ELiCv2: Learned LiDAR Geometry Compression under Real-Time Embedded Constraints

Anonymous authors
Paper under double-blind review

Abstract

Improving learned LiDAR compression increasingly relies on heavier inference, making it harder to meet the constraints of real-time embedded systems. We present ELiCv2, which removes two sources of redundant computation in progressive coding. Its discrete entropy interface selects a precomputed integer CDF instead of materializing one from a per-voxel distribution, while its occupancy-driven kernel-map builder derives finer-level sparse-convolution kernel-maps directly from the decoded hierarchy, eliminating coordinate-based neighbor search. ELiCv2 reinvests the recovered computation budget in lightweight ego-motion-aware temporal conditioning that gathers local occupancy context from the aligned previous reconstruction. On four LiDAR datasets and an NVIDIA Jetson Thor, ELiCv2 achieves a mean BD-rate reduction of 30.97% over G-PCC with channel width $C{=}32$ while sustaining the 20Hz nuScenes capture rate. Increasing the channel width to $C{=}64$ improves the mean reduction to 35.22%. Code will be released upon publication.

1 Introduction

LiDAR geometry compression is increasingly important for applications such as autonomous driving, robotics, V2X, digital twins, and large-scale 3D mapping (Mounier et al., 2024, Lin et al., 2025, Wang et al., 2025c, 2026b, Guang et al., 2026). These applications produce high-rate point clouds that must be transmitted under strict bandwidth and computational constraints. In practical edge–server deployments, LiDAR data are captured on resource-constrained platforms such as robots, drones, and autonomous vehicles, while downstream perception is often executed on more capable servers (Zhang et al., 2021, Luo et al., 2023). This asymmetry calls for lightweight encoders that can operate under limited compute budgets while keeping pace with high-rate LiDAR sensors.

Recent learned codecs substantially improve compression efficiency. Hybrid voxel–octree methods such as RENO (You et al., 2025) and ELiC (Kim et al., 2026) balance compression performance and runtime, achieving real-time throughput on desktop GPUs. Other approaches, including TopNet (Wang et al., 2025b), AnyPcc (Wang et al., 2026a), and Unicorn (Wang et al., 2025a), exploit attention-based or multistage context modeling to improve compression efficiency. Despite these advances, achieving strong compression performance while sustaining real-time encoding under embedded compute constraints remains challenging.

This challenge motivates us to examine how learned codecs spend their limited computation budget. Beyond context prediction, progressive occupancy coding also incurs costs when converting network outputs into coding distributions and constructing sparse-convolution kernel-maps. A conventional neural entropy interface converts each categorical output into an integer CDF for entropy coding. Sparse-convolution frameworks such as SpConv, TorchSparse++, and MinkowskiEngine construct kernel-maps through coordinate-based lookup (SpConv Contributors, 2022, Tang et al., 2023, Choy et al., 2019), even when decoded occupancy and the parent kernel-map already determine the child kernel-map. These operations suggest an opportunity to recover computation for prediction by eliminating repeated distribution conversion and exploiting the hierarchy exposed by progressive decoding.

ELiCv2 eliminates both forms of avoidable computation. The network directly selects a coder-ready distribution from precomputed integer CDFs, while decoded occupancy and the parent kernel-map

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

determine the child kernel-map without coordinate search at finer levels. Together, these mechanisms reduce the time spent preparing entropy-coder inputs and constructing kernel-maps, leaving more of the runtime budget available for context prediction. ELiCv2 uses this budget to exploit redundancy between consecutive LiDAR frames through a lightweight local gate conditioned on the ego-motion-aligned previous decoded frame. This adds temporal context without motion search or a separate temporal backbone.

As shown in Figure 1, ELiCv2 achieves the fastest learned encoding among the evaluated methods on nuScenes using an NVIDIA Jetson Thor. Its $C{=}32$ inter model sustains the 20Hz sensor rate and achieves a 43.15% BD-rate reduction relative to G-PCC intra on this dataset.

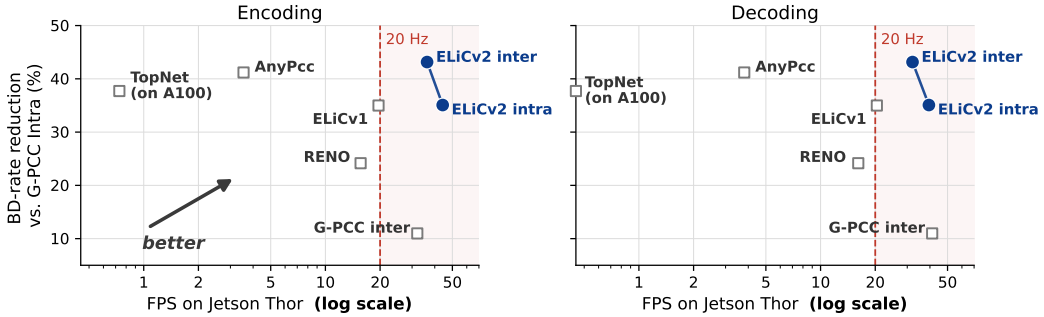

Figure 1. Compression efficiency versus throughput on an NVIDIA Jetson Thor. ELiCv2 reallocates computation from avoidable coding overhead to temporal context while sustaining the 20Hz nuScenes sensor rate. Throughput is measured at bit-depth 14, and BD-rate over bit-depths $16, 15, 14, 13, 12$.

Our main contributions are listed below.

- We identify two sources of avoidable computation in the entropy and sparse-processing interfaces of progressive coding.
- We replace per-voxel CDF materialization and coordinate search with direct selection of precomputed integer CDFs and occupancy-driven kernel-map propagation.
- We reinvest the recovered budget in lightweight ego-motion-aware temporal coding that gathers local context from the aligned previous decoded frame without motion search or complex temporal processing.

2 Related Work

Learned LiDAR geometry compression. Learned point cloud geometry codecs span range-image, octree, and sparse-voxel representations. Range-image methods (Zhou et al., 2022, Liu et al., 2023, Zhao et al., 2022) exploit efficient 2D processing but can lose geometric information through projection. Octree-based methods (Fu et al., 2022, Jin et al., 2024, Que et al., 2021, Wang et al., 2025b) model hierarchical occupancy with increasingly powerful context models, achieving strong compression at the cost of sequential or autoregressive inference. Sparse-convolutional methods (Wang et al., 2022, Wang & Gao, 2025) operate directly on voxelized geometry and provide greater parallelism, although sparse processing can introduce substantial computational overhead.

Hybrid voxel–octree codecs such as RENO (You et al., 2025) and ELiC (Kim et al., 2026) combine sparse-convolutional context modeling with progressive occupancy coding, providing a favorable balance between compression efficiency and runtime. ELiCv2 builds on this direction while explicitly targeting embedded inference. It replaces generic entropy-coding and sparse-convolution interfaces with coder-ready table selection and hierarchy-aware kernel-map propagation rather than increasing entropy-model complexity.

Inter-frame point cloud compression. Learned inter-frame codecs exploit temporal redundancy through motion estimation or cross-frame feature alignment. D-DPCC (Fan et al., 2022) predicts feature-space motion and interpolates reference features, while Akhtar et al. (Akhtar et al., 2024)

learn cross-frame correspondence using generalized sparse convolution. Both primarily target dense dynamic point clouds. Unicorn (Wang et al., 2025a) supports inter-frame geometry coding for diverse point clouds, including LiDAR, using a multiscale reference pyramid and sparse target convolutions. However, its wide $9\times9\times9$ cross-frame operators incur substantial inference cost.

RIDDLE (Zhou et al., 2022) incorporates sensor poses into temporal range prediction by querying neighboring points from the previous scan and reprojecting them into the current laser shot's coordinate system.

ELiCv2 uses ego-motion-aligned references within progressive occupancy coding. Since the current-level voxel coordinates are already known, temporal context is gathered from fixed $3\times3\times3$ neighborhoods in the aligned reference occupancy pyramid and injected through a lightweight local gate. This avoids the estimated query positions and point-neighbor search used by RIDDLE, while preserving parallel processing across voxels within each coding stage.

Kernel-map construction for sparse convolution. Sparse convolution requires a kernel-map specifying neighboring active voxels for each kernel offset. General-purpose engines such as MinkowskiEngine (Choy et al., 2019), SpConv (SpConv Contributors, 2022), and TorchSparse++ (Tang et al., 2023) construct these maps from sparse coordinates, typically through hashing or coordinate search. These engines support arbitrary sparse tensors and therefore reconstruct kernel-maps as active coordinates change. Progressive occupancy coding provides additional structure because finer-level voxels are deterministically generated from decoded parent occupancies. ELiCv2 exploits this hierarchy to propagate kernel-maps across octree levels, avoiding repeated coordinate-based connectivity search.

3 METHOD

3.1 PROGRESSIVE STRUCTURE AND DESIGN PRINCIPLE

ELiCv2 builds upon the progressive occupancy-coding framework of ELiC (Kim et al., 2026), retaining its sparse-convolutional backbone and cross-bit-depth feature propagation. A point cloud quantized to bit-depth B is represented as an octree and encoded coarse-to-fine over levels $\ell = 2, \ldots, B-1$. At each level, an occupied voxel $v \in \mathcal{V}_\ell$ is represented by an 8-bit occupancy symbol $o_v \in \{1, \ldots, 255\}$ indicating its occupied children. Once level ℓ is decoded, the coordinates at level $\ell+1$ are reconstructed deterministically, providing identical multiscale contexts to the encoder and decoder.

Following ELiC, each occupancy symbol is factorized into two sequential 4-bit symbols, reducing entropy prediction to two 16-way classification problems. The two stages are sequential, but all voxels within each stage are processed in parallel rather than through a voxel-by-voxel autoregressive chain. Unlike the low/high nibble split of ELiC, we partition the eight octants according to the parity of $x+y+z$. For octant index $i = 4z+2y+x$, with $(x,y,z) \in \{0,1\}^3$, we use

$$s_0 = \{1,2,4,7\}, \qquad s_1 = \{0,3,5,6\}. \tag{1}$$

This octant parity partition makes each second-stage octant face-adjacent to three first-stage octants, providing local conditioning without additional coding stages. Appendix A shows that it reduces rate at all 20 evaluated dataset and bit-depth combinations relative to the planar split, with a mean reduction of 0.125 bpp.

Progressive decoding exposes discrete structure that generic neural interfaces do not preserve. Once an occupancy symbol is decoded, the child support and parent–child relations are known deterministically. ELiCv2 uses this hierarchy to derive finer-level kernel-maps without coordinate search. The same principle applies at the entropy interface, where the network directly selects a coder-ready integer CDF instead of materializing one from each categorical output. By removing these two sources of interface overhead without reducing the ELiC backbone, ELiCv2 recovers computation for lightweight temporal context.

Figure 2 summarizes how the discrete entropy interface, occupancy-driven kernel-map propagation, and ego-motion-aligned reference gating are integrated into the progressive backbone.

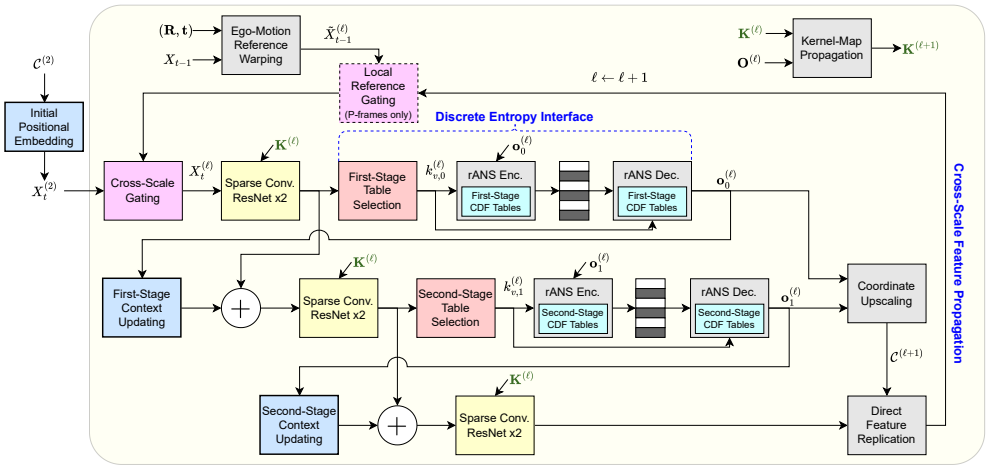

Figure 2. Overview of ELiCv2. At each octree level, the backbone selects indexed CDF tables for two-stage occupancy coding. The decoded occupancy and current kernel-map determine the next-level kernel-map. P-frames additionally use local gating over an ego-motion-aligned reference.

3.2 Discrete Entropy Interface

Conventional neural entropy models output a categorical probability distribution over occupancy symbols for each voxel. Before encoding or decoding a symbol, these probabilities must be quantized into integer frequencies and accumulated into a finite-precision cumulative distribution function (CDF) required by the entropy coder. This probability-to-coder materialization incurs substantial preparation and data-transfer overhead even though rANS (Duda et al., 2015) ultimately consumes a discrete coding rule. ELiCv2 redesigns this interface. It learns a finite codebook of distributions and predicts which coder-ready CDF table should be used.

We retain the Bag-of-Encoders (BoE) scheme of ELiCv1 (Kim et al., 2026), using a total of $K_{\mathrm{BoE}}{=}4$ coding networks. BoE selects a coding network at each bit depth according to occupancy statistics. The selected network then predicts a CDF-table index for each node.

Within each coding network, the codebook is parameterized using row and column sets of T learnable logits, $\mathbf{L}^{\mathrm{row}}, \mathbf{L}^{\mathrm{col}} \in \mathbb{R}^{T\times 16}$, low-rank interaction factors $\boldsymbol{\alpha}^{\mathrm{row}}, \boldsymbol{\alpha}^{\mathrm{col}} \in \mathbb{R}^{T\times r}$, and a shared basis $\mathbf{B} \in \mathbb{R}^{r\times 16}$. Each pair (i, j) defines

$$\mathbf{p}_{ij} = \mathrm{softmax}\big(\mathbf{L}^{\mathrm{row}}_i + \mathbf{L}^{\mathrm{col}}_j + (\boldsymbol{\alpha}^{\mathrm{row}}_i \odot \boldsymbol{\alpha}^{\mathrm{col}}_j)\mathbf{B}\big)\,. \tag{2}$$

The low-rank interaction provides T^2 expressive distributions without independently parameterizing all table pairs.

For each voxel v, a two-layer MLP selection head outputs $2T$ logits, grouped into row and column vectors $\mathbf{z}^{\mathrm{row}}_v, \mathbf{z}^{\mathrm{col}}_v \in \mathbb{R}^T$. The table indices are selected as

$$i_v = \operatorname*{argmax}_i \mathbf{z}^{\mathrm{row}}_{v,i}, \qquad j_v = \operatorname*{argmax}_j \mathbf{z}^{\mathrm{col}}_{v,j}, \qquad k_v = Ti_v + j_v, \tag{3}$$

where $i_v, j_v \in \{0, \ldots, T-1\}$ are zero-based row and column indices, and $k_v \in \{0, \ldots, T^2-1\}$ is the flattened index identifying the selected distribution $\mathbf{p}_{i_v j_v}$ for voxel v. After training, all T^2 distributions are quantized into fixed integer CDFs shared by the encoder and decoder. The index k_v identifies the CDF used by indexed rANS to encode or decode the occupancy symbol of voxel v for the current coding network and stage. With $T = 32$, this gives 1024 CDF tables. Appendix C examines the rate–storage trade-off underlying this choice. At inference, the network therefore selects a discrete coding distribution directly. Entropy preparation becomes top-1 selection and an integer table lookup followed by indexed rANS coding, eliminating per-voxel CDF materialization. We use a Gumbel–Softmax relaxation (Jang et al., 2016) during training and hard top-1 selection at inference.

3.3 Occupancy-Driven Kernel-Map Propagation

General sparse engines assume that each active support is arbitrary. They therefore recover $3\times3\times3$ connectivity through coordinate hashing or neighbor search whenever the support changes. This generic interface discards the hierarchy exposed by progressive occupancy coding. Every finer-level voxel is generated from a known parent according to decoded occupancy, so child connectivity is already determined by parent connectivity and occupancy masks. ELiCv2 reformulates kernel-map construction from coordinate search into hierarchical kernel-map propagation.

Let $\mathbf{K}^p \in \mathbb{Z}^{N_p\times 27}$ be the parent kernel-map and $m_i \in \{0,\dots,255\}$ the 8-bit integer occupancy mask of parent i. We use $m_i[q] \in \{0,1\}$ to denote its bit at octant q. An active child in octant q receives the index

$$\mathrm{idx}(i,q) = \mathrm{base}(i) + \mathrm{popcount}\left(m_i \wedge (2^q - 1)\right), \tag{4}$$

where $\mathrm{base}(i)$ is an exclusive prefix sum over the numbers of active children.

For each child octant q and kernel offset k, the corresponding neighboring parent and target child octant depend only on (q,k). We precompute these relations as $T_{\mathrm{col}}(q,k)$ and $T_{\mathrm{bit}}(q,k)$. The child kernel-map is then obtained directly from the parent map and occupancy.

$$\mathbf{K}^c[u,k] = \begin{cases} \mathrm{idx}(j,q'), & j = \mathbf{K}^p[i, T_{\mathrm{col}}(q,k)] \geq 0 \text{ and } m_j[q'] = 1, \\ -1, & \text{otherwise}, \end{cases} \tag{5}$$

where $u = \mathrm{idx}(i,q)$ and $q' = T_{\mathrm{bit}}(q,k)$.

Thus, the progressive coarse-to-fine structure turns kernel-map construction from coordinate search into deterministic kernel-map propagation using only prefix sums, table lookups, and bit operations. The key improvement is not faster search. Coordinate search is unnecessary at progressively generated finer levels and is eliminated entirely at those levels. The same principle is not specific to the $3\times3\times3$ stencil used in ELiCv2 and extends to larger kernel neighborhoods, as demonstrated in Appendix E. The full derivation of the propagation rule is provided in Appendix D.

3.4 Ego-Motion-Aware Inter Coding

After eliminating redundant entropy and sparse-processing work, we reinvest the recovered computation budget in predictive context. For a P-frame, the previous reconstructed frame is aligned to the current frame using sensor ego-motion and used as additional context. Since geometry coding is lossless, the encoder already holds the same previous reconstruction available to the decoder, requiring neither closed-loop decoding nor an additional temporal backbone.

Reference construction. Let X_t and X_{t-1} denote the current and previous voxelized frames. We align X_{t-1} to the current coordinate system as

$$\tilde{X}_{t-1} = \mathcal{W}_{t-1\to t}(X_{t-1}), \qquad \mathcal{W}(\mathbf{x}) = \left\lfloor \frac{\mathbf{R}\mathbf{x} + \mathbf{t} + 2^{29}}{2^{30}} \right\rfloor, \tag{6}$$

where $(\mathbf{R}, \mathbf{t})$ is the relative ego transform quantized to 30-bit fixed-point parameters and transmitted in the bitstream. Both encoder and decoder therefore construct an identical reference using integer arithmetic. The reported inter-frame bitrates include the uncompressed 96-byte P-frame header containing the nine entries of $\mathbf{R}$ and three entries of $\mathbf{t}$ as 64-bit integers. The aligned frame is downsampled to form the multiscale reference pyramid $\tilde{X}^{(\ell)} = (\tilde{\mathcal{C}}^{(\ell)}, \tilde{O}^{(\ell)})$.

Local reference gating. Exact voxel correspondence is unreliable because consecutive LiDAR sweeps sample slightly different spatial locations. For each current voxel, we therefore gather its $3\times3\times3$ reference neighborhood.

$$r_{v,i} = \begin{cases} \tilde{O}^{(\ell)}[\mathbf{x}_v + \boldsymbol{\delta}_i], & \mathbf{x}_v + \boldsymbol{\delta}_i \in \tilde{\mathcal{C}}^{(\ell)}, \\ \textsc{absent}, & \text{otherwise}, \end{cases} \qquad i = 1,\dots,27. \tag{7}$$

The gathered symbols are embedded and combined into a position-aware context,

$$\mathbf{h}_v^{\mathrm{ref}} = \mathbf{W}\left[E(r_{v,1}) \| \cdots \| E(r_{v,27})\right] + \mathbf{b}, \tag{8}$$

where $\|$ denotes feature concatenation and E is a learned embedding table shared across all 27 offsets, with a dedicated entry for ABSENT. Each block $\mathbf{W}_i$ of the projection $\mathbf{W}$ corresponds to one offset, preserving the relative spatial position of the embedded reference symbol. The embedding table and bias $\mathbf{b}$ are initialized to zero, so the temporal path initially contributes no reference context. The resulting context is injected into the propagated feature through

$$\mathbf{f}_v \leftarrow \mathbf{f}_v + \sigma\big(g([\mathbf{f}_v \| \mathbf{h}_v^{\mathrm{ref}}])\big) \odot \mathbf{h}_v^{\mathrm{ref}}. \tag{9}$$

where g is a two-layer MLP and σ denotes the sigmoid function. This position-aware gate is the only structural addition to the intra backbone. Temporal conditioning is restricted to $\ell \leq \ell_{\mathrm{ref}} = 13$, beyond which voxel-level correspondence rapidly deteriorates. Appendix F provides the alignment visualization and resolution-dependent analysis.

Overall, the inter model is both the payoff of structure-preserving inference and a lightweight temporal mechanism. It exploits pose-aligned redundancy using only local occupancy gathering and feature gating without motion search or complex temporal processing. Further design rationale and comparisons with cross-frame operators are provided in Appendix G.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate ELiCv2 on four LiDAR sequence datasets. SemanticKITTI (Behley et al., 2019), Ford (Schwarz et al., 2018), nuScenes (Caesar et al., 2020), and V2X-Seq (Yu et al., 2023). These datasets cover diverse acquisition settings, from moving vehicle-mounted LiDAR to stationary infrastructure LiDAR, allowing us to evaluate inter-frame coding under different ego-motion characteristics. Table 1 summarizes the dataset statistics.

For SemanticKITTI, we use sequences 00–10 for training and 11–21 for testing, with sensor poses obtained from the provided odometry and calibration. For Ford, we train on Ford_01 and evaluate on Ford_02 and Ford_03. Since ground-truth sensor poses are unavailable, inter-frame transforms are estimated using KISS-ICP (Vizzo et al., 2023). For nuScenes, we use the full 20Hz LiDAR sweep stream rather than the 2Hz annotated keyframes, with LiDAR poses obtained by composing the provided ego poses and sensor extrinsics. V2X-Seq consists of stationary roadside LiDAR, for which the relative sensor transform is the identity.

Training. We train a separate model for each dataset at the finest operating point ($\mathrm{posQ} = 4$, corresponding to a 16-bit octree) and use the same model for all coarser rate points. Since ELiCv2 performs lossless geometry coding on the quantized voxel grid, no distortion term is required and the training objective directly minimizes the expected code length. Specifically, for a voxel v at octree level ℓ and coding stage $s \in \{0, 1\}$, we minimize

$$\mathcal{L} = \frac{1}{N} \sum_{\ell} \sum_{v \in \mathcal{V}_\ell} \sum_{s \in \{0,1\}} -\log_2 \mathbf{p}_{v,s}[o_{v,s}]\,, \tag{10}$$

where $o_{v,s} \in \{0, \ldots, 15\}$ is the target 4-bit occupancy symbol, $\mathbf{p}_{v,s}$ is the corresponding 16-way distribution produced by the entropy model, and N is the number of input points.

The intra model is trained for 300k steps using Adam (Kingma & Ba, 2014) with a batch size of 4 and a cosine learning-rate schedule from 5×10^{-4} to 5×10^{-6}. Since the hard CDF-table selection in Section 3.2 is non-differentiable, we use a Gumbel–Softmax relaxation (Jang et al., 2016) during training and anneal its temperature from $\tau = 2.0$ to $\tau = 0.1$.

The inter model is initialized from the corresponding trained intra model and jointly fine-tuned for 200k steps. We use Adam with a cosine learning-rate schedule from 1×10^{-4} to 5×10^{-6} and fix the Gumbel–Softmax temperature at $\tau = 0.1$. Each training sample consists of consecutive LiDAR frames, with four P-frames conditioned on their respective preceding frames.

Evaluation metrics. We evaluate compression performance using bits per input point (bpp) and D1-PSNR. For all operating points, bpp is normalized by the number of occupied voxels at the finest quantization level ($\mathrm{posQ} = 1$), providing a fixed denominator across rate points. Since all evaluated

Table 1. Dataset statistics. Points denotes the mean number of points per test frame, and ego step is the median translation between consecutive sensor poses.

Dataset	Train	Test	Points	Ego step	Setting
SemanticKITTI	23,201	20,351	120,557	0.932 m	Vehicle, 10Hz
Ford	1,500	3,000	83,927	0.634 m	Vehicle
nuScenes	297,737	52,423	34,722	0.314 m	Vehicle, 20Hz
V2X-Seq	11,275	4,862	56,273	0	Infrastructure

codecs are lossless on the quantized voxel grid, distortion at a given bit-depth is determined solely by quantization and is identical across methods.

We summarize rate–distortion performance using BD-rate (Bjøntegaard, 2001) with G-PCC intra as the anchor. Unless otherwise stated, compression results are evaluated over the complete test split of each dataset.

For inter-frame coding, we use a GOP size of 30 with an IPPPP structure. The first frame is coded by the intra model, and each of the following 29 P-frames references its immediate predecessor. GOPs never cross sequence boundaries.

Implementation and hardware. ELiCv2 is implemented in PyTorch (Paszke et al., 2019) with sparse-convolution operations based on TorchSparse++ (Tang et al., 2023), and uses 16-bit rANS for entropy coding. The conventional baselines use the G-PCC standard (Li et al., 2024) and its TM13v34 test model (WG 07 MPEG 3D Graphics Coding and Haptics Coding, 2026). All models are evaluated in FP32 and trained on NVIDIA A100 GPUs. Runtime measurements are conducted on an NVIDIA Jetson Thor (NVIDIA Corporation, 2025), our target embedded platform, under the 120 W nvpmodel power profile, with the CPU and GPU clocks fixed at 2,601 MHz and 1,386 MHz, respectively. Encoding and decoding latency excludes file I/O. All methods use the same Jetson Thor hardware and power settings, except TopNet, which is measured on an A100 because it does not fit on the Jetson.

4.2 Compression Performance

Table 2 compares the compression performance of ELiCv2 with TopNet (Wang et al., 2025b), RENO (You et al., 2025), AnyPcc (Wang et al., 2026a), and ELiCv1 (Kim et al., 2026), together with the G-PCC baselines. The $C{=}32$ intra model achieves a mean BD-rate reduction of 26.66% over G-PCC, which improves to 30.97% with inter coding. Increasing the model capacity to $C{=}64$ further improves the mean reduction from 31.47% for intra coding to 35.22% for inter coding.

Table 2. BD-rate (%) relative to G-PCC intra over bit-depths $\{16, 15, 14, 13, 12\}$. Negative values indicate better compression. † TopNet is evaluated on a 600-frame sample. Best results are in **bold**.

Method	nuScenes	Sem.KITTI	Ford	V2X-Seq	Mean
G-PCC (intra)	0.00	0.00	0.00	0.00	0.00
G-PCC (inter)	-10.98	-1.64	-4.80	-4.25	-5.42
TopNet†	-37.71	-31.66	**-30.00**	-19.59	-29.74
RENO	-24.16	-17.11	-14.53	-12.28	-17.02
AnyPcc	-41.19	-32.57	-28.32	-29.40	-32.87
ELiCv1	-34.99	-26.06	-23.72	-25.40	-27.54
Ours ($C{=}32$, intra)	-35.08	-24.62	-23.57	-23.39	-26.66
Ours ($C{=}32$, inter)	-43.15	-28.27	-27.35	-25.09	-30.97
Ours ($C{=}64$, intra)	-40.47	-29.09	-27.03	-29.27	-31.47
Ours ($C{=}64$, inter)	**-48.24**	**-32.61**	-29.73	**-30.32**	**-35.22**

The $C{=}32$ configuration provides a strong operating point for embedded deployment. Its inter model achieves a mean BD-rate reduction of 30.97%, outperforming ELiCv1 (27.54%), RENO

(17.02%), and the sampled TopNet result (29.74%). The improvement is particularly large on nuScenes, where ELiCv2 C=32 inter reduces BD-rate by 43.15%. As shown in Section 4.3, these gains are obtained while retaining the low latency targeted by ELiCv2.

The larger C=64 model provides a compression-oriented operating point. With inter coding, it achieves the best results on nuScenes (−48.24%), SemanticKITTI (−32.61%), and V2X-Seq (−30.32%), while remaining competitive with TopNet on Ford. Its mean BD-rate reaches −35.22%, improving upon AnyPcc (−32.87%), the strongest competing method on average.

Inter coding consistently improves both model capacities across all four datasets. Relative to their intra models, the C=32 and C=64 inter models provide additional mean BD-rate reductions of 4.31 and 3.75 percentage points, respectively. This consistent improvement shows the payoff of computation reallocation. After removing entropy preparation and kernel-map construction overhead, ELiCv2 adds useful temporal context without motion search or complex temporal processing.

4.3 Runtime Performance

Table 3. Encoding and decoding latency (ms/frame) on NVIDIA Jetson Thor with I/O excluded. Green cells indicate real-time operation at the corresponding sensor rate (20Hz for nuScenes and 10Hz for the other datasets). † TopNet is measured on an A100 because it does not fit on the Jetson.

		nuScenes					SemanticKITTI					Ford					V2X-Seq				
Method		16	15	14	13	12	16	15	14	13	12	16	15	14	13	12	16	15	14	13	12
G-PCC (intra)	enc	38.7	31.1	24.7	19.3	13.7	140.0	112.5	83.4	54.8	31.2	107.6	91.2	72.9	55.6	38.0	90.8	78.4	65.9	52.8	40.1
	dec	28.3	22.7	17.7	13.1	8.8	103.1	79.9	55.8	34.2	18.4	82.0	66.9	52.4	37.5	23.9	70.6	59.8	48.7	38.0	27.0
G-PCC (inter)	enc	45.7	38.6	31.2	24.9	17.7	163.7	135.2	104.1	71.5	41.7	125.4	107.1	89.2	69.8	50.1	104.0	91.0	77.9	64.7	51.1
	dec	36.4	30.6	24.3	18.9	14.3	129.4	103.7	77.6	51.5	29.5	101.9	85.8	69.7	54.3	38.8	85.2	73.6	62.4	50.4	39.0
TopNet†	enc	2249	1796	1362	968.4	641.7	7322	5352	3561	2044	1011	5411	4325	3280	2249	1314	4908	4087	3276	2426	1700
	dec	725K	594K	445K	313K	203K	1.59M	1.20M	787K	453K	222K	1.22M	983K	732K	495K	292K	1.08M	890K	702K	550K	366K
RENO	enc	97.0	81.0	64.0	51.0	41.0	246.0	187.0	129.0	84.0	50.0	204.0	168.0	132.0	96.0	66.0	186.0	158.0	132.0	103.0	78.0
	dec	88.0	76.0	62.0	50.0	37.0	242.0	186.0	128.0	81.0	46.0	205.0	170.0	134.0	97.0	65.0	178.0	151.0	124.0	99.0	73.0
AnyPcc	enc	404.8	341.1	282.2	227.8	176.6	1025	773.6	537.1	339.8	208.5	869.0	697.5	559.0	403.9	278.6	784.4	671.4	555.7	441.9	325.1
	dec	387.7	318.6	263.1	203.9	156.2	1043	787.3	543.9	350.1	198.7	846.1	684.8	527.8	376.1	251.6	789.6	660.9	545.1	415.4	317.1
ELiCv1	enc	77.0	64.0	51.0	42.0	33.0	204.0	155.0	110.0	73.0	48.0	164.0	135.0	106.0	79.0	54.0	152.0	129.0	107.0	85.0	64.0
	dec	73.0	62.0	49.0	41.0	33.0	207.0	157.0	111.0	69.0	42.0	170.0	139.0	109.0	82.0	54.0	150.0	126.0	107.0	82.0	62.0
ELiCv2 (C=32 intra)	enc	31.2	26.2	22.6	18.6	15.2	95.3	78.1	59.9	41.6	25.5	69.9	60.7	51.0	40.8	29.3	61.0	54.7	48.0	41.2	33.2
	dec	34.5	29.9	25.3	21.0	17.0	113.6	92.4	70.8	49.4	30.9	80.3	69.2	58.1	45.5	32.5	71.3	62.9	55.4	46.9	37.1
ELiCv2 (C=32 inter)	enc	34.8	31.2	27.6	23.5	19.6	101.1	85.0	67.5	47.5	31.0	75.2	66.7	57.8	46.9	34.2	65.6	59.5	53.5	47.4	38.4
	dec	40.3	36.0	31.2	25.9	21.4	118.3	97.8	76.3	52.6	33.6	89.1	77.9	66.2	52.4	37.6	77.8	69.8	62.1	53.7	43.3
ELiCv2 (C=64 intra)	enc	55.6	48.4	42.2	34.6	28.3	199.7	174.2	138.2	94.6	55.1	138.8	126.1	111.5	90.9	64.3	116.4	107.8	98.8	87.7	71.8
	dec	57.7	51.4	43.7	35.9	28.9	218.9	192.3	156.9	109.7	62.2	148.1	133.8	117.1	94.7	66.5	125.4	115.6	104.7	91.9	74.3
ELiCv2 (C=64 inter)	enc	59.8	54.6	48.1	40.2	33.2	208.6	184.3	149.6	102.1	60.4	147.1	135.3	121.6	99.0	70.2	122.7	114.9	106.3	95.9	78.3
	dec	65.1	58.7	50.6	42.2	34.4	230.7	199.1	161.7	109.8	62.6	164.3	148.1	130.8	105.9	73.3	134.3	124.1	113.6	101.6	81.9

Table 3 reports end-to-end latency on Jetson Thor. The C=32 intra model is faster than RENO, ELiCv1, and AnyPcc across all evaluated datasets and bit-depths. On nuScenes, the C=32 inter model encodes and decodes a 16-bit frame in 34.8 ms and 40.3 ms, respectively, meeting the 50 ms budget for 20Hz operation. At 14 bits, these times decrease to 27.6 ms and 31.2 ms. RENO, ELiCv1, and AnyPcc require 97.0 ms, 77.0 ms, and 404.8 ms for 16-bit encoding, respectively. ELiCv2 is thus the only evaluated learned codec to sustain 20Hz at the finest resolution.

At 16 bits, C=32 inter encoding takes 101.1 ms on SemanticKITTI, 75.2 ms on Ford, and 65.6 ms on V2X-Seq. Temporal conditioning adds only 3.6–5.8 ms over intra encoding across the four datasets. Increasing width to C=64 improves compression at additional runtime cost: nuScenes 16-bit inter encoding rises from 34.8 to 59.8 ms. We therefore use C=32 as the deployment-oriented configuration and C=64 as the rate-oriented configuration. Power, energy, and memory measurements appear in Appendices H and I.

4.4 Ablation Studies

4.4.1 Discrete Entropy Interface

Table 4 compares continuous-CDF arithmetic coding (Witten et al., 1987) and rANS against discrete-table indexed rANS for the C=32 intra backbone. We average the four datasets at bit-depth 16 after the shared network prediction. Per-dataset results are in Appendix B.

Table 4. Entropy-coding ablation on Jetson Thor, averaged equally over the four datasets. *Prep.* denotes per-voxel CDF construction for continuous coding and top-1 table selection for indexed rANS. *D2H* denotes device-to-host transfer from the GPU to the CPU.

	Encoding time (ms/frame)					
Entropy coder	Prep.	D2H	Code	Total	× ours	bpp
Arithmetic coding	38.24	2.33	14.27	54.84	5.0×	12.7872 (−1.11%)
rANS	38.24	2.33	6.42	46.98	4.3×	12.7882 (−1.10%)
Indexed rANS	5.75	0.80	4.05	10.61	–	12.9271

Replacing arithmetic coding with rANS leaves the dominant 38.24 ms CDF construction unchanged. Discrete table selection reduces preparation to 5.75 ms and total entropy coding to 10.61 ms. D2H transfer also falls because each voxel transfers one table index instead of a complete CDF. Indexed rANS is $5.0\times$ faster than arithmetic coding and $4.3\times$ faster than continuous-CDF rANS, at approximately 1.1% rate cost. The gain comes primarily from the neural–coder interface.

4.4.2 Kernel-Map Propagation

Table 5. Kernel-map construction latency on Jetson Thor for the 3×3×3 kernel used in ELiCv2. Runtime includes all progressively constructed levels, with sparse convolution excluded. Both methods produce identical kernel-maps.

Dataset	Voxels	Ours (ms)	Hashing (ms)	Speed-up
SemanticKITTI	120,826	1.69	5.78	3.42×
Ford	80,002	1.46	5.16	3.54×
nuScenes	29,824	0.70	2.80	4.00×
V2X-Seq	63,416	1.27	4.51	3.55×
Mean		1.28	4.56	**3.63**×

Table 5 compares our progressive builder with TorchSparse++'s hash-based method. Both produce identical $3\times3\times3$ kernel-maps, but propagation reduces mean latency from 4.56 to 1.28 ms. The arithmetic mean of the per-dataset speedups is $3.63\times$. The 3.28 ms saving recovers 6.6% of the 50 ms budget for 20Hz operation, providing time for temporal conditioning. This gain comes from eliminating coordinate search at finer levels. Results for larger kernels appear in Appendix E.

5 Conclusion and Limitations

ELiCv2 improves embedded LiDAR geometry compression by redesigning the entropy-coding interface and exploiting the hierarchy exposed by progressive occupancy coding. The entropy model directly selects a coder-ready integer CDF, while the sparse engine derives finer-level kernel-maps from decoded occupancy and parent kernel-maps. Together, these mechanisms eliminate per-voxel CDF materialization and finer-level coordinate search. ELiCv2 uses the recovered computation budget for lightweight ego-motion-aware temporal conditioning. Across four LiDAR datasets, the C=32 inter model achieves a mean BD-rate reduction of 30.97% over G-PCC. On nuScenes, it sustains 20Hz encoding and decoding on an NVIDIA Jetson Thor. These results demonstrate that reducing computation at the entropy and sparse-processing interfaces can support effective temporal prediction under real-time embedded constraints.

The current implementation performs neural inference in FP32, leaving opportunities to reduce computation and memory through integer quantization. Temporal conditioning relies on accurate ego poses and sufficient local overlap with the previous decoded frame. Its benefit may diminish with pose errors, dynamic objects, and fine resolutions where LiDAR samples no longer align locally. Future work will investigate deterministic low-precision inference and temporal conditioning that remains lightweight while becoming more robust to these cases.

AI Assistance Disclosure

We used ChatGPT to assist with writing and polishing the manuscript, and Claude to assist with creating some of the figures.

References

Anique Akhtar, Zhu Li, and Geert Van der Auwera. Inter-frame compression for dynamic point cloud geometry coding. *IEEE Transactions on Image Processing*, 33:584–594, 2024.

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*, pp. 9297–9307, 2019.

Gisle Bjøntegaard. Calculation of average psnr differences between RD curves. VCEG Document VCEG-M33, ITU-T Video Coding Experts Group, Austin, TX, USA, April 2001.

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 *2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 11618–11628. IEEE, 2020.

Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3075–3084, 2019.

Jarek Duda, Khalid Tahboub, Neeraj J Gadgil, and Edward J Delp. The use of asymmetric numeral systems as an accurate replacement for huffman coding. In *2015 Picture Coding Symposium (PCS)*, pp. 65–69. IEEE, 2015.

Tingyu Fan, Linyao Gao, Yiling Xu, Zhu Li, and Dong Wang. D-dpcc: Deep dynamic point cloud compression via 3d motion prediction. In Lud De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 898–904. International Joint Conferences on Artificial Intelligence Organization, 7 2022. Main Track.

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*, volume 36, pp. 625–633, 2022.

Jinzheng Guang, Qianyi Zhang, and Jingtai Liu. Mdslca: Multi-scale dilated spatial and local channel attention for lidar point cloud semantic segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 2026.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

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)*, pp. 7985–7989. IEEE, 2024.

Junsik Kim, Gun Bang, and Soowoong Kim. Elic: Efficient lidar geometry compression via cross-bit-depth feature propagation and bag-of-encoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 39011–39020, June 2026.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ge Li, Wei Gao, and Wen Gao. Mpeg geometry-based point cloud compression (g-pcc) standard. In *Point Cloud Compression: Technologies and Standardization*, pp. 135–165. Springer, 2024.

R.D. Lin, Pengcheng Weng, Yinqiao Wang, Han Ding, Jinsong Han, and Fei Wang. Hilots: High-low temporal sensitive representation learning for semi-supervised lidar segmentation in autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1429–1438, June 2025.

Chia-Sheng Liu, Jia-Fong Yeh, Hao Hsu, Hung-Ting Su, Ming-Sui Lee, and Winston H Hsu. Birdpcc: Bi-directional range image-based deep lidar point cloud compression. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

Guiyang Luo, Chongzhang Shao, Nan Cheng, Haibo Zhou, Hui Zhang, Quan Yuan, and Jinglin Li. Edgecooper: Network-aware cooperative lidar perception for enhanced vehicular awareness. *IEEE Journal on Selected Areas in Communications*, 42(1):207–222, 2023.

Eslam Mounier, Mohamed Elhabiby, Michael Korenberg, and Aboelmagd Noureldin. Lidar-based multi-sensor fusion with 3d digital maps for high-precision positioning. *IEEE Internet of Things Journal*, 2024.

NVIDIA Corporation. NVIDIA Jetson AGX Thor. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-thor/`, 2025. Accessed: 2026-05-27.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Zizheng Que, Guo Lu, and Dong Xu. Voxelcontext-net: An octree based framework for point cloud compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6042–6051, June 2021.

Sebastian Schwarz, Gaëlle Martin-Cocher, David Flynn, and Madhukar Budagavi. Common test conditions for point cloud compression. *Document ISO/IEC JTC1/SC29/WG11 w17766, Ljubljana, Slovenia*, 2018.

SpConv Contributors. SpConv: Spatially sparse convolution library. `https://github.com/traveller59/spconv`, 2022.

Haotian Tang, Shang Yang, Zhijian Liu, Ke Hong, Zhongming Yu, Xiuyu Li, Guohao Dai, Yu Wang, and Song Han. Torchsparse++: Efficient training and inference framework for sparse convolution on gpus. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 225–239, 2023.

Ignacio Vizzo, Tiziano Guadagnino, Benedikt Mersch, Louis Wiesmann, Jens Behley, and Cyrill Stachniss. Kiss-icp: In defense of point-to-point icp – simple, accurate, and robust registration if done the right way. *IEEE Robotics and Automation Letters*, 8(2):1029–1036, 2023.

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.

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*, 47(1):269–287, 2025a.

Kangli Wang and Wei Gao. Unipcgc: Towards practical point cloud geometry compression via an efficient unified approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 12721–12729, 2025.

Kangli Wang, Qianxi Yi, Yuqi Ye, Shihao Li, and Wei Gao. Anypcc: Compressing any point cloud with a single universal model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2972–2982, 2026a.

Ke Wang, Weilin Gao, Kai Chen, Tianyi Shao, Liyang Li, Tianqiang Zhou, and Jianbo Lu. Pvfdectnet++: Adaptive multi-modal fusion with perspective voxels for 3d object detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 2026b.

Xinjie Wang, Yifan Zhang, Ting Liu, Xinpu Liu, Ke Xu, Jianwei Wan, Yulan Guo, and Hanyun Wang. Topnet: Transformer-efficient occupancy prediction network for octree-structured point cloud geometry compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 27305–27314, June 2025b.

Xu Wang, Yi Jin, Yigang Cen, Yidong Li, and Hui Yu. V2pnet: a voxel-to-point network framework for task-oriented point cloud sampling. *IEEE Transactions on Circuits and Systems for Video Technology*, 35(7):6286–6299, 2025c.

WG 07 MPEG 3D Graphics Coding and Haptics Coding. Enhanced g-pcc test model tm13v34. Technical Report MDS26337 WG07 N01456, ISO/IEC JTC 1/SC 29/WG 7, June 2026. URL `https://git.mpeg.expert/MPEG/3dgh/g-pcc/software/tm/mpeg-pcc-tmc13/-/tree/release-v34.0-rc2?ref_type=tags`. Version 1.

Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

Kang You, Tong Chen, Dandan Ding, M. Salman Asif, and Zhan Ma. Reno: Real-time neural compression for 3d lidar point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22172–22181, June 2025.

Haibao Yu, Wenxian Yang, Hongzhi Ruan, Zhenwei Yang, Yingjuan Tang, Xu Gao, Xin Hao, Yifeng Shi, Yifeng Pan, Ning Sun, Juan Song, Jirui Yuan, Ping Luo, and Zaiqing Nie. V2x-seq: A large-scale sequential dataset for vehicle-infrastructure cooperative perception and forecasting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Xumiao Zhang, Anlan Zhang, Jiachen Sun, Xiao Zhu, Y Ethan Guo, Feng Qian, and Z Morley Mao. Emp: Edge-assisted multi-vehicle perception. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pp. 545–558, 2021.

Lili Zhao, Kai-Kuang Ma, Xuhu Lin, Wenyi Wang, and Jianwen Chen. Real-time lidar point cloud compression using bi-directional prediction and range-adaptive floating-point coding. *IEEE transactions on broadcasting*, 68(3):620–635, 2022.

Xuanyu Zhou, Charles R Qi, Yin Zhou, and Dragomir Anguelov. Riddle: Lidar data compression with range image deep delta encoding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17212–17221, 2022.

A Octant-Partition Ablation

Each 8-bit occupancy symbol is coded as two sequential 4-bit partial symbols. This operation partitions the eight bits of the occupancy symbol rather than the voxel set itself. Each bit corresponds to one octant of the $2 \times 2 \times 2$ child grid, so the partition determines which decoded octant bits are available when predicting the second partial symbol. We compare the octant-parity partition used by ELiCv2 with a planar partition corresponding to the two nibbles of the occupancy byte. The planar partition codes $\{4, 5, 6, 7\}$ before $\{0, 1, 2, 3\}$, which groups the bits associated with the two planes along the z axis. Our partition codes $\{1, 2, 4, 7\}$ before $\{0, 3, 5, 6\}$. Every octant represented in the second partial symbol is therefore face-adjacent to three octants represented in the decoded first partial symbol instead of one.

Figure 3 visualizes the two partitions and their face-adjacency structure.

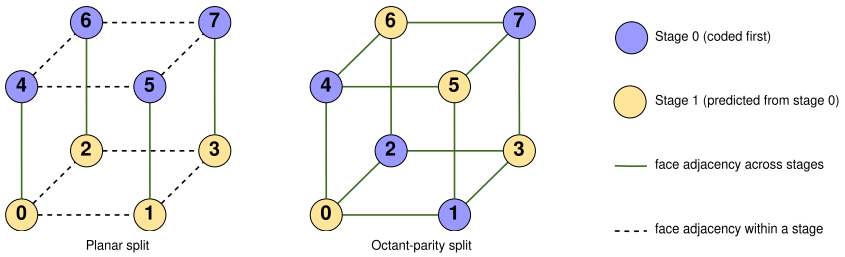

Figure 3. Comparison of planar and octant-parity partitions of the eight bits in an occupancy symbol. The diagram visualizes the octant associated with each bit and does not indicate a partition of the voxel set. Stage 0 is coded first, and Stage 1 is predicted from Stage 0. Solid edges connect face-adjacent octants across stages, while dashed edges connect octants within the same stage. Under octant parity, every Stage 1 octant has three face-adjacent Stage 0 neighbors.

Table 6 compares the two partitions with all other settings fixed. Both variants use the same channel width, kernel size, training schedule, and full-test evaluation procedure. Rate is computed with the deployed top-1 table selection by summing the predicted code lengths and normalizing by the number of occupied voxels at $\mathrm{posQ} = 1$. This common evaluation path isolates the occupancy partition from differences in the bitstream implementations of the two code trees.

Octant parity achieves a lower rate at all 20 dataset and bit-depth combinations. The reduction ranges from 0.062 to 0.246 bpp and averages 0.125 bpp. The mean reduction is largest on Ford at 0.189 bpp, followed by SemanticKITTI at 0.112 bpp, V2X-Seq at 0.101 bpp, and nuScenes at 0.098 bpp. These results support exposing spatially adjacent occupancy to the second stage. The comparison covers two natural partitions at one channel width and kernel size and does not claim optimality over all possible partitions.

Table 6. Rate comparison of octant-parity and planar occupancy splits for the $C{=}32$ intra model with a $3\times3\times3$ kernel after 300k training steps. Values are bpp on the complete test split. Positive Δ indicates that the planar split requires more bits.

Dataset	Split	16-bit	15-bit	14-bit	13-bit	12-bit
SemanticKITTI	Octant parity	**11.048**	**8.659**	**6.445**	**4.302**	**2.493**
	Planar	11.144	8.783	6.584	4.424	2.571
	Δ	$+0.096$	$+0.124$	$+0.139$	$+0.123$	$+0.078$
Ford	Octant parity	**12.912**	**10.377**	**8.075**	**6.021**	**4.095**
	Planar	13.025	10.523	8.276	6.267	4.332
	Δ	$+0.114$	$+0.146$	$+0.201$	$+0.246$	$+0.238$
nuScenes	Octant parity	**10.878**	**8.528**	**6.561**	**4.883**	**3.481**
	Planar	10.940	8.616	6.675	5.001	3.592
	Δ	$+0.062$	$+0.088$	$+0.113$	$+0.118$	$+0.111$
V2X-Seq	Octant parity	**15.950**	**13.171**	**10.570**	**8.244**	**6.246**
	Planar	16.052	13.262	10.648	8.325	6.398
	Δ	$+0.103$	$+0.092$	$+0.077$	$+0.081$	$+0.153$

B Per-Dataset Entropy-Interface Ablation

We report the per-dataset measurements underlying the averaged entropy-coding ablation in Table 4. All methods share the same network prediction. The continuous baseline forms a probability-weighted mixture of the codebook distributions, whereas indexed rANS uses the distribution selected by hard top-1 row and column indices. The comparison therefore isolates the interface between the shared network prediction and the entropy coder.

A conventional entropy model produces a different continuous categorical distribution for every voxel, whereas an integer entropy coder consumes discrete cumulative frequencies. The interface must therefore convert each voxel distribution into a complete monotonic integer CDF before coding. This work scales with both the number of voxels and the number of symbols, and the resulting CDF values must all cross the neural–coder boundary. ELiCv2 instead predicts the index of a coder-ready distribution from a shared codebook. Each voxel exposes only one compact index, and the corresponding integer CDF is retrieved from a table constructed once rather than materialized again for every voxel.

Table 7 shows the same pattern on every dataset. Replacing arithmetic coding with rANS reduces coding time but leaves per-voxel CDF construction and transfer unchanged. The discrete entropy interface instead predicts compact table indices directly, reducing both preparation and device-to-host transfer before indexed rANS coding. This interface change provides a $3.8\times$ to $5.4\times$ total speed-up for a rate difference of 0.91% to 1.43%.

Table 7. Per-dataset entropy-coding ablation on Jetson Thor using ten random frames per dataset at bit-depth 16 with the $C{=}32$ intra model. *Prep.* denotes per-voxel CDF construction for continuous coding and top-1 table selection for indexed rANS. Negative rate differences indicate lower bpp than indexed rANS.

		Encoding time (ms/frame)					Rate	
Dataset	Entropy coder	Prep.	D2H	Code	Total	× ours	bpp	Δ rate
nuScenes	Arithmetic coding	15.22	1.85	5.37	22.44	4.3×	11.0319	−1.43%
	rANS	15.22	1.85	2.52	19.59	3.8×	11.0338	−1.42%
	Indexed rANS	2.75	0.74	1.69	5.18	–	11.1924	–
SemanticKITTI	Arithmetic coding	53.96	2.62	19.89	76.46	5.4×	10.1504	−1.11%
	rANS	53.96	2.62	8.98	65.55	4.6×	10.1508	−1.11%
	Indexed rANS	7.80	0.81	5.54	14.15	–	10.2644	–
Ford	Arithmetic coding	45.62	2.48	16.76	64.85	5.3×	12.9944	−0.98%
	rANS	45.62	2.48	7.62	55.71	4.5×	12.9950	−0.98%
	Indexed rANS	6.67	0.81	4.83	12.32	–	13.1233	–
V2X-Seq	Arithmetic coding	38.17	2.37	15.06	55.60	5.2×	16.9722	−0.91%
	rANS	38.17	2.37	6.54	47.08	4.4×	16.9732	−0.91%
	Indexed rANS	5.79	0.83	4.15	10.77	–	17.1282	–

C Effect of the Table Codebook Size

The codebook size T controls the rate–storage trade-off of the discrete entropy interface in Section 3.2. We examine this trade-off to motivate the default choice of $T{=}32$.

Setup. We evaluate $T \in \{16, 32, 64\}$ for geometry-only intra-frame coding on SemanticKITTI with $C{=}32$. All models use the same training recipe and four BoE coding networks, with two coding stages per network. Only the codebook size and selection-head width vary. We report model rates on the complete test split at five bit depths, using deterministic $\arg\max$ selection and the same normalisation as in the main experiments.

Table 8. Effect of the table codebook size on SemanticKITTI for the geometry-only intra model ($C{=}32$, $K{=}3$). Values are model bpp on the complete test split. Best rates are in **bold**. Negative Δ indicates rate savings from doubling T; differences are computed before rounding. CDF storage accounts for four BoE coding networks and two stages per network.

	Geometry rate (bpp)					CDF storage		
T	16-bit	15-bit	14-bit	13-bit	12-bit	T^2	Rows	Size
16	11.258	8.845	6.599	4.416	2.560	256	2,048	68 KiB
32 (default)	11.048	8.659	6.445	4.302	2.493	1,024	8,192	272 KiB
64	**10.950**	**8.574**	**6.373**	**4.252**	**2.463**	4,096	32,768	1.0625 MiB
Δ 16→32	−0.210	−0.186	−0.154	−0.114	−0.067			
Δ 32→64	−0.098	−0.086	−0.072	−0.050	−0.030			

Results. Table 8 shows diminishing rate gains as T increases. Doubling T from 16 to 32 saves 0.067–0.210 bpp across the five bit depths, whereas doubling it again saves only 0.030–0.098 bpp. Each doubling quadruples CDF storage, since the coder stores $4 \times 2 \times T^2$ distributions: 68 KiB, 272 KiB, and 1.0625 MiB for the three configurations. At $b{=}16$, reducing T from 32 to 16 costs 0.210 bpp, while increasing it to 64 saves only 0.098 bpp. We therefore use $T{=}32$ as a practical rate–storage compromise. This ablation uses one training seed per configuration on a single dataset; decoding latency at $T \neq 32$ was not measured.

D KERNEL-MAP REUSE DERIVATION AND EXAMPLE

This appendix expands the kernel-map reuse of Section 3.3. We derive how child-level connectivity can be obtained directly from the parent kernel-map and decoded occupancy, give the two lookup tables used by our implementation, and provide a worked example. We denote the active parent voxels by $\mathcal{P} = \{\mathbf{p}_i\}$, their kernel-map by $\mathbf{K}^p \in \mathbb{Z}^{N_p \times 27}$, the child occupancy mask of parent i by the 8-bit integer $m_i \in \{0, \dots, 255\}$, and the resulting child kernel-map by $\mathbf{K}^c \in \mathbb{Z}^{N_c \times 27}$. We use $m_i[q] \in \{0, 1\}$ to denote the bit associated with octant q.

Deterministic child indexing. Each parent voxel generates up to eight children. We denote an octant by the zero-based scalar index $q = x + 2y + 4z$ for $(x, y, z) \in \{0, 1\}^3$ and denote its binary offset by $\boldsymbol{\beta}(q) = (x, y, z)$. Only children satisfying $m_i[q] = 1$ are instantiated. We assign active children contiguous indices using

$$\mathrm{idx}(i, q) = \mathrm{base}(i) + \mathrm{popcount}(m_i \wedge (2^q - 1)), \qquad \mathrm{base}(i) = \sum_{j<i} \mathrm{popcount}(m_j), \tag{11}$$

where $\mathrm{base}(i)$ is an exclusive prefix sum over the numbers of active children. Thus, the global index of every child is determined directly from its parent index and occupancy mask, without constructing child coordinates.

Hierarchical connectivity. Consider a child of parent $\mathbf{p}$ at octant q with coordinate $\mathbf{x} = 2\mathbf{p} + \boldsymbol{\beta}(q)$. For a submanifold-convolution offset $\mathbf{d} \in \{-1, 0, 1\}^3$, its target neighbor is $\mathbf{x}' = 2\mathbf{p} + \boldsymbol{\beta}(q) + \mathbf{d}$. We decompose this target as $\mathbf{x}' = 2\mathbf{p}' + \boldsymbol{\gamma}$, where

$$\mathbf{t} = \left\lfloor \frac{\boldsymbol{\beta}(q) + \mathbf{d}}{2} \right\rfloor, \qquad \mathbf{p}' = \mathbf{p} + \mathbf{t}, \qquad \boldsymbol{\gamma} = \boldsymbol{\beta}(q) + \mathbf{d} - 2\mathbf{t}. \tag{12}$$

Here, $\mathbf{t} \in \{-1, 0, 1\}^3$ identifies the neighboring parent containing the target, while $\boldsymbol{\gamma} \in \{0, 1\}^3$ identifies the target child offset within that parent. Therefore, a child-level neighbor can always be resolved by one parent-level lookup followed by one occupancy test.

For the $3 \times 3 \times 3$ kernel, we index an offset $\mathbf{d}_k = (d_x, d_y, d_z)$ by

$$k = (d_x + 1) + 3(d_y + 1) + 9(d_z + 1), \qquad k \in \{0, \dots, 26\}. \tag{13}$$

Since Equation equation 12 depends only on the source child octant q and kernel index k, its result can be precomputed into two small lookup tables,

$$T_{\mathrm{col}}(q, k) = k', \qquad T_{\mathrm{bit}}(q, k) = \mathrm{ind}(\boldsymbol{\gamma}), \tag{14}$$

where T_{col} specifies which parent-level kernel-map column to consult and T_{bit} specifies which child octant to test within that neighboring parent. Here, $\mathrm{ind}(x, y, z) = x + 2y + 4z$ converts a binary offset into its scalar octant index. Tables 9 and 10 provide the complete lookup tables.

Table 9. Lookup table $T_{\mathrm{col}}(q,k)$ for the 3×3×3 submanifold kernel. Each entry specifies the parent-level kernel-map column containing the target child. Together with T_{bit} in Table 10, it completely determines child-level connectivity.

$q\backslash k$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
0	0	1	1	3	4	4	3	4	4	9	10	10	12	13	13	12	13	13	9	10	10	12	13	13	12	13	13
1	1	1	2	4	4	5	4	4	5	10	10	11	13	13	14	13	13	14	10	10	11	13	13	14	13	13	14
2	3	4	4	3	4	4	6	7	7	12	13	13	12	13	13	15	16	16	12	13	13	12	13	13	15	16	16
3	4	4	5	4	4	5	7	7	8	13	13	14	13	13	14	16	16	17	13	13	14	13	13	14	16	16	17
4	9	10	10	12	13	13	12	13	13	9	10	10	12	13	13	12	13	13	18	19	19	21	22	22	21	22	22
5	10	10	11	13	13	14	13	13	14	10	10	11	13	13	14	13	13	14	19	19	20	22	22	23	22	22	23
6	12	13	13	12	13	13	15	16	16	12	13	13	12	13	13	15	16	16	21	22	22	21	22	22	24	25	25
7	13	13	14	13	13	14	16	16	17	13	13	14	13	13	14	16	16	17	22	22	23	22	22	23	25	25	26

Table 10. Lookup table $T_{\mathrm{bit}}(q,k)$ for the 3×3×3 submanifold kernel. Each entry specifies the target child octant within the neighboring parent selected by T_{col}.

$q\backslash k$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
0	7	6	7	5	4	5	7	6	7	3	2	3	1	0	1	3	2	3	7	6	7	5	4	5	7	6	7
1	6	7	6	4	5	4	6	7	6	2	3	2	0	1	0	2	3	2	6	7	6	4	5	4	6	7	6
2	5	4	5	7	6	7	5	4	5	1	0	1	3	2	3	1	0	1	5	4	5	7	6	7	5	4	5
3	4	5	4	6	7	6	4	5	4	0	1	0	2	3	2	0	1	0	4	5	4	6	7	6	4	5	4
4	3	2	3	1	0	1	3	2	3	7	6	7	5	4	5	7	6	7	3	2	3	1	0	1	3	2	3
5	2	3	2	0	1	0	2	3	2	6	7	6	4	5	4	6	7	6	2	3	2	0	1	0	2	3	2
6	1	0	1	3	2	3	1	0	1	5	4	5	7	6	7	5	4	5	1	0	1	3	2	3	1	0	1
7	0	1	0	2	3	2	0	1	0	4	5	4	6	7	6	4	5	4	0	1	0	2	3	2	0	1	0

Kernel-map propagation. Given the two lookup tables, the child kernel-map is generated directly from the parent kernel-map and occupancy masks. For an active child $u = \mathrm{idx}(i,q)$ and kernel index k, we first obtain

$$k' = T_{\mathrm{col}}(q,k), \qquad q' = T_{\mathrm{bit}}(q,k), \qquad j = \mathbf{K}^p[i,k']. \tag{15}$$

The corresponding child-level entry is then

$$\mathbf{K}^c[u,k] = \begin{cases} \mathrm{idx}(j,q'), & \text{if } j \geq 0 \text{ and } m_j[q'] = 1, \\ -1, & \text{otherwise.} \end{cases} \tag{16}$$

Thus, each child-level connection requires only fixed table lookups, a parent-kernel-map access, an occupancy test, and child indexing. No child coordinate hashing or dynamic neighbor search is required.

Worked example. Consider the three parent voxels

$$\mathbf{p}_0 = (0,0,0), \qquad \mathbf{p}_1 = (1,0,0), \qquad \mathbf{p}_2 = (1,0,1), \tag{17}$$

with occupancy masks

$$m_0 = 0b00100010, \qquad m_1 = 0b00000011, \qquad m_2 = 0b00010001. \tag{18}$$

The active octants are therefore $\{1,5\}$, $\{0,1\}$, and $\{0,4\}$, respectively. Equation equation 11 assigns the resulting children the contiguous indices shown in Table 11.

Table 11. Child indices for the worked example. The global child ordering is obtained directly from the parent occupancy masks using Equation equation 11.

Parent i	Active q	$\mathrm{base}(i)$	$\mathrm{idx}(i,q)$	Child coordinate
0	1	0	0	$(1,0,0)$
0	5	0	1	$(1,0,1)$
1	0	2	2	$(2,0,0)$
1	1	2	3	$(3,0,0)$
2	0	4	4	$(2,0,2)$
2	4	4	5	$(2,0,3)$

As one concrete propagation, consider $\mathbf{K}^c[2,21]$. From Table 11, child $u = 2$ corresponds to $(i,q) = (1,0)$. Reading row $q = 0$ and column $k = 21$ of Tables 9 and 10 gives

$$T_{\mathrm{col}}(0,21) = 12, \qquad T_{\mathrm{bit}}(0,21) = 5. \tag{19}$$

The parent kernel-map gives

$$j = \mathbf{K}^p[1,12] = 0, \tag{20}$$

so the target lies in parent $\mathbf{p}_0$. Since $m_0[5] = 1$, child octant 5 exists, and Equation equation 11 gives

$$\mathrm{idx}(0,5) = 1. \tag{21}$$

Therefore,

$$\mathbf{K}^c[2,21] = 1. \tag{22}$$

The same procedure is applied independently to every active child and kernel offset to construct the complete child kernel-map.

Equivalence to coordinate-based construction. Kernel-map reuse changes only how connectivity is obtained, not the connectivity itself. For any child coordinate $\mathbf{x} = 2\mathbf{p} + \boldsymbol{\beta}(q)$ and kernel offset $\mathbf{d}$, Equation equation 12 uniquely identifies the parent containing $\mathbf{x} + \mathbf{d}$ and the corresponding child octant. Equation equation 16 therefore produces the same child-level neighbor relation as explicit coordinate-based search, provided that the parent kernel-map is identical. The proposed method replaces repeated coordinate discovery with deterministic connectivity propagation while leaving the sparse convolution unchanged.

E Kernel-Map Construction with Larger Kernels

Our kernel-map propagation in Section 3.3 exploits a property specific to progressive coarse-to-fine occupancy coding. At each level, the active voxels are not an arbitrary coordinate set. They are deterministically generated by expanding the previous level according to the decoded child occupancy. Consequently, once the parent kernel-map and child occupancy are known, the connectivity of the finer level can be propagated without reconstructing it through coordinate-based neighbor search.

While this advantage arises from the progressive coding structure, it is not restricted to the $3\times3\times3$ kernel used in ELiCv2. The same parent-to-child connectivity propagation can be constructed for larger neighborhoods by extending the corresponding offset lookup tables. We verify this by evaluating $3\times3\times3$, $5\times5\times5$, $7\times7\times7$, and $9\times9\times9$ kernels.

Measurement. We isolate kernel-map construction from sparse convolution and compare our progressive builder with the hash-based builder of TorchSparse++. For each frame, runtime is accumulated over all progressively constructed octree levels from bit-depth 3 to the level preceding the leaf. Bit-depth 2 is excluded because it has no parent kernel-map and is constructed identically by both methods. All experiments are conducted on Jetson Thor at bit-depth 16 ($\mathrm{posQ} = 4$). Both implementations were verified to produce identical kernel-maps at every evaluated level and kernel size.

Table 12 reports construction latency for all evaluated kernel sizes and datasets.

Table 12. Kernel-map construction latency on Jetson Thor for different kernel sizes. Our method propagates connectivity across progressively decoded octree levels, whereas the baseline reconstructs each kernel-map using TorchSparse++'s hash-based builder. Both methods produce identical kernel-maps.

Kernel	Dataset	Voxels	Ours (ms)	Hashing (ms)	Speed-up
$3\times3\times3$	SemanticKITTI	120,826	1.69	5.78	$3.42\times$
	Ford	80,002	1.46	5.16	$3.54\times$
	nuScenes	29,824	0.70	2.80	$4.00\times$
	V2X-Seq	63,416	1.27	4.51	$3.55\times$
	Mean		1.28	4.56	$\mathbf{3.63}\times$
$5\times5\times5$	SemanticKITTI	120,826	7.58	21.17	$2.79\times$
	Ford	80,002	6.15	17.62	$2.86\times$
	nuScenes	29,824	2.24	6.59	$2.94\times$
	V2X-Seq	63,416	4.97	14.16	$2.85\times$
	Mean		5.24	14.88	$\mathbf{2.86}\times$
$7\times7\times7$	SemanticKITTI	120,826	25.84	54.02	$2.09\times$
	Ford	80,002	20.59	45.06	$2.19\times$
	nuScenes	29,824	6.88	15.55	$2.26\times$
	V2X-Seq	63,416	16.82	36.60	$2.18\times$
	Mean		17.53	37.81	$\mathbf{2.18}\times$
$9\times9\times9$	SemanticKITTI	120,826	58.16	111.30	$1.91\times$
	Ford	80,002	46.28	92.67	$2.00\times$
	nuScenes	29,824	15.71	31.79	$2.02\times$
	V2X-Seq	63,416	37.61	75.10	$2.00\times$
	Mean		39.44	77.71	$\mathbf{1.98}\times$

Generalization across kernel sizes. Table 12 shows that the computational advantage enabled by progressive occupancy coding is not specific to the $3\times3\times3$ stencil. For the $3\times3\times3$ kernel used in ELiCv2, our builder reduces mean construction latency from 4.56 to 1.28 ms. The arithmetic mean of the per-dataset speedups is $3.63\times$. The same construction remains effective for larger neighborhoods, with mean per-dataset speedups of $2.86\times$, $2.18\times$, and $1.98\times$ for $5\times5\times5$, $7\times7\times7$, and $9\times9\times9$ kernels, respectively.

The relative gain decreases with kernel size because our builder explicitly propagates the connectivity for all K^3 kernel offsets, whereas the hash-based implementation amortizes part of its coordinate-map construction across offsets. Nevertheless, the proposed method remains consistently faster for every tested kernel size and dataset while producing exactly the same kernel-maps.

These results clarify both the scope and the source of the proposed optimization. Kernel-map propagation is enabled by the deterministic parent–child structure inherent in progressive coarse-to-fine occupancy coding, rather than by a particular convolution stencil. Therefore, within such progressive codecs, the same principle can be applied beyond the $3\times3\times3$ kernel used in ELiCv2 without reverting to coordinate-based connectivity search.

F Ego-Motion Alignment and Reference Resolution

Figure 4 visualizes the ego-motion alignment used by the inter model and examines how its effectiveness changes with spatial resolution. The example is taken from consecutive frames of SemanticKITTI sequence 18, where the sensor translates by 0.826 m with a yaw change of $0.82°$.

Why alignment is necessary. Even moderate vehicle motion is large relative to the voxel resolution used for inter prediction. In Figure 4(a), the 0.826 m translation corresponds to approximately 26 voxels at bit-depth 13 (32 mm cells), substantially displacing static structures between consecutive frames. After applying the relative ego pose, these structures become locally aligned, as shown in Figure 4(b). Ego-motion compensation therefore provides a useful temporal reference without learned motion estimation or correspondence search.

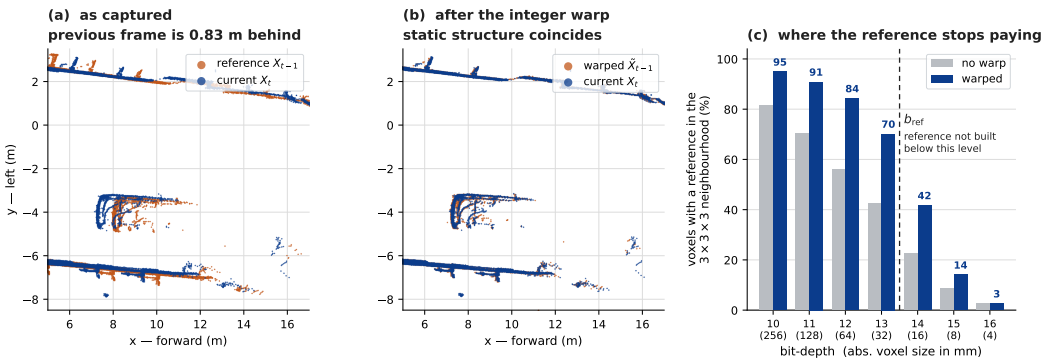

Figure 4. Effect of ego-motion alignment on the inter reference. (a) Previous and current frames before alignment. (b) The previous frame after the deterministic ego-motion warp. (c) Fraction of current voxels having an occupied reference within a $3\times3\times3$ neighborhood at each bit-depth. The dashed line denotes the reference cutoff $\ell_{\mathrm{ref}} = 13$.

Deterministic encoder–decoder alignment. To ensure identical reference construction at the encoder and decoder, the relative rigid transform is quantized once into fixed-point parameters $(\mathbf{R}, \mathbf{t})$ and transmitted with each P-frame. Reference coordinates are then generated entirely with integer arithmetic. Both sides therefore construct bit-identical warped coordinates without independently evaluating floating-point pose transformations.

Why the reference stops at $\ell_{\mathrm{ref}} = 13$. The reference pyramid is used only for $\ell \le \ell_{\mathrm{ref}} = 13$, corresponding to a 32 mm cell. This cutoff is defined by absolute bit-depth, so its physical resolution is unchanged across operating points. Two effects make finer references progressively less useful. First, the $3\times3\times3$ neighborhood has a radius of one voxel, so its physical coverage halves at each finer level. Second, consecutive LiDAR sweeps sample different surface points, causing voxel-level correspondence itself to vanish at sufficiently fine resolutions even after correct ego-motion alignment.

This behavior is visible in Figure 4(c). For the shown frame pair, alignment increases the local reference hit rate from 81.6% to 94.9% at a 256 mm cell and from 42.4% to 70.1% at the 32 mm cells of $\ell = 13$. The aligned hit rate then drops to 41.7% at 16 mm and 14.0% at 8 mm. More generally, on SemanticKITTI the reference hit rate decreases from 63% at $\ell = 13$ to 32% at $\ell = 14$ and 8.5% at $\ell = 15$. Increasing the neighborhood radius to two voxels recovers much of the $\ell = 14$ correspondence but provides little recovery at $\ell = 15$, indicating that the fine-scale degradation is not merely caused by a limited search radius. We therefore restrict temporal conditioning to $\ell \le 13$, where the warped reference retains reliable local correspondence.

Sensitivity to ego-motion. Figure 5 compares per-voxel coding behavior under four representative ego-motion regimes.

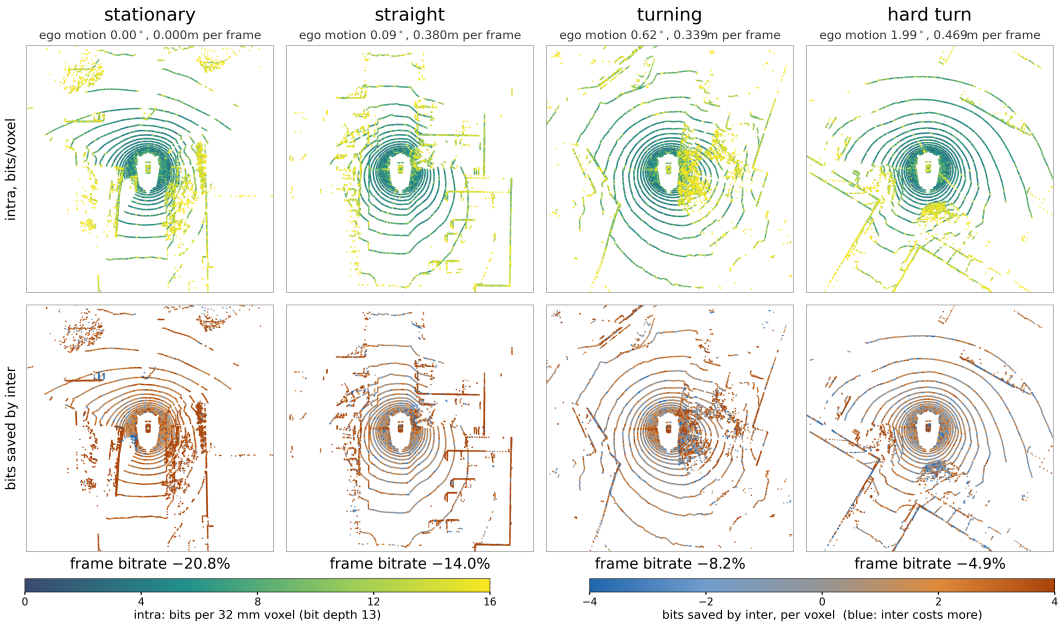

Figure 5. Per-voxel intra coding cost (top) and inter-frame bit saving (bottom) under four ego-motion regimes in nuScenes. Positive values in the bottom row indicate savings from the aligned reference. Examples are from the same drive.

Figure 5 illustrates variation in coding gains across ego-motion regimes within the same drive, using the same sensor and model. A nearly stationary frame obtains a 20.8% rate reduction, and straight driving with $0.380\,\mathrm{m}$ translation and $0.09°$ rotation obtains 14.0%. Turning with similar translation ($0.339\,\mathrm{m}$) but $0.62°$ rotation yields 8.2%, while a $1.99°$ hard turn yields 4.9%. Although ego-motion alignment compensates for rigid displacement, changes in visibility and surface sampling can still reduce correspondence between consecutive sweeps. These examples illustrate motion-associated variation rather than isolate the effect of rotation alone.

Despite this variation, 99.9% of the $47{,}734$ nuScenes P-frames improve over intra coding, with a mean saving of 13.1%. The reference gate can suppress uninformative temporal context, avoiding a motion-dependent fallback. These within-drive results characterize motion sensitivity. Cross-dataset gains also depend on viewpoint consistency, scene dynamics, and intra-model strength.

G Rationale for Ego-Motion Alignment

Our inter model first aligns the previous LiDAR sweep using the sensor ego-motion and then reads a local 3×3×3 neighborhood. An alternative, adopted by existing learned inter point cloud codecs, is to leave the two frames unaligned and let a learned operator search for cross-frame correspondence. We explain why ELiCv2 instead uses explicit ego-motion alignment for embedded LiDAR compression.

Cost of cross-frame correspondence. Without alignment, the displacement between consecutive vehicle-mounted LiDAR frames can be much larger than a local sparse-convolution stencil. As analyzed in Appendix F, a $3 \times 3 \times 3$ window would need to grow substantially to cover the displacement caused by typical ego-motion. Existing inter-frame methods therefore employ wider or data-dependent cross-frame operators.

Table 13 summarizes representative designs. D-DPCC (Fan et al., 2022) combines cross-frame features and performs nearest-neighbor interpolation. Akhtar et al. (2024) use a $3\times3\times3$ geometry-aware sparse convolution, while Unicorn (Wang et al., 2025a) uses a $9\times9\times9$ target convolution for its LiDAR warping model. In contrast, ELiCv2 removes the dominant rigid displacement before temporal conditioning, allowing the reference operator to remain local.

Table 13. Cross-frame operators in learned inter point cloud codecs. *Taps* denotes the number of reference positions accessed per output position for fixed-stencil operators.

Method	Cross-frame operator	Taps	Application
D-DPCC (Fan et al., 2022)	Conv. + k-NN interpolation	$27/3^{\dagger}$	Single scale
Akhtar et al. (Akhtar et al., 2024)	GSConv $3\times3\times3$	27	Once
Unicorn (Wang et al., 2025a)	Target conv. $9\times9\times9$	729	Multi-scale
Ours	Ego warp + local gather	27	$\ell \leq 13$

† The interpolation uses a k=3 nearest-neighbor query rather than three fixed stencil accesses.

Sparse convolution requires a kernel-map that specifies which voxels are neighbors. Within one frame, ELiCv2 can propagate this map through the known octree hierarchy. A cross-frame operator instead connects voxels from two different frames, whose coordinates change for every frame pair. It must therefore construct a new cross-frame kernel-map each time. A wide cross-frame operator is costly for two reasons. It accesses more neighbors and must also build the connectivity between the two frames.

Illustrative cost of wide kernel-map construction. To illustrate the cost of wide sparse neighborhoods, we refer to the within-frame kernel-map construction benchmark in Appendix E. For a 9×9×9 kernel, construction takes 39.44 ms with our progressive builder and 77.71 ms with coordinate hashing, averaged over the four datasets and accumulated across the evaluated octree levels.

These measurements demonstrate that wide kernel-map construction can incur substantial overhead. They do not estimate the cost of the cross-frame operators in Table 13, whose active supports, scales, and map-construction procedures differ from this benchmark.

ELiCv2 instead uses ego-motion alignment followed by a local $3\times3\times3$ occupancy gather. This temporal path requires no cross-frame sparse-convolution kernel-map, although reference warping and local lookup still incur computation.

Benefits of ego-motion alignment. For vehicle-mounted LiDAR, much of the displacement between consecutive sweeps is caused by the rigid motion of the sensor itself. Instead of asking the network to rediscover this displacement, we transmit the quantized relative ego transform and warp the reference onto the current coordinate system. The transform requires only 96 bytes per P-frame and is applied using the integer warp in Equation equation 6.

After alignment, temporal conditioning requires only a local $3\times3\times3$ occupancy lookup. This gives three practical advantages. First, the dominant rigid displacement is removed before inference, avoiding the need for a wide correspondence search. Second, the temporal path does not require an additional cross-frame sparse-convolution backbone or its associated kernel-map construction. Third, the fixed-point transform guarantees that the encoder and decoder construct the same aligned reference exactly.

This choice is specific to driving LiDAR. Ego-motion cannot compensate for independently moving objects, whereas learned flow or correspondence models can represent such motion. For deformable point cloud sequences, this flexibility can be essential. In vehicle-mounted LiDAR, however, ego-motion is readily available and removes a large deterministic component of inter-frame displacement at negligible side-information cost. Appendix F visualizes the resulting alignment and correspondence rates and evaluates the inter gain across different ego-motion regimes.

Our design therefore uses the network primarily to model residual temporal dependency rather than recover a rigid transform already known to the platform. This design allows ELiCv2 to obtain inter-frame gains using a local reference gate without motion search or complex temporal processing.

H Energy per Frame

Measurement protocol. We measure the marginal energy per frame of learned codecs on Jetson Thor using the module-level INA238 sensor sampled at 200 Hz. Encoding and decoding are mea-

sured separately over 30-frame runs. For each run, we subtract the idle power measured immediately beforehand and integrate the remaining power over the I/O-excluded execution time, then divide by the number of processed frames. This metric measures the additional energy consumed by coding on an already-powered platform; it excludes the platform's idle energy. Table 14 reports the arithmetic mean across the four datasets at bit-depth 16.

Table 14. Marginal energy per frame at bit-depth 16 ($\mathrm{posQ} = 4$) on Jetson Thor, averaged equally over four datasets. Idle energy is excluded. Lower values are better; the lowest value in each column is in **bold**.

Codec	Encode (mJ/frame)	Decode (mJ/frame)
RENO	3,496	3,607
ELiCv1	3,423	3,631
AnyPcc	21,345	21,246
ELiCv2 C=32 intra	**1,695**	**2,118**
ELiCv2 C=32 inter	1,970	2,375
ELiCv2 C=64 intra	4,626	5,782
ELiCv2 C=64 inter	4,997	6,110

Energy efficiency. The C=32 intra model has the lowest encoding and decoding energy among the codecs in Table 14, requiring $1{,}695$ and $2{,}118$ mJ/frame, respectively. With temporal conditioning, the C=32 inter model requires $1{,}970$ mJ/frame for encoding and $2{,}375$ mJ/frame for decoding. Its encoding energy is approximately 44% lower than RENO and 42% lower than ELiCv1, while its decoding energy is approximately 34% and 35% lower, respectively. Both C=32 variants also require substantially less energy than AnyPcc, which consumes more than $21{,}000$ mJ/frame for either operation.

Cost of temporal conditioning and model capacity. For C=32, switching from intra to inter coding increases encoding energy by 275 mJ/frame (16.2%) and decoding energy by 257 mJ/frame (12.1%). This additional energy accompanies the improvement in mean BD-rate reduction from 26.66% to 30.97% reported in Table 2. Thus, temporal conditioning improves compression while retaining lower energy per frame than RENO and ELiCv1.

Increasing the width to C=64 further improves compression but raises energy consumption. The C=64 inter model requires $4{,}997$ mJ/frame for encoding and $6{,}110$ mJ/frame for decoding, approximately $2.5\times$ and $2.6\times$ the corresponding C=32 inter values. These measurements support using C=32 when energy efficiency is a priority and C=64 when the additional compression gain justifies the higher energy cost.

I Memory Footprint

We measure GPU memory on Jetson Thor after two warm-up passes. Activation memory is the increase from the resident baseline to the maximum allocated memory during coding. Table 15 reports encoding and decoding on SemanticKITTI at bit-depth 16, the most memory-intensive operating point in our evaluation.

Table 15. Activation memory for SemanticKITTI encoding and decoding at bit-depth 16 on Jetson Thor. Memory is measured after warm-up. Lower values are better.

Method	Encode (MB)	Decode (MB)
RENO	181.0	209.5
ELiCv1	**138.0**	165.1
AnyPcc	912.0	863.2
ELiCv2 (C=32, intra)	174.5	**118.8**
ELiCv2 (C=32, inter)	229.0	209.0
ELiCv2 (C=64, intra)	239.0	181.7
ELiCv2 (C=64, inter)	325.0	301.0

The C=32 intra decoder uses the least memory among the evaluated learned codecs, reducing activation memory by 28% relative to ELiCv1, 43% relative to RENO, and 86% relative to AnyPcc. ELiCv2 also uses less memory for decoding than encoding across all four configurations. Inter coding retains the temporal reference and therefore increases memory, while the C=64 configurations trade additional memory for compression efficiency. All codecs listed in Table 15 fit on Jetson Thor, so this comparison reflects headroom for concurrent perception and planning workloads and portability to smaller devices rather than device compatibility.

Joint deployment trade-off. Memory should be considered together with compression efficiency and latency. At the same SemanticKITTI bit-depth 16 operating point, the C=32 inter model uses 229.0 MB for encoding and requires 101.1 ms, compared with 138.0 MB and 204.0 ms for ELiCv1 and 181.0 MB and 246.0 ms for RENO. The modest increase in encoding memory therefore approximately halves latency while improving the mean BD-rate reduction from 27.54% for ELiCv1 and 17.02% for RENO to 30.97%. Its decoding memory is also comparable to RENO at 209.0 versus 209.5 MB. The C=64 inter model offers a rate-oriented option with a stronger 35.22% mean BD-rate reduction at a higher memory cost. Together, these results show that ELiCv2 provides a favorable compression–latency–memory trade-off rather than optimizing one deployment metric in isolation.