RENO: Real-Time Neural Compression for 3D LiDAR Point Clouds

Supplementary Material

1. Overview

This appendix provides supplementary comparisons and discussions to complement the manuscript. Section 2 compares our approach with learning-based methods, while Sec. 3 presents additional ablation studies. Implementation details are discussed in Sec. 4, and more visualizations are provided in Sec. 5. All experiments in this appendix are conducted on an Intel Xeon Platinum 8352V CPU and one RTX 4090 GPU.

2. Comparison with Learning-based Methods

In this section, we provide a detailed comparison of RENO with representative learning-based methods: EHEM [5] and Unicorn [7]. EHEM is recognized as the most prominent and high-performing model among tree-based approaches [1–3, 5], while Unicorn exemplifies the latest and most representative sparse tensor-based models [6, 7, 10]. Table 1 shows the rate distortion performance of different methods, where RENO-L represents the large version of RENO, which increases the number of channels from 32 to 128 and the kernel size from 3 to 5. Table 2 provides detailed encoding time for various methods applied to 12-bit and 14-bit point clouds, where the first frame in SemanticKITTI sequence 11 is used for test.

- It can be observed that EHEM demonstrates the highest rate-distortion performance (note that the BD-BR of EHEM is directly cited from their paper and is presented for reference only); however, this comes at the cost of substantial time consumption. Its preprocessing stage demands significantly more time to construct the tree and prepare contextual information, ultimately undermining its real-time applicability.
- In contrast, Unicorn performs inference directly within the sparse tensor domain, effectively minimizing preprocessing overhead. However, its reliance on the upsampling-based inference framework results in a substantially prolonged neural network computation time. While the inference speed of the network can be enhanced using 1-stage inference, Unicorn[1] fails to achieve substantial speed improvements (as it still necessitates the introduction of a large number of voxels via deconvolution) and results in a significant degradation in performance.

3. Additional Ablation Study

CPU vs. GPU Arithmetic Coding. This paper employs a bitwise two-stage coding strategy to accelerate the arithmetic coding process, leveraging the Torchac[1] library, which operates on the CPU. An alternative acceleration strategy involves dividing symbols into packets for parallel coding directly on the GPU, an approach implemented in GPUAC[2]. Here, we performed ablation experiments for both methods, and the results are shown in Tab. 3. It should

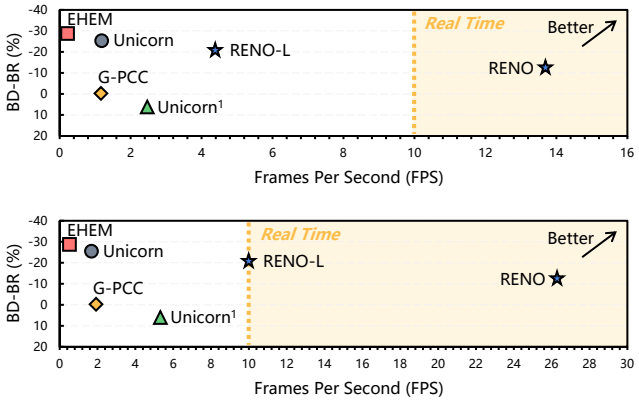

Figure 1. Comparison with the learning-based methods EHEM and Unicorn. The encoding speeds are reported separately for 14-bit precision (top) and 12-bit precision (bottom). Unicorn[1] refers to the one-stage configuration of the Unicorn model. G-PCC serves as the anchor.

Table 1. Rate-distortion comparison with learning-based methods on SemanticKITTI. BD-BR refers to the BD-BR gain over G-PCCv23 in D1 metric. Unicorn[1] denotes the one-stage configuration of the Unicorn model.

Method	EHEM	Unicorn	Unicorn[1]	RENO-L	RENO
BD-BR (%)	-28.89	-25.62	+6.56	-20.63	-12.47

Table 2. Encoding time comparison. Preprocessing (Prep), neural network inference (NN), and arithmetic encoding (AE) time are independently reported for detailed comparison. D refers to the bit depth of reconstructed point clouds. For sparse tensor-based methods (e.g., Unicorn, Unicorn[1], RENO-L, and RENO), the time spent on dyadic downscaling is reported as preprocessing.

Method	D=12 (s)				D=14 (s)			
	Prep	NN	AE	**Total**	Prep	NN	AE	**Total**
EHEM	0.463	0.427	0.297	**1.187**	1.796	1.385	1.233	**4.414**
Unicorn	0.003	0.565	0.030	**0.598**	0.003	0.772	0.074	**0.849**
Unicorn[1]	0.003	0.163	0.021	**0.187**	0.003	0.336	0.063	**0.402**
RENO-L	0.006	0.080	0.014	**0.100**	0.007	0.184	0.037	**0.228**
RENO	0.006	0.018	0.014	**0.038**	0.007	0.028	0.038	**0.073**

[1] https://github.com/fab-jul/torchac
[2] https://github.com/zb12138/GPUAC

Table 3. Comparison of arithmetic coding implementations. The first frame in the SemanticKITTI sequence 11 is used for testing. Encoding time for different bit depths D are reported.

Module	Library	D=12		D=14	
		Time (s)	Bitrate	Time (s)	Bitrate
One-Stage	GPUAC	0.065	2.24	0.096	6.56
Two-Stage	GPUAC	0.064	2.25	0.093	6.55
One-Stage	Torchac	0.017	2.27	0.102	6.58
Two-Stage	Torchac	0.014	2.28	0.038	6.57

be noted that an independent arithmetic encoding process is conducted for each scale, and the reported time represents the accumulated arithmetic encoding time across all scales. The default packet size of 8192 is used when implementing GPUAC. As observed, GPUAC provides a slight speedup over the naive one-stage approach only at higher bitrates (D=14); however, this performance improvement is considerably inferior compared to the proposed bitwise decomposition strategy. At lower bitrates (D=12), GPUAC performs significantly slower, as the number of symbols is relatively small, and the CPU is sufficiently capable of processing the data with low latency.

4. Implementation Details

4.1. Detailed Network Structure

Figure 2 illustrates the neural network architecture and detailed parameters employed by RENO. The model parameters are shared across all point cloud scales.

4.2. Configuration of Comparative Methods

G-PCCv23[3] is employed following the Common Test Condition (CTC) recommended by the MPEG committee. For KITTI point clouds, the coordinates are scaled by dividing by 0.001, and the $posQ$ parameter is adjusted to achieve different bitrates. After decoding, the reconstructed point clouds are rescaled to the original coordinate system by multiplying by 0.001. For Ford point clouds, as they are already quantized to 1 mm precision, the original coordinate system is retained.

DracoPy[4] , utilizing Google's Draco version 1.5.2, is employed for compression in this study. Despite its longevity, Draco remains a benchmark for real-time scenarios [4]. The compression level is maintained at its default setting of 1, while the quantization bits are adjusted to achieve varying compression ratios.

R-PCC[5] is considered as a baseline for range image-based compression. The BZip2 compressor is employed for the optimal compression rate, and varying compression ratios are achieved by adjusting the $accuracy$ parameter.

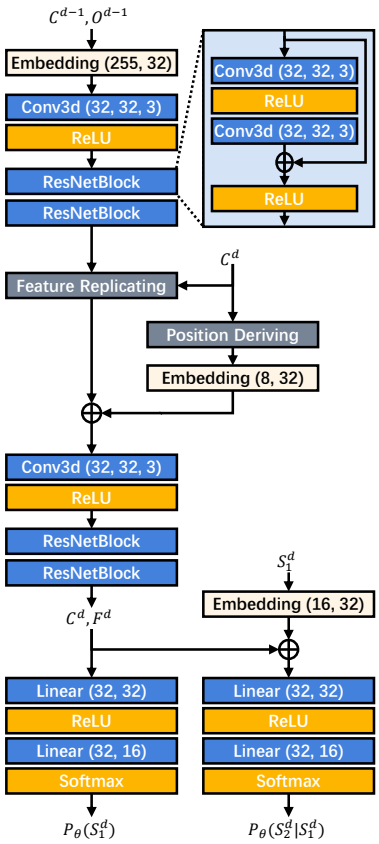

Figure 2. Detailed network structure of the devised Target Occupancy Predictor (TOP). "Conv3d" represents 3D sparse convolution, with parameters defined as (input channels, output channels, kernel size); "Embedding" maps discrete variables according to (dictionary size, vector dimension); "Linear" applies an affine linear transformation with parameters (input channels, output channels); "$\oplus$" denotes element-wise addition.

RTST[6] is employed as an additional range image-based compression approach. Single-frame compression mode is adopted, and the horizontal and vertical degree granularities are systematically varied to attain different compression ratios. Note that it is currently supports only the KITTI dataset.

Remark. We observe a notable discrepancy between our experimental results and those reported in the original R-PCC paper [8], which reported markedly superior performance of R-PCC over G-PCC. This divergence primarily stems from methodological differences in PSNR computation. Specifically, the original implementation of R-PCC (as verified through their GitHub repository) employs back-projected geometry rather than original point cloud data

[3] https://github.com/MPEGGroup/mpeg-pcc-tmc13
[4] https://github.com/seung-lab/DracoPy
[5] https://github.com/StevenWang30/R-PCC
[6] https://github.com/horizon-research/Real-Time-Spatio-Temporal-LiDAR-Point-Cloud-Compression

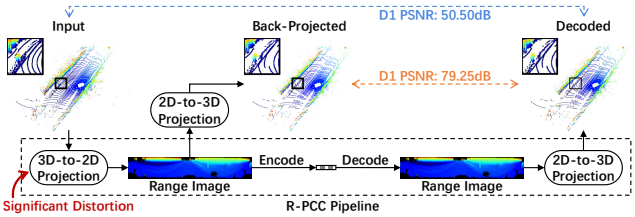

Figure 3. Ours (blue line) vs. R-PCC (orange line) PSNR evaluations. A point cloud from the KITTI dataset is utilized as an exemplar. The setting of PSNR peak value is consistent at 59.70.

as ground truth, thereby circumventing projection-induced distortions illustrated in Fig. 3. The projection distortion is confirmed in both prior studies [9] and our experiments. In contrast, our calculation is more reasonable and aligns with prior arts.

4.3. Metrics

PSNR. The MPEG PCC quality measurement software version 0.13.4 is used to report PSNR values. The peak values are set to 59.70 for KITTI point clouds and 30,000 for Ford point clouds, following conventional practices [1, 2, 5].

Chamfer Distance. The Chamfer Distance in the Fig. 1 of the main manuscript serves as an auxiliary description of point-level distortion. This metric is computed using the following mathematical formulation, implemented according to the Point Cloud Utils[7]:

$$\begin{aligned}\text{chamfer}\left(P_1, P_2\right) = &\frac{1}{2n}\sum_{i=1}^{n}\left|x_i - \text{NN}\left(x_i, P_2\right)\right| \\ &+ \frac{1}{2m}\sum_{j=1}^{m}\left|x_j - \text{NN}\left(x_j, P_1\right)\right| \quad (1)\end{aligned}$$

where $P_1 = \{x_i\}_{i=1}^{n}$ and $P_2 = \{x_j\}_{j=1}^{m}$ refer to two point cloud samples; $\text{NN}\left(x, P\right) = \text{argmin}_{x' \in P}\left\|x - x'\right\|$ is the nearest neighbor function.

5. More Subjective Results

More visualization results are provided in Fig. 4. As seen from the leftmost column (D=12), RENO reconstructs point cloud samples with shapes sufficient to distinguish object categories at around 50 ms latency, differing mainly in fine details. For 14-bit precision (D=14), it achieves real-time reconstruction with details closely matching the originals.

References

[1] Chunyang Fu, Ge Li, Rui Song, Wei Gao, and Shan Liu. Octattention: Octree-based large-scale contexts model for point cloud compression. In *Proceedings of the AAAI conference on artificial intelligence*, pages 625–633, 2022. 1, 3

[2] Lila Huang, Shenlong Wang, Kelvin Wong, Jerry Liu, and Raquel Urtasun. Octsqueeze: Octree-structured entropy model for lidar compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1313–1323, 2020. 3

[3] Yiqi Jin, Ziyu Zhu, Tongda Xu, Yuhuan Lin, and Yan Wang. Ecm-opcc: Efficient context model for octree-based point cloud compression. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7985–7989. IEEE, 2024. 1

[4] Zhicheng Liang et al. Fumos: Neural compression and progressive refinement for continuous point cloud video streaming. *IEEE TVCG*, 2024. 2

[5] Rui Song, Chunyang Fu, Shan Liu, and Ge Li. Efficient hierarchical entropy model for learned point cloud compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14368–14377, 2023. 1, 3

[6] Jianqiang Wang, Dandan Ding, Zhu Li, Xiaoxing Feng, Chuntong Cao, and Zhan Ma. Sparse tensor-based multiscale representation for point cloud geometry compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(7):9055–9071, 2022. 1

[7] Jianqiang Wang, Ruixiang Xue, Jiaxin Li, Dandan Ding, Yi Lin, and Zhan Ma. A versatile point cloud compressor using universal multiscale conditional coding – part i: Geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–18, 2024. 1

[8] Sukai Wang, Jianhao Jiao, Peide Cai, and Lujia Wang. R-pcc: A baseline for range image-based point cloud compression. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10055–10061. IEEE, 2022. 2

[9] Tao Wu, Hao Fu, Bokai Liu, Hanzhang Xue, Ruike Ren, and Zhiming Tu. Detailed analysis on generating the range image for lidar point cloud processing. *Electronics*, 10(11):1224, 2021. 3

[10] Ruixiang Xue, Jianqiang Wang, and Zhan Ma. Efficient lidar point cloud geometry compression through neighborhood point attention. *arXiv preprint arXiv:2208.12573*, 2022. 1

[7] https://fwilliams.info/point-cloud-utils/sections/shape_metrics/

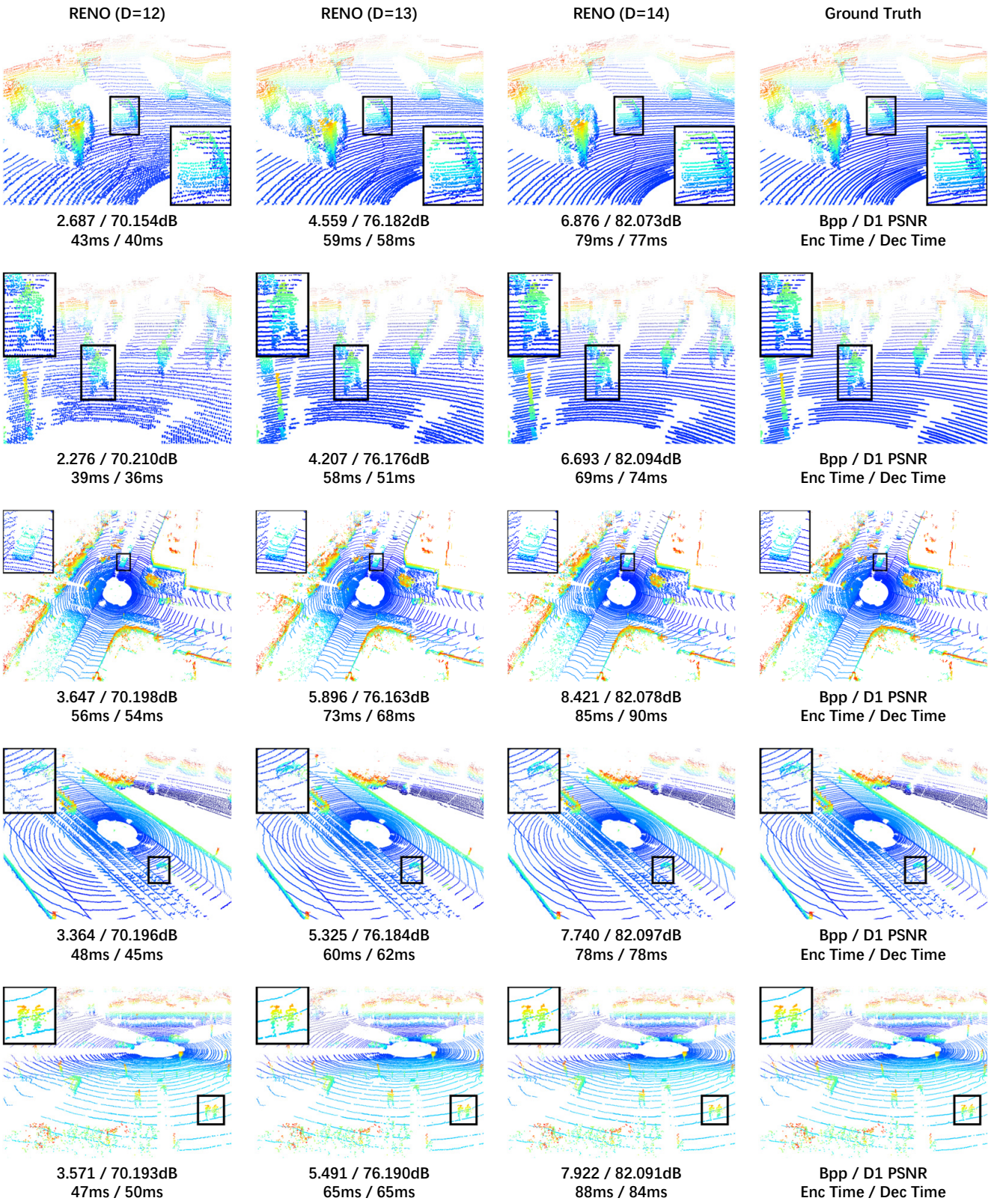

Figure 4. Visualization of compression results for KITTI point cloud. The reconstruction results of RENO under bit depths (D) of 12, 13, and 14 are presented.