real/64 counterparts when evaluating on the real data. This is expected as there is a domain gap between the dense and sparse data, and the models may learn to rely on the richer geometry information in the dense data that does not exist in the sparse counterpart. However, if we perform the sparse-to-dense operation and lift the model to a higher density, we can see a decent performance improvement over the Sim/512 model. Moreover, we see that our model performs better than the ContComp model that uses continuous representations, indicating the effectiveness of using discrete representation in this task.

Cross-dataset genearlization results: To verify the generalization ability of our model, we directly test it on KITTI in a *zero-shot* manner, that is without seeing KITTI data in advance. As shown in Tab. 2, both entries that utilize sparse-to-dense achieve better performance, and even surpass the Real/64 model, showing the benefits of sparse-to-dense completion. Since the discrete representation is naturally more robust to noise because of the quantization operation, our model with discrete representations can outperform ContComp by a very significant margin when doing zero-shot generalization, once again showing the benefits of using discrete representations.

SemanticKITTI results: We further test our model on SemanticKITTI scene completion benchmark. As shown in Tab. 3, our model can achieve on-par performance with state-of-the-art methods [34, 35, 48] that employ architectures and operations heavily tuned for this task, for example shape-aware point-voxel interaction [48] or multi-resolution scene representation [34, 35].

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 [37]	1.25×10^{-3}	0.190
LiDARGen [53]	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.

4.2. LiDAR Scene Generation

We first compare quantitative results on KITTI-360 [27] with other baselines in the unconditional generation setting. In this case, we directly train our model with the sparse-to-sparse reconstruction task without the dense encoder; and use the sparse encoder to generate code maps in the generative model training instead. We follow LiDARGen [53] and use the first two sequences as the validation set and use the rest for model training. In Fig. 4[1], we show the qualitative comparison between our model and two baselines; and include the real data for reference. We can see that our model can generate results that are much more similar to the real data, with more structured and reasonable scene layouts and more stable/clearer beam patterns. More unconditional generation results on KITTI-360 can be found in the second row of Fig. 1 and supp. materials.

Quantitative results: Following [53], we use the Maximum-Mean Discrepancy (MMD) and Jensen–Shannon divergence (JSD) with a 100×100 2D histogram along the ground plane (x and y coordinates) as metrics. Since our model generates points based on voxels, the number of points may differ from the real point cloud. We thus use occupancy as a measurement when doing histogram bin count (*i.e.*, points from the same voxel will only count once) for point clouds from real data and other baselines. We believe this change captures the global structure difference better and lowers the weight on the local point density estimation, which is more reasonable and aligns better with the perceptual quality. As shown in Tab. 4, our method achieves superior performance compared with the baselines.

Model parameters: We calculate the number of parameters to make sure the model capacities are at the same level when compared with baselines. The number of parameters of LiDARGen [53] and our model are 29.7M and 40.3M, which are both smaller than a standard Swin-Small model.

[1] We contacted the authors of [53] to obtain outputs of all models they used for visualization and metrics calculation

Method	Percent Prefer Ours
Ours vs LiDAR VAE [3]	99.5%
Ours vs LiDAR GAN [3]	100%
Ours vs ProjGAN [37]	100%
Ours vs LiDARGen [53]	98.5%

Table 5. **Human study results on KITTI-360.** Results from our model show significantly better visual quality.

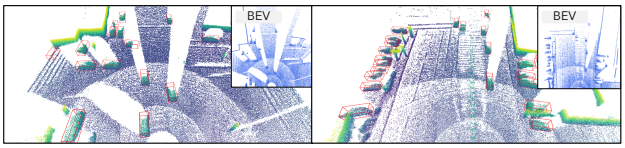

Figure 5. **Unconditional generation results on Pandaset.** We train our model on dense Pandaset data and generate dense results. The generated samples show diverse scenario layouts with proper actor placement (*e.g.*, parked car in the right sample). The synthesized point clouds are realistic such that a pre-trained detector can directly work out-of-the-box.

Human study: We perform an A/B test on a set of 8 researchers who have LiDAR expertise to better evaluate the visual quality of the generated samples. We use the same test system as [53], which shows a pair of randomly chosen images of two point clouds each time and lets the human decide which one is more realistic. We compare with four baselines as well as with the real data, with 200 image pairs each, leading to 1000 image pairs in total. We show the percentage of examples where participants believed our generations are more realistic against other baselines in Tab. 5. It is clear that our model can generate results with superior visual quality; over 98.5% of the time (100% for some baselines), the testers prefer our results over the baselines. It is worth noting that in 32% cases, people believe our results are *more realistic than real data*, which is very significant given the fact that for data that is indistinguishable from real, the winning ratio would be 50% with random choice.

Unconditional generation: Besides the KITTI-360 results in Fig. 4 and 1, we also train our model on dense Panadset for dense point cloud generation; and additionally run detection models on the generated samples, shown in Fig. 5. We can see that our model is able to generate diverse scenes with proper actor placement (*e.g.*, parked car in the right example), indicating the superiority of our model for this LiDAR generation task. Moreover, the excellent detection results showcase that the generated samples also have high fidelity w.r.t. the perception model.

Conditional generation: We show conditional generation results on KITTI-360 in the bottom row of Fig. 1. Our model can fully exploit the visible part(colored by purple) as context, do reasonable extrapolation for the surrounding environment, and generate diverse scenes (*e.g.*, curved road or crossroad) that align with the input condition well. See supp. materials for more results on KITTI-360/PandaSet.

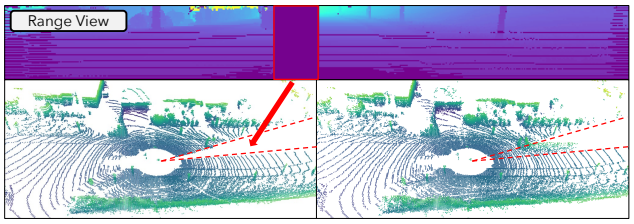

Figure 6. **Conditional LiDAR generation for dirt removal.** We mask the red rectangular region in the range view image to mimic dirt occluder. **Left**: Original input. The masked region is not visible to the model. **Right:** Our generation results. Our model can successfully recover the vehicle that is partially observed.

Meanwhile, we further consider a practical setting where the LiDAR sensor can fail for a specific region due to dirt or mechanical issues. To mimic this situation, we mask a part of the range images, as shown in Fig. 6. We can see that our model can still do accurate completion even on partially visible cars, recovering the occluded region, and potentially avoiding dangerous situations.

Manipulation: We show manipulation results with actor insertion and removal in the second row of Fig. 1. This is achieved by explicitly changing the code indices on the code map and letting the decoder generate new results. For example, we can copy the codes for the ground plane and paste them to overwrite the region where a car exists, resulting in a controllable manipulation process. We refer the readers for more results in supp. materials.

5. Conclusion

In this paper, we propose a framework to learn novel LiDAR representations that effectively capture the data priors that enable various downstream applications, including scene completion and generation. Based on the learned representation, we propose a novel sparse-to-dense data reconstruction pipeline that can accurately densify the sparse input and improve the generalization ability of the trained detection models in a zero-shot manner. We further show that our model can generate high-quality LiDAR point clouds unconditionally and do reasonable conditional generation and manipulation, which leads to more realistic and controllable data-driven LiDAR simulation.

One potential future direction for improving our model is to exploit the structure in range view representations, including beam info (which point from which ray) and occlusion reasoning, and combine them with our existing model. This may potentially improve the fidelity of our results.

Acknowledgement We thank Vlas Zyrianov and Shenlong Wang for providing A/B test code and baseline results, Arnaud Bonnet for infrastructure support, Siva Manivasagam for proofreading. We also thank the Waabi team for their valuable support. YX is partially supported by an NSERC scholarship and a Borealis AI fellowship. WCM is partially supported by a Siebel scholarship.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018. 3

[2] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9297–9307, 2019. 2, 6

[3] Lucas Caccia, Herke Van Hoof, Aaron Courville, and Joelle Pineau. Deep generative modeling of lidar data. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5034–5040. IEEE, 2019. 2, 3, 7, 8

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

[5] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16123–16133, 2022. 3

[6] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5799–5809, 2021. 3

[7] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11315–11325, 2022. 3, 4, 5, 6

[8] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, 2017. 4

[9] Ran Cheng, Christopher Agia, Yuan Ren, Xinhai Li, and Liu Bingbing. S3cnet: A sparse semantic scene completion network for lidar point clouds. *arXiv preprint arXiv:2012.09242*, 2020. 2

[10] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. https://github.com/open-mmlab/mmdetection3d, 2020. 6

[11] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021. 3

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 6

[13] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021. 3, 4

[14] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. *arXiv preprint arXiv:2209.11163*, 2022. 3

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. 6

[16] Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *European Conference on Computer Vision*, pages 484–499. Springer, 2016. 3

[17] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9785–9795, 2019. 3

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 3

[19] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018. 3

[20] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022. 5

[21] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Escaping plato's cave: 3d shape from adversarial rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9984–9993, 2019. 3

[22] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 4

[23] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv*, 2016. 4

[24] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 4

[25] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 3

[26] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019. 6

[27] Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *arXiv*, 2021. 2, 7

[28] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021. 6

[29] Sebastian Lunz, Yingzhen Li, Andrew Fitzgibbon, and Nate Kushman. Inverse graphics gan: Learning to generate 3d shapes from unstructured 2d data. *arXiv preprint arXiv:2002.12674*, 2020. 3

[30] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11167–11176, 2020. 6

[31] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *European Conference on Computer Vision*, pages 162–177. Springer, 2020. 3

[32] Chi-Han Peng, Yong-Liang Yang, and Peter Wonka. Computing layouts with deformable templates. *ACM Transactions on Graphics (TOG)*, 33(4):1–11, 2014. 3

[33] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019. 3

[34] Christoph Rist, David Emmerichs, Markus Enzweiler, and Dariu Gavrila. Semantic scene completion using local deep implicit functions on lidar data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 2, 7

[35] Luis Roldao, Raoul de Charette, and Anne Verroust-Blondet. Lmscnet: Lightweight multiscale 3d semantic completion. In *2020 International Conference on 3D Vision (3DV)*, pages 111–119. IEEE, 2020. 2, 7

[36] Ahmad El Sallab, Ibrahim Sobh, Mohamed Zahran, and Nader Essam. Lidar sensor modeling and data augmentation with gans for autonomous driving. *arXiv preprint arXiv:1905.07290*, 2019. 3

[37] Axel Sauer, Kashyap Chitta, Jens Müller, and Andreas Geiger. Projected gans converge faster. *Advances in Neural Information Processing Systems*, 34:17480–17492, 2021. 5, 7, 8

[38] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *Advances in Neural Information Processing Systems*, 33:20154–20166, 2020. 3

[39] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems*, 34:6087–6101, 2021. 3

[40] Yuan Shen, Wei-Chiu Ma, and Shenlong Wang. SGAM: Building a virtual 3d world through simultaneous generation and mapping. In *NeurIPS*, 2022. 4

[41] Edward J Smith and David Meger. Improved adversarial systems for 3d object generation and reconstruction. In *Conference on Robot Learning*, pages 87–96. PMLR, 2017. 3

[42] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1746–1754, 2017. 2, 7

[43] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. In *International Conference on 3D Vision (3DV)*, 2017. 2

[44] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. 3, 4

[45] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems*, 29, 2016. 3

[46] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3095–3101. IEEE, 2021. 2, 6

[47] Qiangeng Xu, Yin Zhou, Weiyue Wang, Charles R Qi, and Dragomir Anguelov. Spg: Unsupervised domain adaptation for 3d object detection via semantic point generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15446–15456, 2021. 2

[48] Xu Yan, Jiantao Gao, Jie Li, Ruimao Zhang, Zhen Li, Rui Huang, and Shuguang Cui. Sparse single sweep lidar point cloud segmentation via learning contextual shape priors from scene completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3101–3109, 2021. 2, 7

[49] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018. 6

[50] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4541–4550, 2019. 3

[51] Chris Zhang, Wenjie Luo, and Raquel Urtasun. Efficient convolutions for real-time semantic segmentation of 3d point clouds. In *3DV*, 2018. 4

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

[53] Vlas Zyrianov, Xiyue Zhu, and Shenlong Wang. Learning to generate realistic lidar point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*, October 2022. 2, 3, 5, 7, 8