Supplementary of 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

1. Code Identification

For street scenes in Birds-Eye-View (BEV), we empirically observe that the semantic meaning of the codes aligned well with the actors spatially. For example, we can refer object (*e.g.*, car) location from ground-truth labels or visual check and find the corresponding codes on the code map. The meaning of the codes can then be identified for controllable generation and manipulation. In this paper, we mainly leverage this property to perform scene manipulation by using a copy-paste mechanism for existing code and find it works properly. How to find the meaning of the codes for background/unlabeled effectively is still an open question. We leave this question and more potential use cases for the identified codes as future works.

2. Experimental Details

In this section, we discuss the experimental details for the results presented in the main paper.

PanadSet: PandaSet is a self-driving dataset introduced in recent work [6]. It contains 103 driving sequences captured in San Francisco Bay area with 8 seconds (80 frames, sampled at 10Hz) each. We split it into 73 and 30 sequences as training and validation sets by considering both geographic locations and time (so that data collected at different locations/timespan are in different sets). Specifically, we select the sequences $13-14, 57-79, 86-94$ and 149 as the validation set and put the rest into the training set. We set the region of interest for the point cloud to $[0, 80] \times [-40, 40]$ meters.

Two-stage PIXOR: Two-stage PIXOR can be seen as a variant of the original PIXOR proposed by Yang *et al*. [7]. We made the following modifications to improve the performance to be comparable with the state-of-the-art 3D detectors:

1. We use multi-scale deformable self-attention [8] instead of the upsampling deconvolution layers after the ResNet backbone to aggregate information from different scales and enhance the feature extraction; and output the feature maps with $1/4$ resolution the same as PIXOR.

2. We use the output from the dense detector header as the first-stage results. The top 500 bounding boxes after NMS are used as region proposals for the second stage. The 2D IoU threshold for NMS is set to 0.7

3. We use RotatedRoIAlign [5] to extract 3×3 RoI features from the feature map in the second-stage header; and apply two self-attention layers on features within each RoI and features between each RoI, respectively. Two MLPs are used to predict the classification score and box refinement for the region proposals.

4. We use DETR-like set-based loss with bipartite matching for both stages. An additional IoU loss is used for bounding box regression besides the smooth L1 loss.

LiDAR Simulation: We re-implement the state-of-the-art LiDAR simulation approach LiDARsim [2] for the generation of sim data with 512 laser beams. We only focus on the physics based simulation component based on OptiX ray tracing engine [3] and skip the ML ray-drop network.

We first create the asset bank with log-wise surfel aggregation following [2]. For background assets, we aggregate the points across all the frames in one snippet and remove all actors using 3D bounding box annotations. For dynamic actors, we aggregate the LiDAR points inside the bounding boxes in the object-centric coordinate. We then estimate per-point normals from 200 nearest neighbors with a radius of $20cm$ and orient the normals upwards for flat ground reconstruction. We downsample the LiDAR points into 4cm voxels and create per-point triangle faces (radius $5cm$) according to the estimated normals. Given the asset bank, we place the background and all actors in its original locations and transform the scene to the LiDAR coordinate for ray-triangle intersection computation. For Pandaset, we set the sensor intrinsics (*e.g.*, beam angle, azimuth resolution, etc) based on the public documents and

Method	$\text{MMD}_{\text{BEV}} \downarrow$	$\text{JSD}_{\text{BEV}} \downarrow$
LiDAR GAN [1]	3.06×10^{-3}	-
LiDAR VAE [1]	1.00×10^{-3}	0.161
Projected GAN [4]	3.47×10^{-4}	0.085
LiDARGen [9]	3.87×10^{-4}	**0.067**
Ours	$\mathbf{1.96 \times 10^{-4}}$	0.071

Table 1. **Quantitative results on KITTI-360.** Baseline results are from [9].

raw LiDAR information [1]. To produce simulated LiDAR scans with 512 beams, we linearly interpolate the beam angle from 14.870° to -30.166°, where laser ID $0, 8, \cdots 504$ corresponds to the original 64 beams (14.870° to -24.909°) for Pandar64. To reduce the domain gap between simulated 512-beam LiDAR data and real data, we replace simulated 64-beam LiDAR points (laser IDs $0, 8, \cdots 504$, 64 beams in total) with the original real LiDAR points.

Metrics on KITTI-360: The Maximum-Mean Discrepancy (MMD) and Jensen–Shannon divergence (JSD) metrics [9] measure the distribution between the generated results and the real data with a 100×100 2D histogram along the ground plane. In the main paper, we use occupancy as the measurement for all generation methods when doing histogram bin count for point clouds from real data. We now try to measure the point-based distribution by lifting our results with point duplication. Specifically, we calculate a coefficient for each bin used for the histogram bin count by performing an element-wise division of the 2D histogram matrix calculated on KITTI-360 training data with the histogram calculated on our generated data. We use the coefficients to "duplicate" the points in each voxel and match the point distribution. Our method still performs well in this case, as shown in Tab. 1.

3. Ablation Studies

We now show ablation studies on the design choices described for our model. Specifically, as it is easier to observe the difference between the generation results and the real data, we show qualitative results on KITTI-360.

Free space suppression: In Fig. 1, we show generation results without code restriction (*i.e.*, every code is available to choose, including the `[BLANK]` codes). In this situation, we do not perform any free space suppression, and we can see that the generation is clearly collapsed, indicating the effectiveness of our free space suppression.

[1] https://github.com/scaleapi/pandaset-devkit/issues/67

Voxel size: We empirically find that the spatial dimension that each code represents should not be too large. In Fig. 2, we show qualitative results of downsampling $16\times$ for the model before vector quantization, leading to a 2.5×2.5 m patch for each code. We can see that the results become noisy as 2.5×2.5 m patches contain too many geometry details that cannot be preserved in a single discrete code. We thus stay with $8\times$ downsampling for our model.

Codebook (re)-initialization: We also notice that the codebook initialization and reinitialization matter when learning the codebook. We train a model with conventional uniform initialization and no reinitialization. The code utilization rate at the end of codebook learning is 12.1%, significantly lower than our implementation with reinitialization which can achieve 99%+. As the qualitative results are shown in Fig. 3, it is clear that the limited number of active codes leads to degenerated results.

4. Visualization

We now show more visual examples on PandaSet and KITTI360.

Sparse-to-dense results We show more sparse-to-dense results on PandaSet in Fig. 4; we also include a video for results with a full sequence in the supplementary.

Unconditional generation results: We show more results compared with baselines on KITTI-360 in Fig. 5. Furthermore, we also show KITTI-360 results with the generation process in Fig. 6, and PandaSet results in Fig. 7, respectively.

Conditional generation results: We show KITTI-360 results in Fig. 8, and PandaSet results in Fig. 9 on PandaSet, respectively. The red points indicate the visible part as the condition for the model, and our model can perform reasonable extrapolation based on that.

Manipulation results: Lastly, we show more manipulation results on KITTI-360 and PandaSet in Fig. 11. The learned codes show high spatial alignment with the objects, so we can manipulate the LiDAR sweeps by changing the code placement on the code map.

Missing part in completion: UltraLiDAR is designed to densify the input point clouds while maintaining realistic occlusion patterns such that the data appear as if it were captured by a LiDAR sensor with a higher beam count. The large missing volume in Fig. 3 in the main paper is due to uphill/downhill geometry of the road, which results in occlusion for any LiDAR regardless of their beam count. The RGB camera images shown in Fig. 10 provides a clearer illustration.

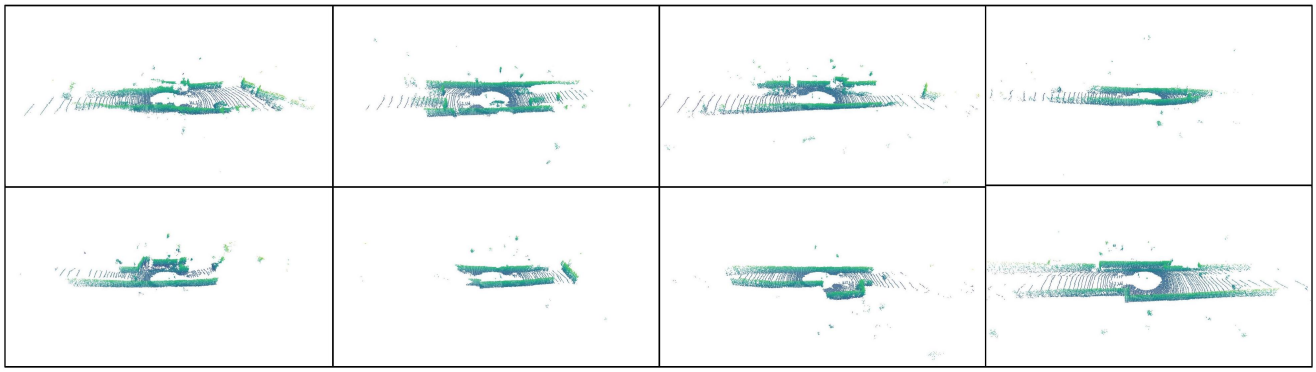


Figure 1. **Results without free space suppression**. Most space is now empty; the model can no longer generate structured layouts.

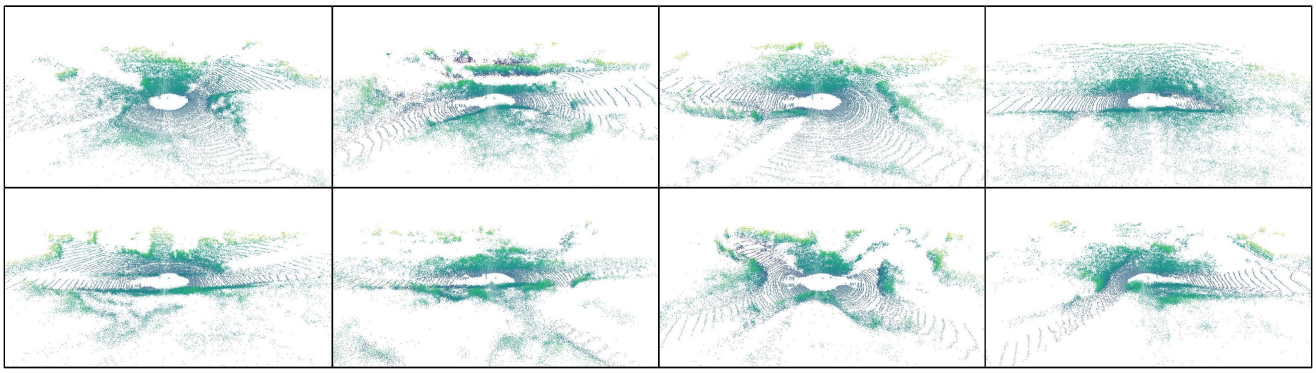


Figure 2. **Results with larger patch size for each code.** The point clouds become noisy as the patch size for each code is too large, and fine-grained geometry cannot be preserved anymore.

References

[1] 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

[2] 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. 1

[3] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. *ACM TOG*, 2010. 1

[4] Axel Sauer, Kashyap Chitta, Jens Müller, and Andreas Geiger. Projected gans converge faster. *Advances in Neural Information Processing Systems*, 34:17480–17492, 2021. 2

[5] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 1

[6] 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. 1

[7] 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. 1

[8] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020. 1

[9] 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

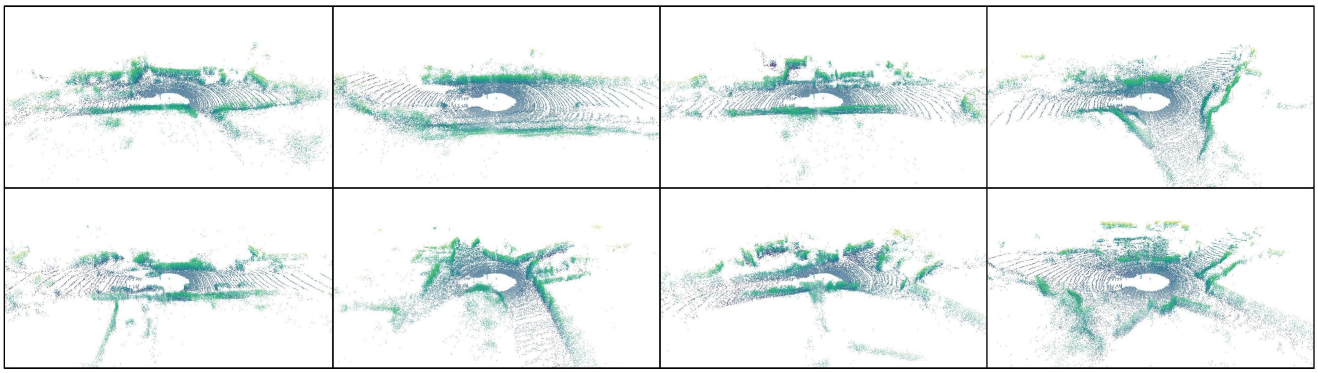


Figure 3. **Results without codebook reinitialization.** We observe similar issue as in Fig. 2. The model needs to use a limited number of codes to perform reconstruction/generation, leading to degenerated results.

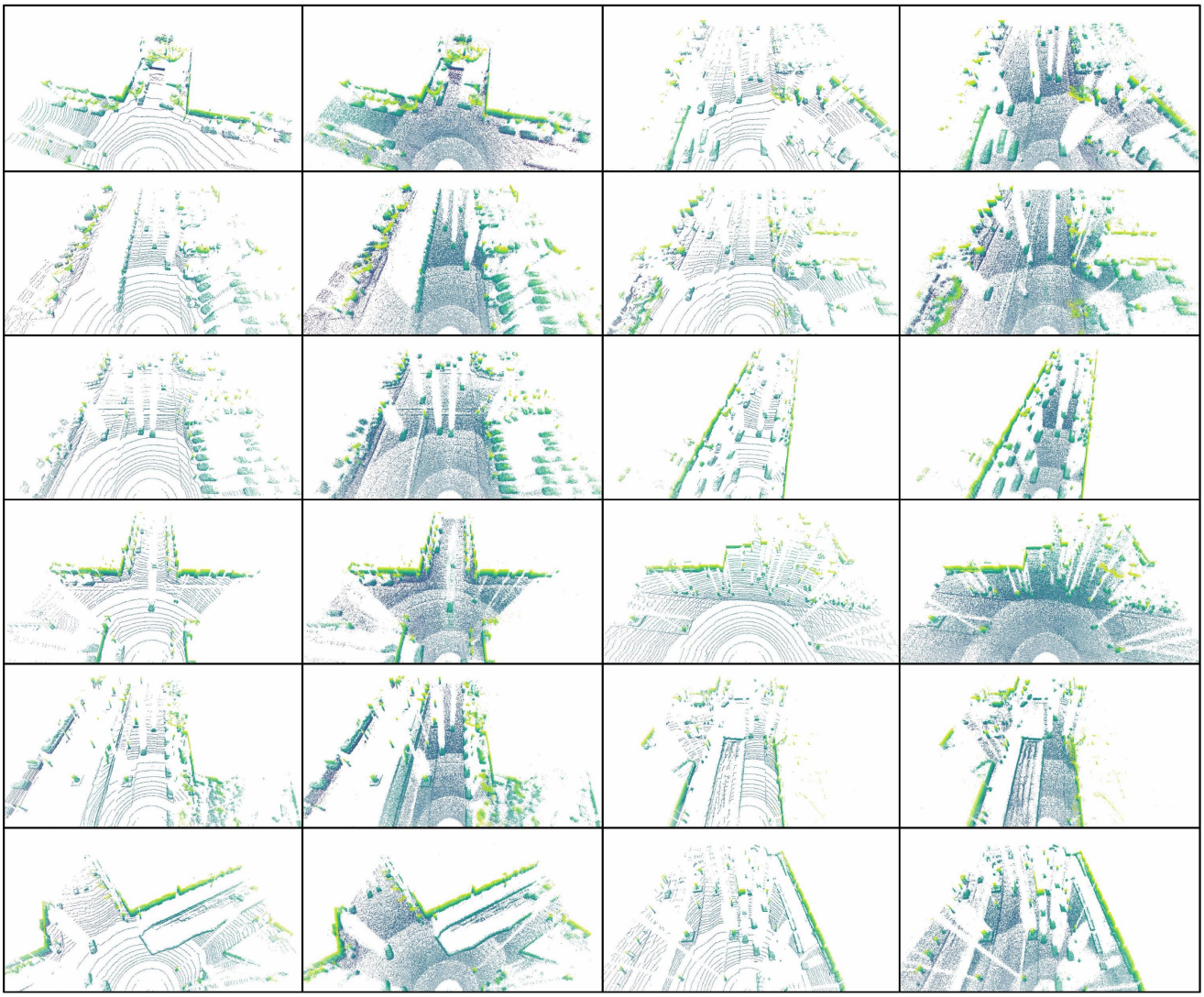


Figure 4. **Sparse-to-Dense results on PandaSet.** The first and third columns are real sparse data, and the second and fourth columns are our densified results.

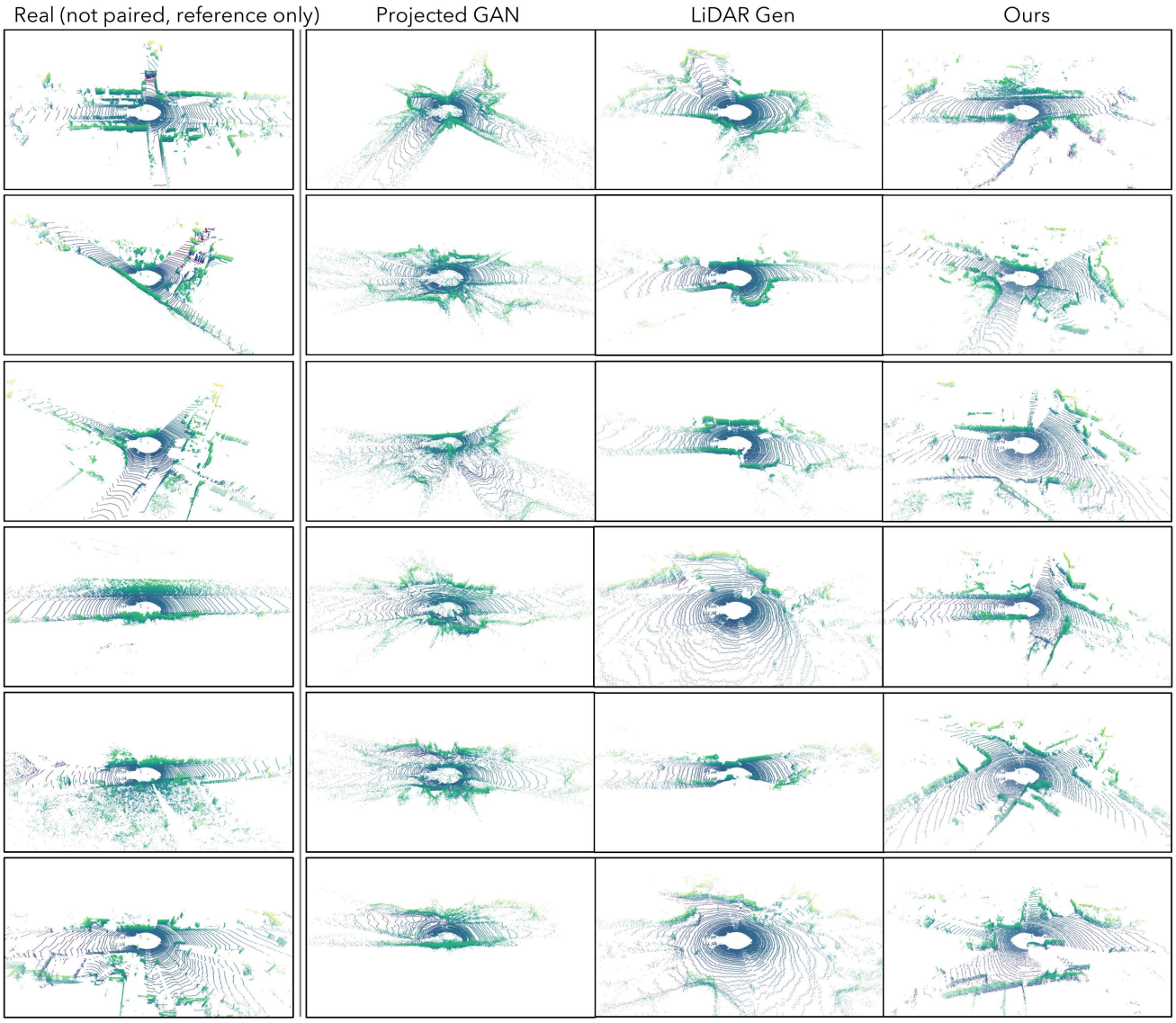


Figure 5. **Qualitative comparison against baselines on unconditional LiDAR generation.** Our model consistently outperforms the baselines and shows results highly similar to the real data.

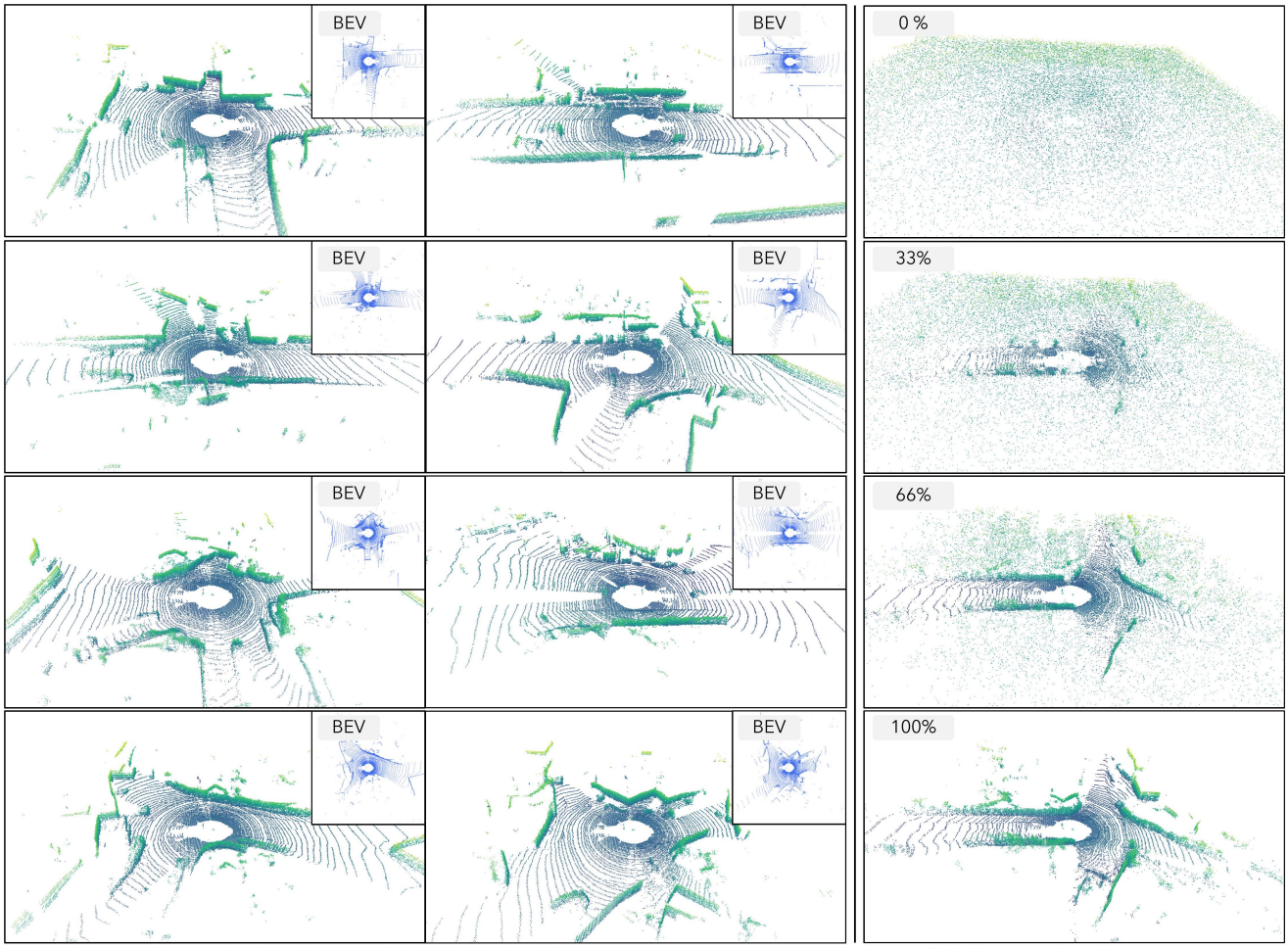


Figure 6. **Unconditional generation results on KITTI-360.** We show a step-by-step generation process starting from a blank canvas in the third column.

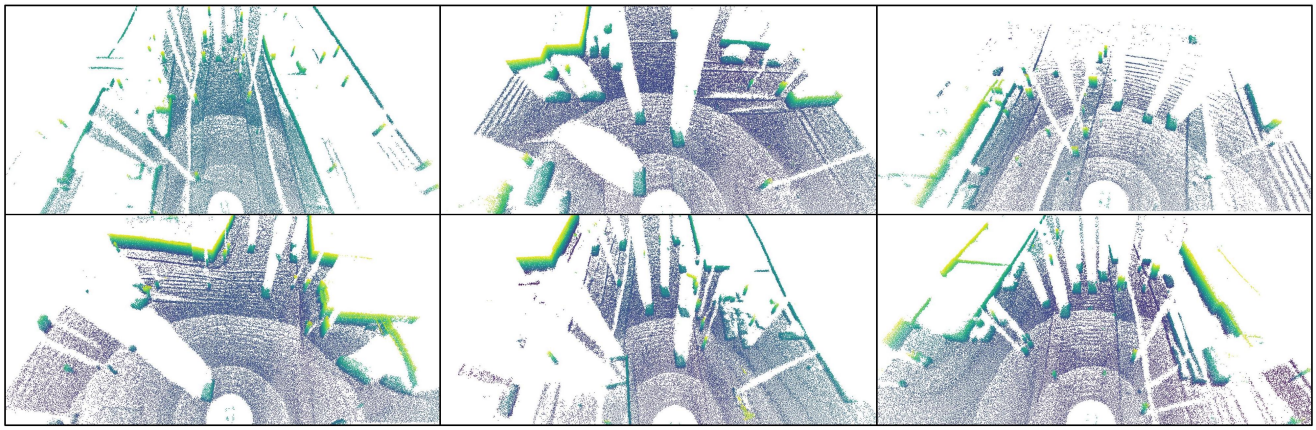


Figure 7. **Unconditional generation results on Pandaset.**

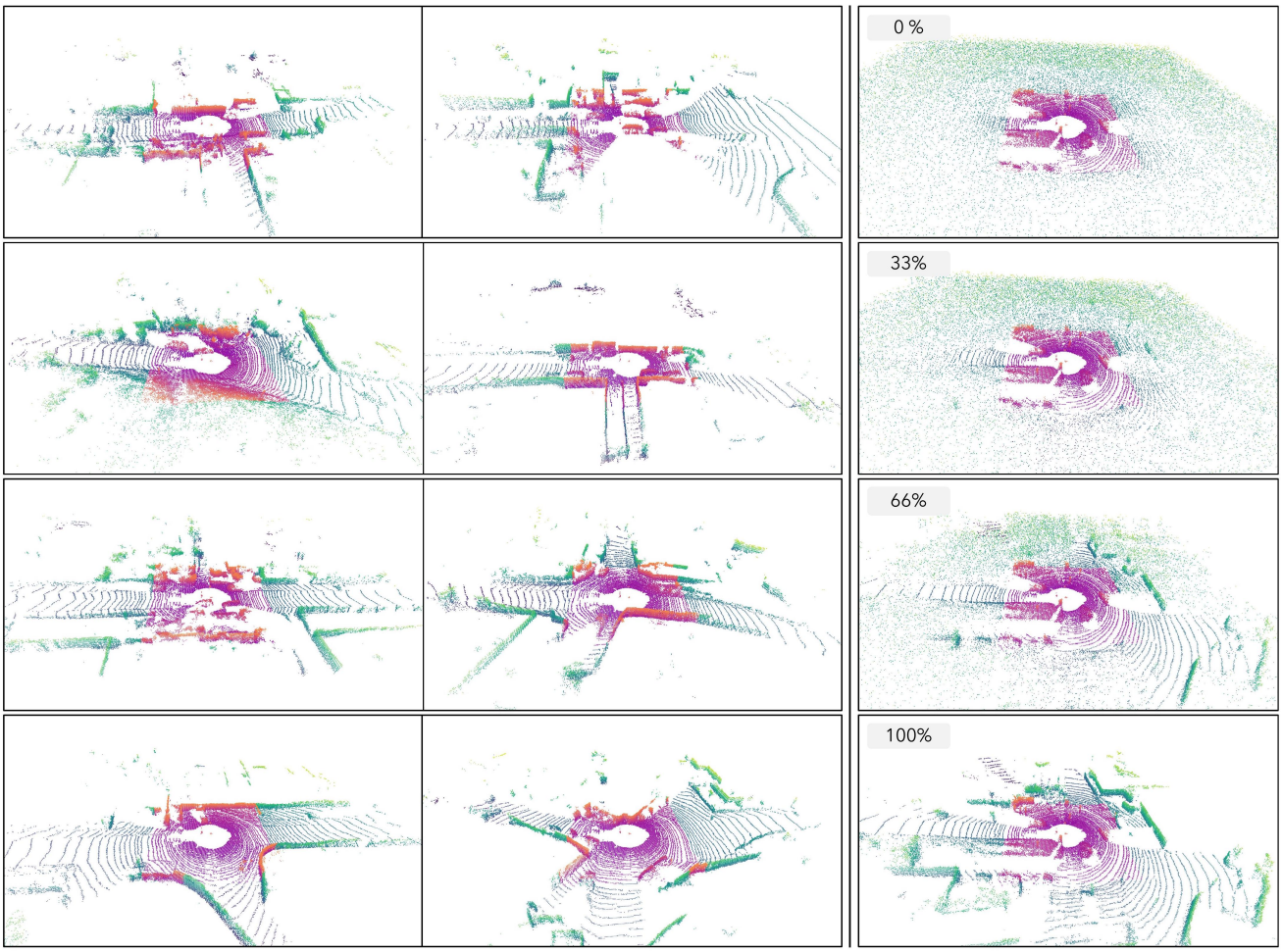


Figure 8. **Conditional generation results on KITTI-360.** The red points are the visible input to the model.

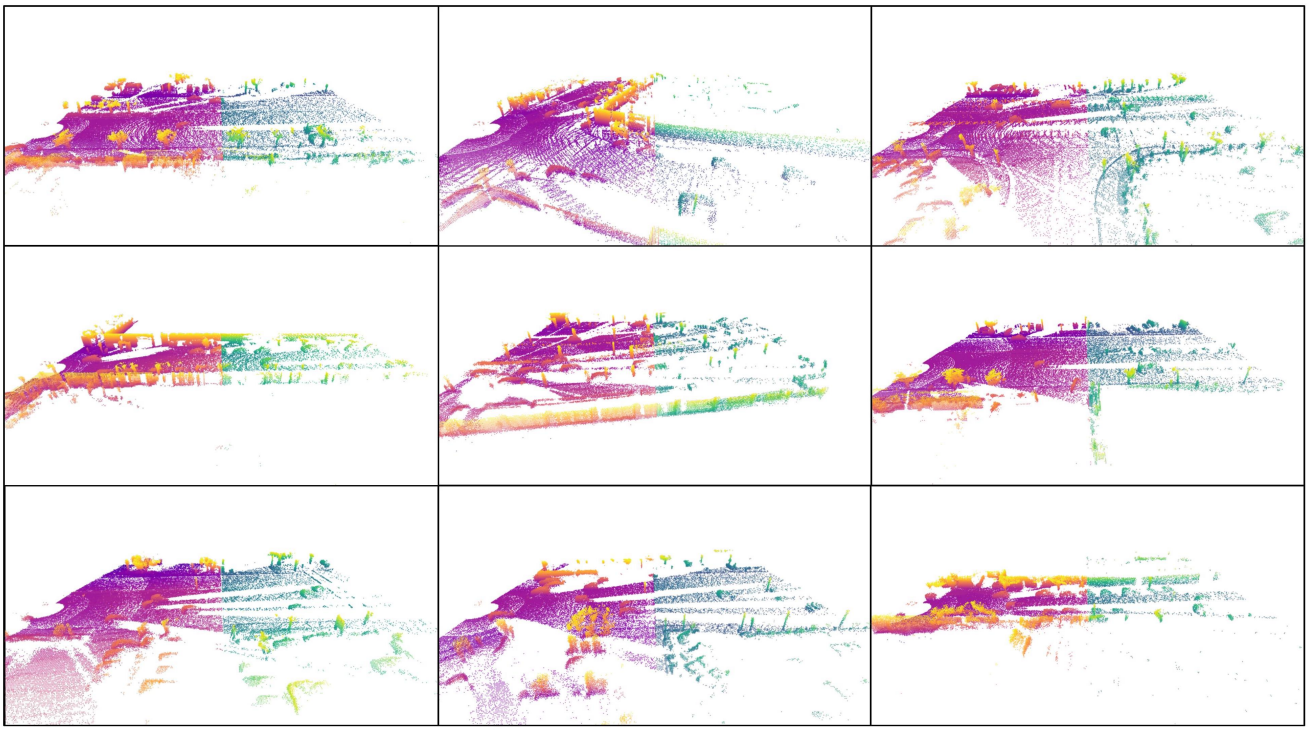
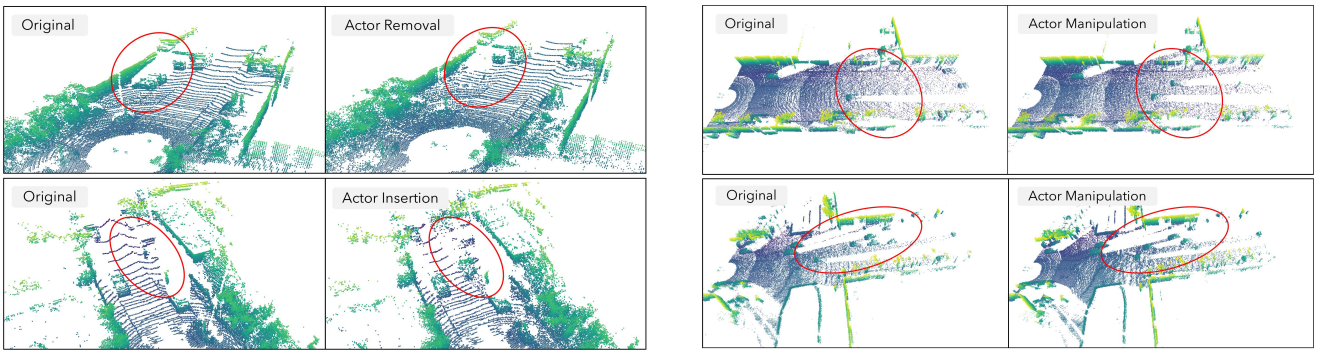


Figure 9. **Conditional generation results on Pandaset.** The red points are the visible input to the model.

Figure 10. Camera images for the corresponding LiDAR completion examples. See text for detailed explanations.



(a) KITTI-360 manipulation results

(b) PandaSet manipulation results

Figure 11. **Manipulation results on (a) KITTI-360 and (b) PandaSet.** The learned codes show high spatial alignment with the objects, so we can manipulate the LiDAR sweeps by changing the code placement on the code map.