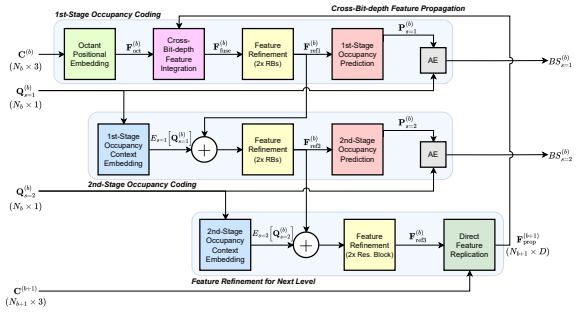
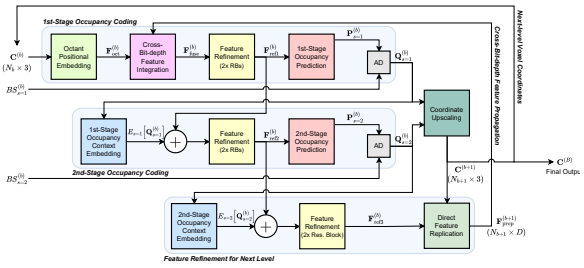
ELiC: Efficient LiDAR Geometry Compression via Cross-Bit-depth Feature Propagation and Bag-of-Encoders

Supplementary Material

A. Encoding and Decoding Pipelines



(a) Encoding pipeline



(b) Decoding pipeline

Figure S1. Coding network execution pipeline for ELiC during encoding and decoding.

Fig. S1 illustrates the ELiC coding-network execution pipeline for encoding and decoding. Below we describe the encoding and decoding flows.

At each bit-depth level b, the encoder receives the voxel coordinates $\mathbf{C}^{(b)}$ and the stage-wise quadrant labels $\{\mathbf{Q}_s^{(b)}\}_{s=1,2}$. Following the selection policy in Sec. 4.3, the index $k^{(b)}$ of the coding network to use at bit-depth level b is determined. The selected network processes the inputs and produces the estimated occupancy probabilities $\{\mathbf{P}_s^{(b)}\}_{s=1,2}$. These probabilities are then passed to the arithmetic encoder, which generates the bitstreams $\{BS_s^{(b)}\}_{s=1,2}$.

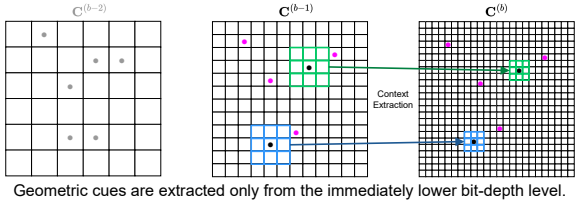
This procedure is repeated sequentially for $b{=}2$ to $B{-}1$. The final compressed representation consists of the base coordinates $\mathbf{C}^{(2)}$, the set of stage-wise bitstreams $\{BS_s^{(b)}\}_{s=1,2}$, and the BoE indices $\{k^{(b)}\}_{b=7}^{B-1}$ (for $b{=}2,\ldots,6$ the base coding network is used).

At decoding time, the process mirrors the encoder. The decoder is given the base coordinates $\mathbf{C}^{(2)}$ and initializes $\mathbf{C}^{(b)}$ for $b{=}2$.

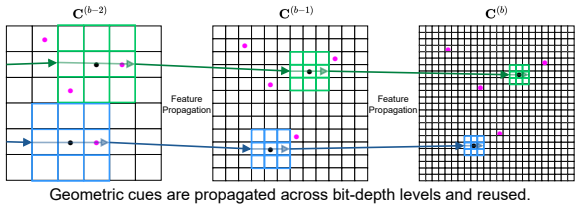
For each bit-depth level $b{=}2$ to $B{-}1$, the decoder first reads the BoE index $k^{(b)}$ from the bitstream, selecting the same coding network as the encoder. Given $\mathbf{C}^{(b)}$, the selected network predicts the stage-wise quadrant probability distributions $\{\mathbf{P}_s^{(b)}\}_{s=1,2}$. These probabilities are then used by the arithmetic decoder to reconstruct the quadrant labels $\{\mathbf{Q}_s^{(b)}\}_{s=1,2}$ from the bitstreams $\{BS_s^{(b)}\}_{s=1,2}$.

Once both stages are decoded, the voxel coordinates are expanded to the next bit-depth, and $\mathbf{C}^{(b+1)}$ is generated from the child-occupancy map $\mathbf{O}^{(b)}{=}16\,\mathbf{Q}_{s=2}^{(b)}{+}\mathbf{Q}_{s=1}^{(b)}$. This procedure repeats until $b{=}B{-}1$, at which point the final reconstructed point set $\mathbf{C}^{(B)}$ is obtained.

B. Comparison of Cross-Bit-depth and Level-Independent Architectures



(a) Bit-depth-level-independent design



(b) Cross-bit-depth feature propagation design

Figure S2. Comparison between level-independent and cross-bit-depth designs for LiDAR geometry compression. (a) Bit-depth-level-independent design: each level re-derives lower-level context solely from the current-level voxel coordinates. (b) Cross-bit-depth feature propagation: features extracted at lower, denser levels are propagated and fused at higher levels.

Fig. S2 compares the bit-depth-level-independent design with ELiC's cross-bit-depth feature propagation design.

In the bit-depth-level-independent design, each bit-depth level computes $\mathbf{C}^{(b-1)}$ or $\mathbf{O}^{(b-1)}$ from the current voxel

coordinates $\mathbf{C}^{(b)}$ and re-derives lower-level occupancy context from them. A drawback is that when both the current and lower levels are highly sparse, the available occupancy context carries minimal predictive information.

In contrast, the cross-bit-depth feature propagation design propagates features from a lower level to the next level. The propagated features are blended with the current features via Eq. 6 of the main paper and then passed upward again, with the blending ratios determined by channel-wise weights learned during training. By naturally propagating features from coarse to fine resolutions, the model preserves informative multi-scale structure even when $\mathbf{C}^{(b)}$ and $\mathbf{C}^{(b-1)}$ are extremely sparse, thereby strengthening occupancy prediction and improving entropy-modeling robustness.

C. Morton-Order-Preserving Coordinate Transformation

By enforcing a Morton-order-preserving hierarchy, we eliminate the expensive up- and down-convolution operations across the entire encoding and decoding pipeline that are typically used to construct the bit-depth hierarchy and its occupancy labels. Our method requires only a single Morton-order sort of the input voxel coordinates before encoding, after which all subsequent coordinate and occupancy labels are obtained via cheap integer operations.

Let $\mathbf{C}^{(b)} = \{\mathbf{c}_i^{(b)} = (x_i^{(b)}, y_i^{(b)}, z_i^{(b)})\}_{i=1}^{N_b}$ denote the occupied voxel coordinates at bit-depth b. Each voxel is assigned a 3D Morton code

$$m(\mathbf{c}^{(b)}) = \sum_{k=0}^{b-1} \big(x_k^{(b)} + (y_k^{(b)} \ll 1) + (z_k^{(b)} \ll 2)\big) 8^k, \quad (11)$$

where $(x_k^{(b)}, y_k^{(b)}, z_k^{(b)})$ are the k-th least significant bits of each coordinate. Sorting by $m(\mathbf{c})$ yields a Z-ordered sequence in which spatial adjacency is preserved along a one-dimensional index.

A duplicated set of parent coordinates is obtained by halving the integer coordinates,

$$\widetilde{\mathbf{C}}^{(b-1)} = \Big\lfloor \frac{\mathbf{C}^{(b)}}{2} \Big\rfloor, \quad (12)$$

which is equivalent to truncating the three least significant bits of the Morton code:

$$m^{(b-1)}(\widetilde{\mathbf{c}}^{(b-1)}) = m^{(b)}(\mathbf{c}^{(b)}) \gg 3. \quad (13)$$

Therefore, when $\mathbf{C}^{(b)}$ is sorted in Morton order, removing duplicates from each consecutive group of identical entries in $\widetilde{\mathbf{C}}^{(b-1)}$ directly yields the unique parent coordinates $\mathbf{C}^{(b-1)}$.

Because the three-bit right shift preserves lexicographic order on Morton codes, the parent coordinates inherit the same Morton ranking as their children. As a result, all voxels that share an identical parent code form a contiguous interval in this order. For each parent voxel, we then define an occupancy label

$$\mathbf{O}_j^{(b-1)} = \sum_{\mathbf{c}_i^{(b)} \in \mathrm{child}(\mathbf{c}_j^{(b-1)})} 2^{\,\pi(\mathbf{c}_i^{(b)})},$$
$$\pi(\mathbf{c}_i^{(b)}) = (x_i^{(b)} \bmod 2) + 2(y_i^{(b)} \bmod 2) + 4(z_i^{(b)} \bmod 2), \quad (14)$$

where $\pi(\mathbf{c}_i^{(b)}) \in \{0, \ldots, 7\}$ assigns each child to one of the eight local octants in the $2\times2\times2$ voxel block centered at $\mathbf{c}_j^{(b-1)}$. The resulting 8-bit integer $\mathbf{O}_j^{(b-1)}$ therefore encodes exactly which of these eight child positions are occupied.

In practice, this occupancy-label generation uses only inexpensive integer operations on the child coordinates and avoids the conventional $2\times2\times2$ down-convolution with fixed kernel weights $[1, 2, 4, 8, 16, 32, 64, 128]$ and the associated kernel-map construction overhead.

Conversely, the finer level can be reconstructed directly from $(\mathbf{C}^{(b-1)}, \mathbf{O}^{(b-1)})$ by expanding every parent voxel according to its occupancy bits. Let the fixed octant offsets be $\{\boldsymbol{\delta}_u\}_{u=0}^{7} \subset \{0, 1\}^3$ listed in Morton octant order $u = \delta_x + 2\delta_y + 4\delta_z$. Given parent coordinates $\mathbf{C}^{(b)} = \{\mathbf{c}_n^{(b)}\}_{n=1}^{N_b}$ and 8-bit occupancies $\mathbf{O}^{(b)} \in \{0, \ldots, 255\}^{N_b}$, form the candidate set

$$\widetilde{\mathbf{C}}^{(b+1)} = \big\{\, 2\,\mathbf{c}_n^{(b)} + \boldsymbol{\delta}_u \;\big|\; n \in [1\!:\!N_b],\, u \in [0\!:\!7] \,\big\}, \quad (15)$$

i.e., for each parent, enumerate its eight child candidates in a fixed (Morton) octant order. Define a Boolean mask

$$\mathbf{M}^{(b)}(n, u) = \big((\mathbf{O}_n^{(b)} \gg u)\,\&\,1\big) \in \{0, 1\}, \quad (16)$$

which selects the octant u when its occupancy bit is set. Since the mapping from each occupancy value in $\{0, \ldots, 255\}$ to its 8-bit pattern is fixed, $\mathbf{M}^{(b)}$ can be implemented more efficiently via a precomputed lookup table, avoiding repeated bit-shift and mask operations. Applying this mask by row selection yields

$$\mathbf{C}^{(b+1)} = \big\{\, \widetilde{\mathbf{C}}^{(b+1)}(n, u) \;\big|\; \mathbf{M}^{(b)}(n, u) = 1 \,\big\}. \quad (17)$$

Equivalently, the same selection can be expressed in vectorized form. Flattening $\mathbf{M}^{(b)}$ in row-major order yields a vector in $\{0, 1\}^{8N_b}$, which we use to index $\widetilde{\mathbf{C}}^{(b+1)}$ and obtain the actual child coordinates:

$$\mathbf{C}^{(b+1)} = \widetilde{\mathbf{C}}^{(b+1)}\big|_{\mathbf{M}^{(b)}=1}. \quad (18)$$

If per-parent features $\mathbf{F}^{(b)} \in \mathbb{R}^{N_b \times D}$ are propagated, replicate them to the candidate set and select with the same mask:

$$\widetilde{\mathbf{F}}^{(b)}(n, u) = \mathbf{F}^{(b)}(n),$$
$$\mathbf{F}_{\text{prop}}^{(b+1)} = \big\{\, \widetilde{\mathbf{F}}^{(b)}(n, u) \;\big|\; \mathbf{M}^{(b)}(n, u) = 1 \,\big\}, \quad (19)$$

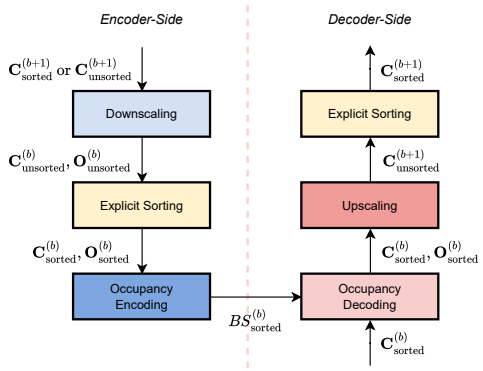
which can equivalently be written in vectorized form as

$$\mathbf{F}_{\text{prop}}^{(b+1)} \;=\; \widetilde{\mathbf{F}}^{(b)}|_{\mathbf{M}^{(b)}=1} \;\in \mathbb{R}^{N_{b+1}\times D}. \tag{20}$$

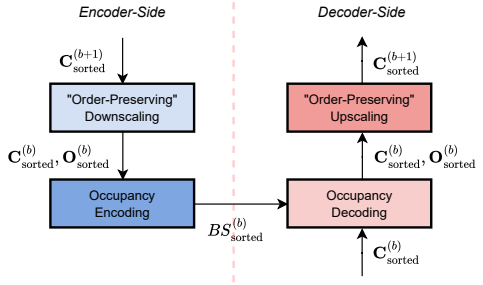
Since parents are listed in Morton order and $\{\boldsymbol{\delta}_u\}_{u=0}^{7}$ is enumerated in Morton octant order, the selection by $\mathbf{M}^{(b)}$ preserves the global Morton order for both $\mathbf{C}^{(b+1)}$ and $\mathbf{F}_{\text{prop}}^{(b+1)}$ without any re-sorting.

The reference implementation of this Morton-order-preserving hierarchy and occupancy labeling procedure is available in the `morton.py` file in the code repository.

D. Runtime Impact of Morton-Order Hierarchy



(a) With explicit per-level sorting



(b) With Morton-order-preserving hierarchy

Figure S3. Comparison between explicit per-level sorting and a Morton-order-preserving hierarchical representation. (a) Explicit sorting reorders points at each bit-depth level, incurring additional overhead in both encoding and decoding. (b) A Morton-order-preserving hierarchy maintains consistent spatial ordering across levels, enabling stable parent-child mapping and reducing per-level processing cost.

Fig. S3 provides a high-level comparison, at a single bit-depth level during compression, between explicit per-level sorting and our Morton-order-preserving hierarchy. In sparse-tensor-based geometry compression, it is essential that the encoder and decoder operate on exactly the same coordinate order so that occupancy bits are generated and consumed with aligned voxel indices. Existing sparse-tensor-based approaches satisfy this by performing identical sorting at every downscaling and upsampling step in both encoding and decoding. In contrast, ELiC performs a single initial 3D Morton sort of the input voxel coordinates $\mathbf{C}^{(B)}$ and then applies downscaling and upsampling that preserve this global order, eliminating the per-level re-sorting required by prior methods in both encoding and decoding.

Table S1 compares the runtime of ELiC with and without the Morton-order-preserving hierarchy. When explicit sorting is replaced by Morton-order traversal, the average encoding and decoding times are reduced from 0.142 seconds and 0.128 seconds to 0.121 seconds and 0.111 seconds, respectively—corresponding to latency reductions of approximately 14.5% and 13.3%. This confirms that maintaining hierarchical Morton order effectively removes sorting overhead and improves overall runtime efficiency.

All runtimes reported in this supplementary material were measured on a machine equipped with an NVIDIA GeForce RTX 3090 GPU and an Intel Core i9-9900K CPU with 64 GB of RAM.

Table S1. Runtime comparison between ELiC with explicit sorting and ELiC with Morton-order-preserving hierarchy.

	ELiC with Explicit Sorting		**ELiC with Morton-Order**		**Latency Reduction**	
	enc (s)	**dec (s)**	**enc (s)**	**dec (s)**	**enc (s)**	**dec (s)**
16 bit	0.192	0.186	0.172	0.157	-0.020	-0.029
15 bit	0.171	0.157	0.145	0.135	-0.026	-0.022
14 bit	0.143	0.126	0.121	0.112	-0.022	-0.014
13 bit	0.113	0.097	0.095	0.086	-0.018	-0.011
12 bit	0.091	0.072	0.074	0.063	-0.017	-0.009
Avg.	0.142	0.128	0.121	0.111	-0.021	-0.017

E. Asynchronous Arithmetic Encoding for Latency Reduction

In the ELiC pipeline, the neural network inference and arithmetic encoding operate sequentially. This means the CPU must wait idly for the GPU to finish the neural network forward pass before the arithmetic encoding can begin, and conversely, the GPU remains idle while the CPU performs the encoding operation. To improve runtime efficiency, we implement an asynchronous pipeline that overlaps these two tasks on separate compute threads.

Specifically, the stage-wise occupancy probabilities $\{\mathbf{P}_s^{(b)}\}_{s=1,2}$ at bit-depth b are emitted to a CPU buffer immediately upon computation. While the GPU advances to bit-depth $b+1$, the CPU arithmetic-encodes the symbols from level b. This producer-consumer pipeline overlaps neural network inference with entropy coding, reducing

end-to-end encoding time without changing the bitstream or compression efficiency.

Table S2 compares end-to-end encoding latency with and without asynchronous arithmetic encoding. In our measurements, this asynchronous execution achieves an average reduction of 12% in end-to-end encoding latency. As shown in Fig. S4, applying asynchronous arithmetic encoding increases end-to-end encoding throughput by more than 1 FPS for both ELiC and ELiC-Large on 12-bit-depth LiDAR geometries. This demonstrates that effective scheduling of the encoding pipeline, rather than modifying the model architecture, can further reduce latency.

Note that the runtime numbers reported in the main paper do not include this additional speed-up from asynchronous arithmetic encoding.

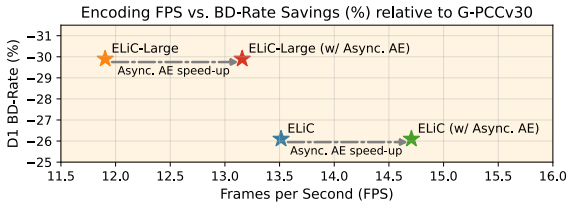


Figure S4. Speed-up achieved by applying asynchronous arithmetic encoding to ELiC.

Table S2. Encoding latency reduction (seconds per frame) achieved by applying asynchronous arithmetic encoding to ELiC.

	ELiC	ELiC w/ Async. AE	ELiC-Large	ELiC-Large w/ Async. AE
16 bit	0.172	0.145	0.198	0.170
15 bit	0.145	0.125	0.162	0.141
14 bit	0.121	0.105	0.140	0.117
13 bit	0.095	0.086	0.105	0.094
12 bit	0.074	0.068	0.084	0.076
Avg.	0.121	0.106	0.138	0.120

F. Model Size and Parameter Efficiency

Table S3 reports the number of model parameters and the relative model size of RENO, RENO-Large, ELiC w/o BoE, ELiC, and ELiC-Large, all normalized by the RENO baseline. The reported BD-Rate values are computed on the SemanticKITTI dataset in terms of D1-PSNR, as RENO-Large did not exhibit competitive performance on the Ford dataset, and SemanticKITTI was therefore used for comparison. Compared to RENO, which is a very small model with 0.29M parameters, ELiC w/o BoE remains very small—incurring only a 17% increase in the number of parameters—yet achieves a 6.61% BD-Rate saving. ELiC and ELiC-Large fall into the small-to-medium model regime with 2.02M and 8.04M parameters, respectively, and achieve D1-PSNR BD-Rate savings of 10.53% and 15.63% over RENO. RENO-Large reduces the bitrate by 13.43% relative to RENO, but its 20.55M parameters make it 2.56 times larger than ELiC-Large. Overall, these results indicate that the ELiC family uses model capacity more efficiently than RENO while achieving better compression performance.

Table S3. Number of model parameters and relative size.

Model	# Parameters (Millions)	Relative Ratio	D1 BD-Rate (%) (SemanticKITTI, vs. RENO)
RENO	0.29 M	1.00×	-
RENO-Large	20.55 M	71.21×	-13.43%
ELiC w/o BoE	0.34 M	1.17×	-6.61%
ELiC ($K = 5$)	2.02 M	6.99×	-10.53%
ELiC-Large ($K = 5$)	8.04 M	27.86×	-15.63%

G. Rate-Distortion Curves

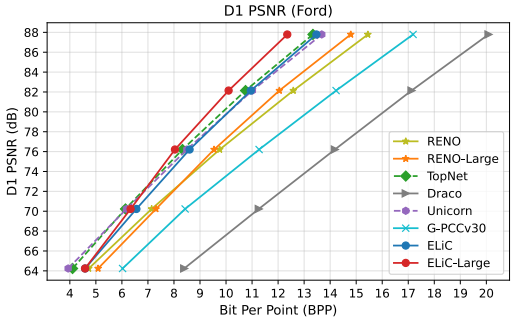
Fig. S5 shows rate-distortion (R-D) curves derived from Table 2 in the main paper (Sec. 5.3), used for the compression efficiency comparison. Each curve contains five points representing the rate-distortion performance for input LiDAR geometries at 16, 15, 14, 13, and 12 bit-depths.

ELiC markedly improves R-D performance over the RENO baseline, and ELiC-Large achieves compression performance comparable to Unicorn and TopNet. Referencing Sec. 5.2, RENO-Large, Unicorn, and TopNet operate at least 3× higher time complexity than ELiC variants for both encoding and decoding, indicating that ELiC offers a substantially better compression-runtime trade-off and stronger practical deployability.

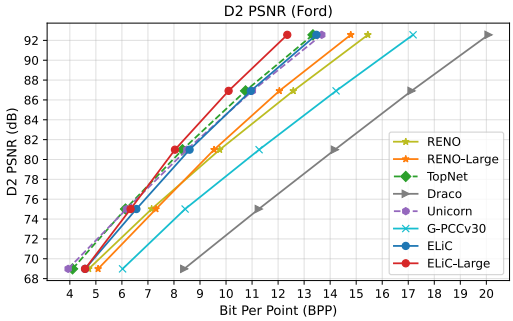
At the 16- and 15-bit coordinate inputs, the ELiC variants exhibit notably superior compression performance. However, at 13- and 12-bit coordinate inputs, a slight degradation is observed. This trend likely stems from the design of the training objective in Eq. 10, which minimizes the total bit count for the entire hierarchy. As a result, the model tends to allocate more capacity to support the highly sparse upper bit-depths, where a larger number of bits is required. Future work should explore strategies to balance optimization across all bit-depth levels for more uniformly strong performance.

H. Additional Bit-Depth-Wise Results

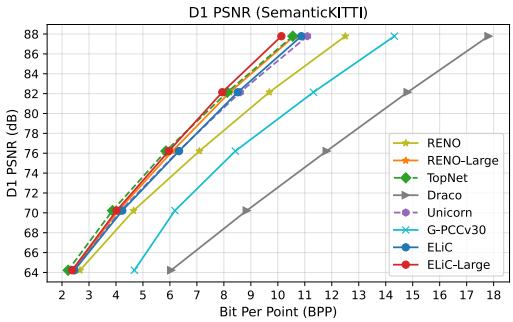
Fig. S6 visualizes per-point bit cost when compressing a 16-bit LiDAR frame with RENO, ELiC, and ELiC-Large. The per-point bit cost at bit-depth level b is the estimated code length from Eq. (10), i.e., for point n it equals $-\log_2 \mathbf{P}^{(b)}_{s=1,k}\big(n, \mathbf{Q}^{(b)}_{s=1}(n)\big) - \log_2 \mathbf{P}^{(b)}_{s=2,k}\big(n, \mathbf{Q}^{(b)}_{s=2}(n)\big)$. We show levels $b \in \{15, 13, 11, 9, 7\}$ and annotate the total bits for each level. RENO allocates similar totals at b=13 and b=15 (about 240.6 vs. 248.5 KBits), which indicates weak context exploitation in the sparse regime and limited



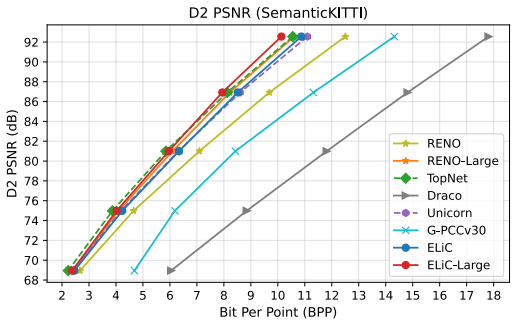
(a) BPP vs. D1 PSNR (Ford)



(b) BPP vs. D2 PSNR (Ford)



(c) BPP vs. D1 PSNR (SemanticKITTI)



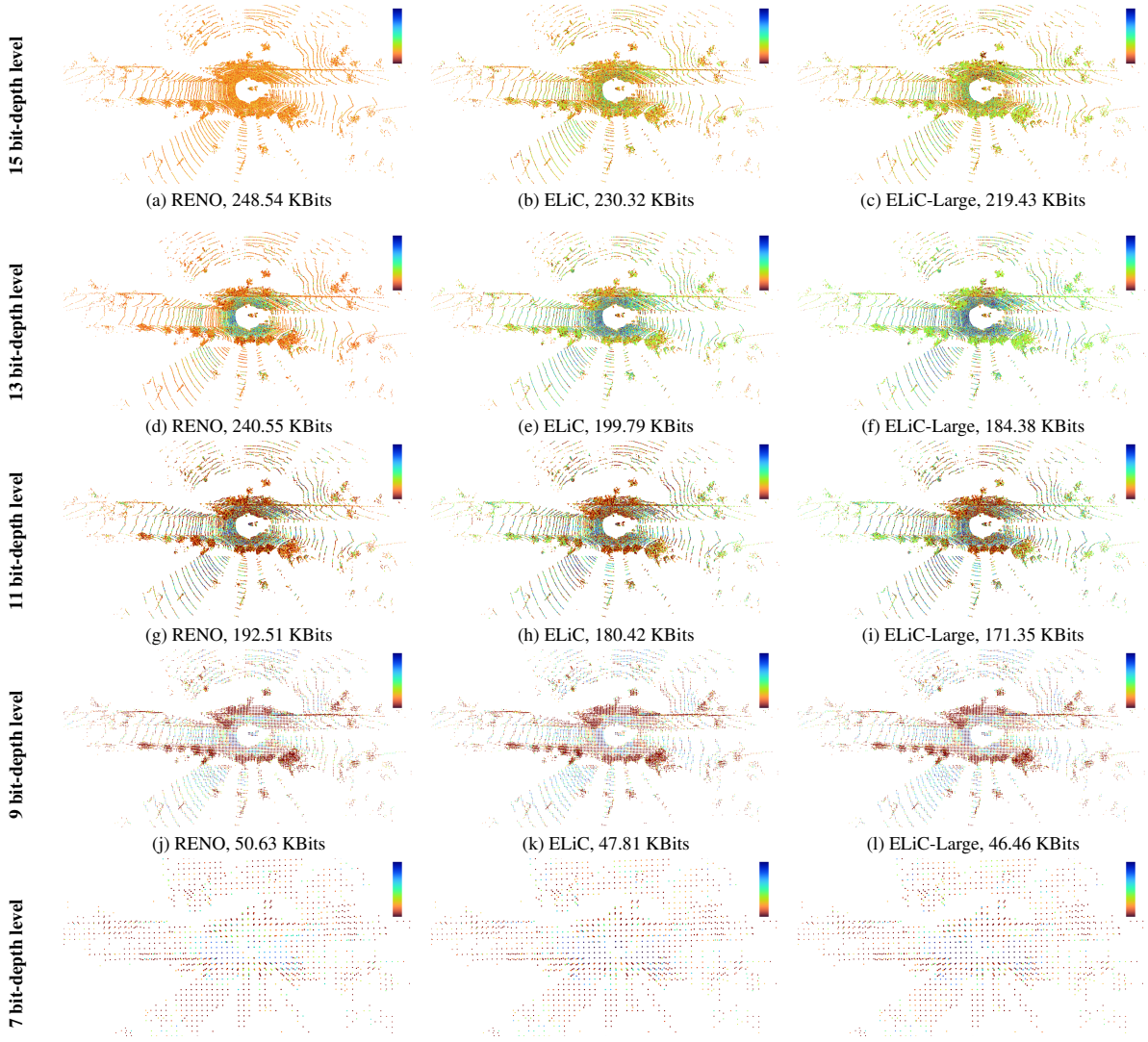
(d) BPP vs. D2 PSNR (SemanticKITTI)

Figure S5. Rate-distortion curves from Table 2 in Sec. 5.3

occupancy prediction accuracy. ELiC and ELiC-Large yield lower bit costs at b=13 and b=15 bit-depth levels and maintain better compression performance in far-range areas.

Across all three models, the gap between ground and non-ground grows as bit-depth level decreases and density increases. Non-ground regions are complex and irregular, which aligns poorly with a fixed 4+4 two-stage factorization. The bit grouping does not always match spatial correlation, and the best factorization likely depends on density and local structure.

At present, all BoE coding networks use the same 4+4 split. A natural extension is to let BoE select not only the network parameters but also the symbol factorization, so that different occupancy distributions can choose a more suitable split per level.



(a) RENO, 248.54 KBits (b) ELiC, 230.32 KBits (c) ELiC-Large, 219.43 KBits

(d) RENO, 240.55 KBits (e) ELiC, 199.79 KBits (f) ELiC-Large, 184.38 KBits

(g) RENO, 192.51 KBits (h) ELiC, 180.42 KBits (i) ELiC-Large, 171.35 KBits

(j) RENO, 50.63 KBits (k) ELiC, 47.81 KBits (l) ELiC-Large, 46.46 KBits

(m) RENO, 9.34 KBits (n) ELiC, 8.87 KBits (o) ELiC-Large, 8.86 KBits

Figure S6. Per-point bit allocation on the "Ford_03_vox1mm-0200.ply" frame at the 15, 13, 11, and 9 bit-depth levels for RENO, ELiC, and ELiC-Large. For each bit-depth level, the colormap is normalized using the minimum and maximum predicted bits among the three models. The colormap ranges from lower predicted bits (blue) to higher predicted bits (dark orange) per point.