HIGH-FIDELITY AND TOPOLOGY-LOSSLESS LATTICE COMPRESSION VIA LEARNED CONTEXT MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Lattice structures have been widely used in various applications of additive manufacturing due to its superior physical properties and material efficiency. However, a lattice structure with a huge number of struts would consume massive memory and storage space, which hinders its practical deployment in large-scale design and manufacturing scenarios. The storage challenge becomes even more pressing when moving from individual lattices to assembled lattice models. To solve this issue, we propose the Context-based Lattice Compression (CLC) framework, which leverages learned context models to provide a storage-friendly lattice representation. Specifically, we construct two sets of multi-resolution learnable hash embeddings that separately encode the lattice nodes and struts, and then excavate context dependencies in the embeddings to enable probability prediction for information entropy reduction. Additionally, we exploit a context model to accurately estimate the probability of geometric residuals, which are crucial for high-fidelity restoration, and we also use Delta Encoding to compress the correction edge set for lossless topology reconstruction. We also utilize feature pruning and hash fusion to further improve compactness and context quality across levels. This design allows the framework to jointly address storage reduction, geometry restoration, and topology preservation within a unified compression pipeline. Quantitative and qualitative evaluations demonstrate that CLC achieves up to 16.58$\times$ compression over vanilla lattice representation, achieving a 1.95$\times$ size reduction over the SOTA lattice compression approach while simultaneously improving fidelity and topological integrity.

1 INTRODUCTION

Lightweight lattice structures offer mechanical performance and material efficiency in aerospace, automotive, and biomedical applications du Plessis et al. (2022); Noronha et al. (2023); Nazir et al. (2019); Liu et al. (2021); Dong et al. (2017). Advances in additive manufacturing have made complex lattices practical to fabricate, but representing large numbers of struts demands substantial storage, particularly in designs with spatially graded density. Figure 1 illustrates the storage cost of a large-scale lattice and the fidelity of its compressed reconstruction.

Existing work either compresses instantiated geometry through lightweight encoding or compact boundary construction Zhao & Huang (2023); Zou et al. (2024), or stores generative descriptions to synthesize geometry on demand Zheng et al. (2023); Ha et al. (2023); Zhan et al. (2025). The former requires reprocessing after edits; the latter often depends on specific design spaces and offers limited local control.

We propose Context-based Lattice Compression (CLC), which learns compact node- and strut-level representations and their contextual dependencies. CLC combines multi-resolution hash embeddings and context models with geometric residuals and correction edges for high-fidelity geometry and lossless topology.

Our main contributions can be summarized as follows:

- To our knowledge, we are the first to model contexts for lattice compression, i.e., using multi-resolution hash embeddings to efficiently encode the lattice nodes and struts, while exploiting the inherent consistencies among the embeddings to effectively reduce storage

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.

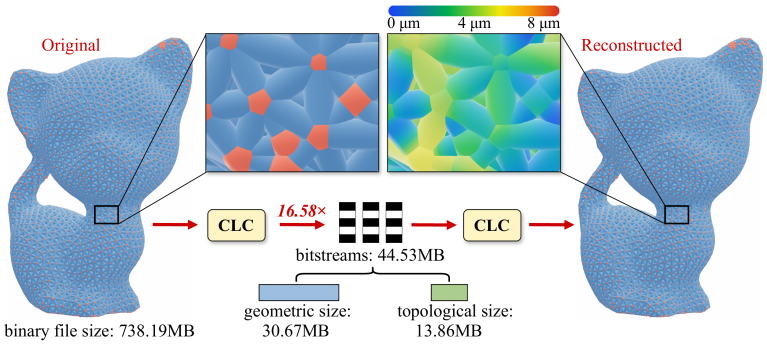


Figure 1: Compression and reconstruction of a large-scale Tetgen lattice with 10,995,066 nodes and 74,765,952 edges. The original binary representation occupies 738.19 MB. CLC encodes the geometry and topology into bitstreams of 30.67 MB and 13.86 MB, respectively. Their combined size of 44.53 MB corresponds to a 16.58× reduction in storage. The deviation heatmap of the magnified region indicates that the reconstructed geometry closely matches the original.

size. We also utilize feature pruning to further facilitate compression, and perform hash fusion to provide more accurate contexts.

- To achieve high-fidelity geometry reconstruction while preserving topology losslessly under compression, we develop a dedicated context model to estimate the probability distribution of geometric residuals for accurate geometry restoration, and further employ Delta Encoding to compress the correction edge set for topology preservation.
- Extensive experiments on three benchmark datasets demonstrate that, compared with the SOTA lattice compression method, our CLC framework achieves more efficient compression together with improved fidelity and topological integrity.

2 Related Work

2.1 Representation of the Lattice Structure

Lattice representations describe topology and geometry Liu et al. (2021); Zou & Luo (2025). Regular strut-based lattices use grids, unit-cell libraries, or transition rules; irregular ones use graphs with node and strut attributes Liu et al. (2020); Letov & Zhao (2023; 2024). Geometric models use primitives, CSG/B-rep operations, or large-scale meshing for fabrication and geometric queries Tang et al. (2019); Gupta et al. (2019a); Zou et al. (2024). Their memory and computation costs on large lattices Zou & Luo (2025) motivate a compact representation that preserves an explicit graph.

2.2 Lattice Structure Compression

Geometric Compression Methods These methods retain explicit lattices while simplifying or encoding their geometry through triangulation, approximate nodal evaluation, connection simplification, meta-meshing, or compact surfaces Chougrani et al. (2017); Zhao & Huang (2023); Wu et al. (2020); Verma et al. (2020); Zou et al. (2024); van Sosin et al. (2021); Savio et al. (2018). They remain compatible with CAD and fabrication pipelines, but editing requires reprocessing encoded geometry, and irregular lattices offer less repetition to exploit.

Generative Methods These methods encode construction rules and synthesize geometry on demand. Early approaches use editable programs, tran-similar rules, or conformal synthesis within prescribed domains Gupta et al. (2019b); Kurzeja & Rossignac (2020); Hong & Elber (2021). Recent graph-based methods optimize or generate lattices under structural or response constraints Wang et al. (2020a); Maurizi et al. (2025). Latent methods encode topology, geometry, and properties for interpolation, inverse design, arithmetic operations, or conditional generation Wang et al. (2020b); Zheng et al. (2023); Kim et al. (2024); Ha et al. (2023); Zhan et al. (2025). Despite their compactness, these methods target specific design spaces; editing latent codes or programs can cause non-local changes in large irregular lattices.

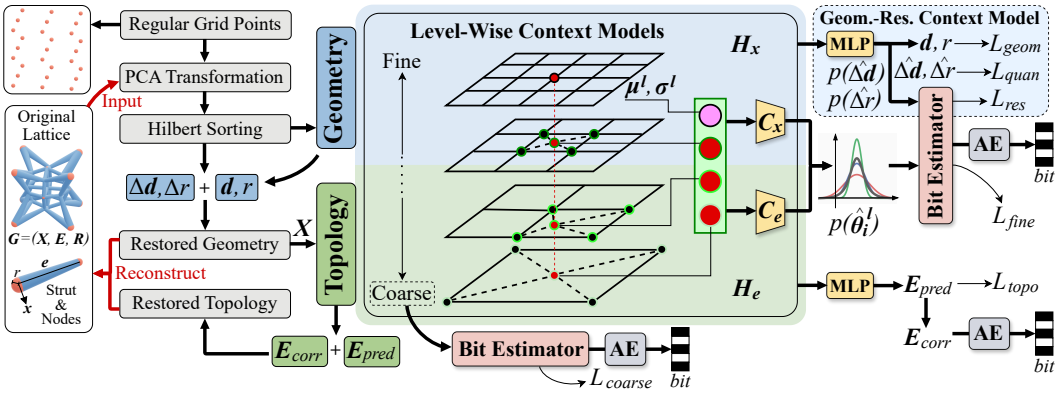


Figure 2: Overview of our CLC framework.

3 METHOD

3.1 PRELIMINARIES

Data Structure of Lattice We represent strut-based lattices as graphs $G = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{R})$, where $\boldsymbol{X} \in \mathbb{R}^{n\times 3}$ is the 3D coordinates of the lattice nodes, $\boldsymbol{E} \in \mathbb{R}^{m\times 2}$ is the edge set representing the connectivity between the lattice nodes, and $\boldsymbol{R} \in \mathbb{R}^{n\times 1}$ is the radius value assigned to each node.

Multi-Resolution Hash Encoding Instant-NGP Müller et al. (2022) represents space with trainable features at progressively finer hash-grid resolutions. For a 3D coordinate $\boldsymbol{x}$, the feature at level $l \in \{1, \ldots, L\}$ interpolates features at the enclosing voxel vertices: $\boldsymbol{f}^l(\boldsymbol{x}) = \text{interp}(\boldsymbol{x}, \Theta^l)$, Here $\Theta^l = \{\boldsymbol{\theta}_i^l = (\theta_{i,1}^l, \ldots, \theta_{i,F}^l) \in \mathbb{R}^F \mid i = 1, \ldots, T^l\}$ is the level-l embedding table, with feature dimension F and size T^l. Spatial hashing maps voxel vertices to a compact table at high resolutions. Concatenating features across levels captures coarse structure and fine detail; we use this encoding for lattice nodes and struts.

3.2 OVERVIEW

Our CLC framework consists of geometric and topological encoding components, as illustrated in Figure 2. At the encoding stage, we first initialize regular grid points with the same cardinality as the lattice nodes, uniformly sample them from $[-1, 1]^3$, transform them into the PCA space of the lattice coordinates, and apply Hilbert sorting to both point sets to establish a bijection. This preprocessing facilitates the learning of the geometric hash encoding H_x.

The geometric branch then uses H_x to learn the displacement field $\boldsymbol{d}$ to the target node coordinates and the node radii r. During training, we exploit contextual dependencies among hash embeddings and train an MLP C_x to model their probability distribution for entropy reduction. Meanwhile, H_x predicts the probability distributions of the geometric residuals $\Delta\boldsymbol{d}$ and Δr, which are later used to restore the original coordinates and radii.

The topological branch takes the restored coordinates as input. Its hash embeddings are combined over the candidate edge set to learn edge connection probabilities, while another MLP C_e models their probability distribution with the same context-modeling strategy. We then compute the correction edge set $\boldsymbol{E}_{corr}$ from the predicted edge set $\boldsymbol{E}_{pred}$ and the ground-truth edge set $\boldsymbol{E}_{gt}$, and further compress $\boldsymbol{E}_{corr}$ with Delta Encoding for lossless topology recovery.

The learned hash embeddings are further pruned using a vertex-activation-based strategy, and the probabilities of multiple vertices mapped to the same hash entry are fused to mitigate hash collisions. Finally, the valid hash embeddings, geometric residuals, and correction edge set are entropy-coded by the Arithmetic Coder (AE), while the parameters of C_x, C_e, the prediction heads of H_x and H_e, and metadata such as PCA parameters are stored in the bitstream. At decoding, the same regular grid points are reconstructed in PCA space, passed through the decoded H_x to recover node

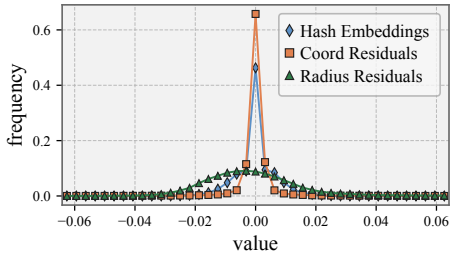


Figure 3: Statistical analysis of the value distributions of the hash embeddings, coordinate and radius residuals. All three components $\{\boldsymbol{\theta}, \Delta\boldsymbol{d}, \Delta r\}$ exhibit Gaussian distributions.

geometry with decoded residuals, and then fed into the decoded H_e; the decoded correction edge set is subsequently applied to recover the lattice topology.

3.3 Hash Embeddings with Context Models

Level-Wise Context Models Inspired by Chen et al. (2024), we design our context models in a level-wise manner to establish contextual dependencies among $\boldsymbol{\theta}_i^l$s across different levels l, with the expectation that more accurately predicted probability $p(\hat{\boldsymbol{\theta}}_i^l)$ leads to greater size reduction. As illustrated in Figure 2, our level-wise context models are designed in a coarse-to-fine manner. Specifically, for a voxel vertex v_i^l with associated feature embedding $\boldsymbol{\theta}_i^l$ at the current level $l = 4$, its context is constructed from the interpolated features at the corresponding spatial location from the previous L_c levels, where L_c is a preset constant (e.g., $L_c = 3$ in Figure 2). We further incorporate the mean and standard deviation of the current level as auxiliary guidance for context modeling. The resulting context features are concatenated as $\boldsymbol{f}_i^l$ and fed into a tiny two-layer MLP, namely C_x or C_e, to estimate the probability $p(\hat{\boldsymbol{\theta}}_i^l)$ at vertex v_i^l.

Feature Quantization For the i-th feature embedding $\boldsymbol{\theta}_i^l$ at level l, we follow Esser et al. (2020) and define the forward quantization process as:

$$\hat{\boldsymbol{\theta}}_i^l = \lfloor \boldsymbol{\theta}_i^l / q_i^l \rceil \cdot q_i^l \tag{1}$$

The gradient through the quantizer round function to the feature embedding is computed using the straight through estimator (STE) Bengio et al. (2013), and the gradient through the quantizer to the step size parameter is computed as:

$$\partial \hat{\boldsymbol{\theta}}_i^l / \partial q_i^l = -\boldsymbol{\theta}_i^l / q_i^l + \lfloor \boldsymbol{\theta}_i^l / q_i^l \rceil \tag{2}$$

The quantization step size q_i^l is refined by a factor o_i^l as:

$$q_i^l = Q_0 \cdot (1 + \tanh(o_i^l)) \tag{3}$$

where Q_0 is the initial quantization step size, and o_i^l is predicted by the context fuser C_x or C_e.

Probability Modeling As depicted in Figure 3, the hash feature embeddings exhibit statistical tendencies of Gaussian distributions. To model the probability of the feature embeddings, we feed the concatenated context feature $\boldsymbol{f}_i^l$ into the context fuser and formulate $p(\hat{\boldsymbol{\theta}}_i^l)$ accordingly:

$$\{\boldsymbol{\mu}_i^l \in \mathbb{R}^F, \boldsymbol{\sigma}_i^l \in \mathbb{R}^F, o_i^l \in \mathbb{R}^1\} = C_{x,e}(\boldsymbol{f}_i^l) \tag{4}$$

$$p(\hat{\boldsymbol{\theta}}_i^l) = \int_{\hat{\boldsymbol{\theta}}_i^l - \frac{q_i^l}{2}}^{\hat{\boldsymbol{\theta}}_i^l + \frac{q_i^l}{2}} \phi(x \mid \boldsymbol{\mu}_i^l, \boldsymbol{\sigma}_i^l) dx \tag{5}$$

where ϕ is the probability density function of the Gaussian distribution, $\boldsymbol{\mu}_i^l$ and $\boldsymbol{\sigma}_i^l$ are the predicted mean and standard deviation of each feature. Consequently, we define a feature-entropy loss as the summation of bit consumption over $\hat{\boldsymbol{\theta}}_i^l$s at fine levels $l = L_c + 1$ to L:

$$L_{fine} = \sum_{l=L_c+1}^{L} \sum_{i=1}^{T^l} \sum_{j=1}^{F} (-\log_2 p(\hat{\theta}_{i,j}^l)) \tag{6}$$

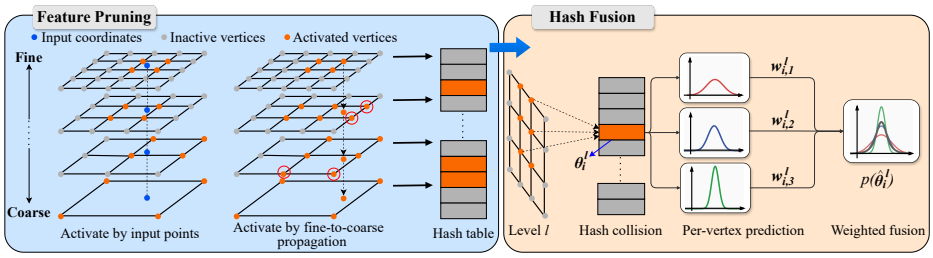


Figure 4: **Feature Pruning**: Only the feature embeddings hashed by valid vertices are retained, where the valid vertices are activated by the input coordinates and the vertices from finer levels. **Hash Fusion**: The predicted probabilities of multiple vertices mapped to the same hash embedding are fused to form a single probability.

As for levels $l = 1$ to L_c, we follow Shin & Park (2023) to binarize the feature $\boldsymbol{\theta}_i^l$ to $\{-1, +1\}$ using a $sign$ function and backpropagating them through STE. Therefore, we define the feature-entropy loss for coarse levels as:

$$L_{coarse} = -n_1 \log_2(\frac{n_1}{n_1 + n_0}) - n_0 \log_2(\frac{n_0}{n_1 + n_0}) \tag{7}$$

where n_1 and n_0 are total counts of $+1$ and -1 in the coarse levels, respectively.

Feature Pruning and Hash Fusion It's worth noting that not all the feature embeddings in the hash table are utilized in the forward interpolation process. As shown in Figure 4, in each level l, the vertices that belong to the voxels bounding the input coordinates are activated first, and we count the number of interpolated coordinate points for each vertex, which is later used as weight for hash fusion. After that, we propagate the voxel-vertex coordinates from each finer level to its previous L_c coarser levels to activate the corresponding vertices which are responsible for constructing the input features for the context fuser. Only the features hashed by the activated vertices are considered valid and would be saved as compression results; invalid features would be discarded.

The hashing indexing of the embedding process introduces an issue of hash collision when building the context models. We propose a hash fusion strategy to resolve this issue, as illustrated in Figure 4. Specifically, suppose a feature embedding $\boldsymbol{\theta}_i^l$ corresponds to K vertices, denoted as $\{v_{i,k}^l \mid k = 1, \ldots, K\}$ (e.g., $K = 3$ in Figure 4). This means for each $\boldsymbol{\theta}_i^l$, multiple probability parameters $\boldsymbol{h}_{i,k}^l \in \{\boldsymbol{\mu}_{i,k}^l, \boldsymbol{\sigma}_{i,k}^l, o_{i,k}^l \mid k = 1, \ldots, K\}$ will be estimated. We use the previously counted coordinate number $n_{i,k}^l$ of each vertex as the weight for hash fusion, which indicates the importance of each vertex in the context model. Consequently, the fused parameters are computed as:

$$\boldsymbol{h}_i^l = \sum_{k=1}^{K} w_{i,k}^l \cdot \boldsymbol{h}_{i,k}^l, \quad w_{i,k}^l = n_{i,k}^l / \sum_{k=1}^{K} n_{i,k}^l \tag{8}$$

And the final fused probability $p(\hat{\boldsymbol{\theta}}_i^l)$ is calculated by equation 5.

3.4 Geometry Restoration and Topology Reconstruction

Using the hash encoding modules H_x and H_e, most geometric and topological information can be retained in the compressed representation. To satisfy the requirements of high-precision 3D printing, we further develop a strategy for high-fidelity geometry restoration and lossless topology reconstruction.

Context Model for Geometric Residuals Geometric residuals include the displacement and the radii error: $\Delta \boldsymbol{d} \in \mathbb{R}^3$, $\Delta r \in \mathbb{R}^1$, which are the differences between the predicted and the ground truth geometry. Since $\Delta \boldsymbol{d}$ and Δr also follow the Gaussian distribution as statistically analyzed in Figure 3, we can use the same context model to predict them. To capture the spatial dependencies among the residuals, we let the hash encoding module H_x additionally predict the distribution parameters of the residuals:

$$\langle \{\boldsymbol{d}, \boldsymbol{\mu}_{\Delta \boldsymbol{d}}, \boldsymbol{\sigma}_{\Delta \boldsymbol{d}}\} \in \mathbb{R}^3, \{o_{\Delta \boldsymbol{d}}, r, \mu_{\Delta r}, \sigma_{\Delta r}, o_{\Delta r}\} \in \mathbb{R}^1 \rangle = MLP(H_x(\boldsymbol{x})) \tag{9}$$

Then, we can model the probability of the residuals by equations 1-5 to get $p(\hat{\Delta}\boldsymbol{d})$ and $p(\hat{\Delta}r)$. And the residual-entropy loss is expressed as:

$$L_{res} = \sum_{i=1}^{N} \big[\frac{1}{3} \sum_{j=1}^{3} \big(-\log_2 p(\hat{\Delta} d_{i,j}) \big) - \log_2 p(\hat{\Delta} r_i) \big] \tag{10}$$

where N is the number of lattice nodes in a batch. To control the quantization error, we further introduce a quantization loss as follows:

$$L_{quan} = MSE(\hat{\Delta}\boldsymbol{d}, \Delta \boldsymbol{d}) + MSE(\hat{\Delta} r, \Delta r) \tag{11}$$

And we also give the geometric loss here for clarity:

$$L_{geom} = MSE(\boldsymbol{d}, \boldsymbol{d}_{gt}) + MSE(r, r_{gt}) \tag{12}$$

Lossless Topology Reconstruction The initial candidate edge set contains n^2 edges, which is too large for a network to learn directly. We therefore employ a heuristic algorithm to search for a much smaller candidate set of edges, which is then fed into the network for training. The implementation details of this heuristic search are provided in the appendix. Once the candidate edge set $\boldsymbol{E}_{cand}$ is obtained, we use it to concatenate the hash embeddings output by the topological hash encoding module H_e:

$$\boldsymbol{f}_e = Concat(H_e(\boldsymbol{x}_1), H_e(\boldsymbol{x}_2), dist(\boldsymbol{x}_1, \boldsymbol{x}_2)) \tag{13}$$

where $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are the coordinates of two lattice nodes connected by the candidate edge. Note that we append the distance between these two coordinates as an auxiliary feature. And we use the Binary Cross-Entropy loss to train the network:

$$L_{topo} = BCELoss(MLP(\boldsymbol{f}_e), \boldsymbol{y}_e) \tag{14}$$

where $\boldsymbol{y}_e$ are the ground-truth labels of the candidate edges. After training, we can use the network to get the predicted edge set $\boldsymbol{E}_{pred}$, and the correction edge set $\boldsymbol{E}_{corr}$ is computed by the boolean operation of $\boldsymbol{E}_{pred}$ and $\boldsymbol{E}_{gt}$:

$$\boldsymbol{E}_{miss} = \boldsymbol{E}_{gt} - \boldsymbol{E}_{pred}, \quad \boldsymbol{E}_{extra} = \boldsymbol{E}_{pred} - \boldsymbol{E}_{gt}, \quad \boldsymbol{E}_{corr} = \boldsymbol{E}_{miss} \cup \boldsymbol{E}_{extra} \tag{15}$$

To efficiently compress the correction edge set, we exploit the locality induced by Hilbert sorting, which assigns nearby node IDs to topologically connected lattice nodes. Specifically, we sort $\boldsymbol{E}_{corr} = \{\boldsymbol{e}_i = (id_{i,1}, id_{i,2}) \in \mathbb{R}^2 \mid i = 1, \ldots, M, id_{i,1} \leq id_{i,2}\}$ in ascending order of $id_{i,1}$. We then perform Delta Encoding across adjacent edges and within each edge:

$$\boldsymbol{D}_{adj} = \{id_{i+1,1} - id_{i,1} \mid i = 1, \ldots, M-1\}, \quad \boldsymbol{D}_{self} = \{id_{i,2} - id_{i,1} \mid i = 1, \ldots, M\} \tag{16}$$

where M is the number of correction edges. $\boldsymbol{D}_{adj}$ and $\boldsymbol{D}_{self}$ are subsequently encoded by the Arithmetic Coder.

3.5 Training and Coding Process

There are two training stages corresponding to the geometric and topological hash encoding modules, respectively. At the geometric training stage, we incorporate the geometric loss, the feature-entropy loss, the residual-entropy loss, and the quantization loss to ensure the model improves fidelity while controlling total bitrate consumption in a differentiable manner. Our overall loss at this stage is:

$$L = L_{geom} + \lambda_{hash}\left(\frac{L_{fine}}{N_{fine}} + \frac{L_{coarse}}{N_{coarse}}\right) + \lambda_{res}\frac{L_{res}}{N} + \lambda_{quan} L_{quan} \tag{17}$$

where N_{fine} and N_{coarse} are the feature number of the fine and coarse levels of the hash grids, respectively. λ_{hash}, λ_{res}, and λ_{quan} are trade-off hyperparameters to balance the rate and fidelity.

At the topological training stage, we combine the topological loss with the feature-entropy loss to ensure the network learns the connections between lattice nodes, and control the storage space simultaneously. The total optimization loss at this stage is:

$$L = L_{topo} + \lambda_{hash}(\frac{L_{fine}}{N_{fine}} + \frac{L_{coarse}}{N_{coarse}}) \tag{18}$$

For the encoding/decoding process, the hash embeddings are first encoded/decoded from coarse to fine levels using level-wise context models. Then, the geometric residuals are encoded/decoded with the assistance of the obtained hash embeddings. The context models are employed to derive the probability for entropy encoding/decoding with Arithmetic Coder. And the correction edge set can be encoded/decoded independently.

4 EXPERIMENTS

4.1 SETUP

All experiments were executed on a computing platform configured with an NVIDIA RTX 4090D GPU (24GB VRAM). The implementations leveraged the PyTorch 2.5.1 framework alongside the CUDA 12.4 acceleration library. We used the AdamW optimizer, with learning rates of 3e-3 for the hash encoding modules and 1e-3 for the level-wise context models. The geometric and topological hash encoding modules were trained for 500 and 300 iterations, respectively. A thorough evaluation was conducted on three benchmark datasets: 1) **Tetgen** This dataset is generated using TetGen Si (2015) for tetrahedral topology construction. 2) **Voronoi** Comprising lattices with Voronoi-diagram topology. 3) **Hexahedron** This dataset encompasses lattices with hexahedral-cell topology. Additional implementation details and the dataset generation process are provided in the appendix.

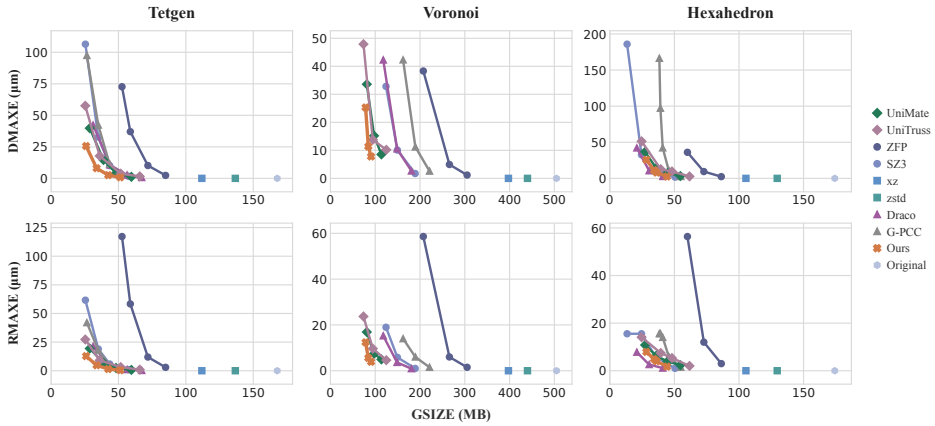
For performance evaluation, we adopted five complementary metrics: 1) **DMAXE (Maximum Displacement Magnitude Error, μm)** This metric measures the maximum Euclidean displacement error between corresponding decompressed and ground-truth lattice coordinates. 2) **RMAXE (Maximum Absolute Error of Radii, μm)** It assesses the maximum absolute error between corresponding decompressed and ground-truth radii. 3) **GSIZE (Geometric Size, MB)** This metric denotes the total size of the geometric coding results, including the model used to encode lattice nodes and the encoded geometric residuals. 4) **TF1L (F1 Loss of Topology)** It is defined as one minus the F1 score between the decompressed and ground-truth edge sets, thereby jointly penalizing missing and extra edges in the recovered lattice topology. 5) **TSIZE (Topological Size, MB)** This metric denotes the total size of the topological coding results, including the model used to encode lattice struts and the encoded correction edge set.

4.2 PERFORMANCE EVALUATION

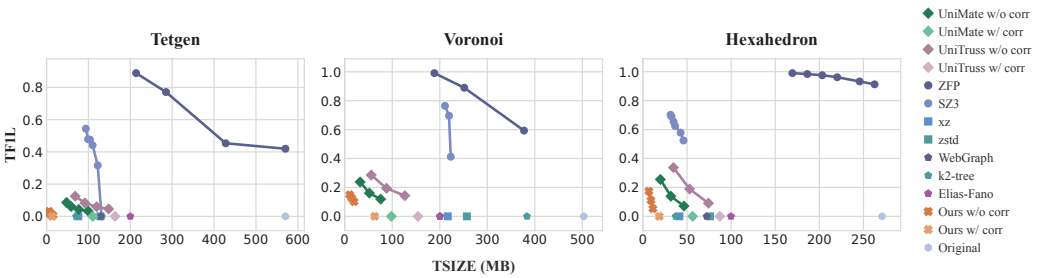
To comprehensively evaluate the performance of our CLC framework, we conducted comparisons against 11 representative compression baselines. Specifically, the selected baselines include: two lattice-specific generative methods UniTruss Zheng et al. (2023) and UniMate Zhan et al. (2025), two lossless generic compressors zstd Collet & Kucherawy (2021) and xz Collin & Pavlov (2024), two lossy generic compressors ZFP Lindstrom (2014) and SZ3 Liang et al. (2023), two geometric compressors Draco Galligan (2017) and G-PCC Graziosi et al. (2020), and three topological compressors WebGraph Boldi & Vigna (2004), k2-tree Brisaboa et al. (2009), and Elias-Fano Vigna (2013). Figure 5 shows the geometric and topological rate-distortion trade-offs; Table 1 reports the corresponding numerical results.

On the Tetgen lattice shown in Figure 1, the previous state-of-the-art method UniMate requires 38.90 MB for geometry and 47.80 MB for topology (86.70 MB in total) at DMAXE = 14.60 μm, RMAXE = 6.90 μm, and TF1L = 0.09 (Table 1). At DMAXE = 11.35 μm, RMAXE = 6.63 μm, and TF1L = 0, CLC requires 30.67 MB for geometry and 13.86 MB for topology (44.53 MB in total; Figure 1), yielding a 1.95$\times$ reduction in total size while improving geometric accuracy and preserving topology losslessly.

Geometric RD curves show smaller CLC reconstruction errors at comparable or lower coding cost than generic and geometry-oriented compressors, especially at low bitrates. For topology, CLC



(a) Geometric RD curves



(b) Topological RD curves

Figure 5: Rate-Distortion (RD) curves of our method and baseline approaches.

Table 1: Comparison of our method against the selected baselines. For all methods with lowrate and highrate variants, lowrate permits larger geometric errors without edge correction, whereas highrate targets higher geometric precision and lossless topology. The best and second-best reported results are in **bold** and underlined, respectively.

Dataset	Tetgen					Voronoi					Hexahedron				
Methods/Metrics	DMAXE	RMAXE	GSIZE	TF1L	TSIZE	DMAXE	RMAXE	GSIZE	TF1L	TSIZE	DMAXE	RMAXE	GSIZE	TF1L	TSIZE
Original Lattice	–	–	167.77	–	570.42	–	–	504.83	–	502.96	–	–	174.16	–	271.07
UniTruss_lowrate	17.80	9.10	36.10	0.13	68.40	13.80	9.65	94.60	0.29	55.70	12.70	7.45	39.40	0.34	34.60
UniTruss_highrate	3.72	3.18	51.70	**0**	163.67	10.20	4.62	124.70	**0**	154.10	2.73	2.02	61.70	**0**	87.19
UniMate_lowrate	14.60	6.90	38.90	0.09	47.80	15.20	7.45	97.50	0.24	32.40	7.45	3.85	44.70	0.25	19.80
UniMate_highrate	4.15	2.45	48.20	**0**	110.51	8.55	4.95	113.80	**0**	98.18	3.22	2.34	54.60	**0**	56.37
SZ3	10.09	5.85	43.54	0.32	122.24	10.10	5.85	149.47	0.41	223.09	10.10	5.85	**35.59**	0.52	45.96
ZFP	10.19	11.95	71.84	0.45	427.82	4.95	6.01	265.63	0.59	377.22	9.42	12.00	72.79	0.98	203.30
xz	**0**	**0**	111.87	**0**	75.95	**0**	**0**	397.46	**0**	216.81	**0**	**0**	105.38	**0**	41.34
zstd	**0**	**0**	136.70	**0**	128.50	**0**	**0**	439.96	**0**	257.01	**0**	**0**	129.60	**0**	76.05
Draco	10.56	4.23	43.89	–	–	10.53	3.63	148.77	–	–	2.64	1.11	40.99	–	–
G-PCC	11.19	4.50	43.48	–	–	11.21	6.00	189.62	–	–	11.24	6.00	45.58	–	–
WebGraph	–	–	–	**0**	129.60	–	–	–	**0**	200.30	–	–	–	**0**	72.23
k2-tree	–	–	–	**0**	70.67	–	–	–	**0**	383.43	–	–	–	**0**	37.12
Elias-Fano	–	–	–	**0**	200.07	–	–	–	**0**	200.35	–	–	–	**0**	99.81
Ours_lowrate	8.05	4.77	**33.82**	0.03	**5.90**	11.38	5.59	**84.61**	0.15	**10.50**	8.02	4.00	36.20	0.17	**6.76**
Ours_highrate	2.54	1.48	42.47	**0**	13.52	7.81	3.96	90.73	**0**	63.01	2.55	1.73	44.30	**0**	18.59

achieves substantially smaller coding size while maintaining near-lossless or lossless structural accuracy. Further metrics and visual comparisons are provided in the appendix.

Figure 6 compares local geometric reconstructions and their deviation heatmaps. The qualitative comparisons show that our method preserves fine node geometry more faithfully and produces smaller local deviations than the competing baselines.

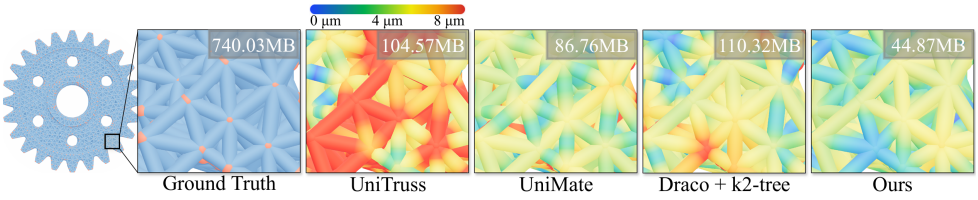


Figure 6: Qualitative comparison of local geometric reconstruction quality. Warmer colors in the deviation heatmaps indicate larger geometric errors.

4.3 Ablation Studies

We conduct ablation studies on three components: 1) **level-wise context models**, 2) **context model for geometric residuals**, and 3) **feature pruning**. Table 2(a) reports the first two ablations: *Ours* is the full model; *w/o LW (full/16/8/4)* removes the level-wise context models and uses full-precision or fixed 16/8/4-bit hash coding; *w/o GR (12/8/4)* removes the residual context model and instead applies fixed 12/8/4-bit quantization to geometric residuals; and *w/o LW&GR (8+8/4+8)* removes both context modules, using fixed 8/4-bit hash coding together with fixed 8-bit geometric residual coding. Table 2(b) reports the impact of feature pruning on total compressed size.

Table 2: Ablation results. (a) Ablation on level-wise context models and the context model for geometric residuals. (b) Ablation on feature pruning, reported by total compressed size on the three datasets.

(a) Context-model ablation

Method	DMAXE	RMAXE	GSIZE	TSIZE
Ours	11.35	6.63	30.67	13.86
w/o LW (full)	11.36	6.65	64.91	58.35
w/o LW (16)	11.36	6.55	45.74	30.28
w/o LW (8)	11.40	6.66	33.89	14.28
w/o LW (4)	11.34	6.61	31.37	17.74
w/o GR (8)	11.40	6.65	30.75	14.34
w/o GR (4)	195.12	104.35	11.77	248.53
w/o LW&GR (8+8)	11.34	6.71	34.04	14.13
w/o LW&GR (4+8)	11.38	6.75	31.15	17.69

(b) Feature-pruning ablation

Method	Tetgen	Voronoi	Hexahedron
w/ feature pruning	44.29	153.71	52.8
w/o feature pruning	77.46	196.14	75.73

Overall, both context modules are important for a favorable rate-distortion trade-off. Removing the level-wise context models degrades compression efficiency across coding settings, while replacing the residual context model with fixed residual coding exposes a fidelity-size trade-off across bit widths. Removing both context modules further confirms that the two context designs contribute complementary compression gains.

Feature pruning also consistently reduces total compressed size, confirming that pruning inactive hash entries improves compactness.

5 Conclusion

We presented CLC, a context-based lattice compression framework for reducing the storage cost of large-scale lattice structures. By combining multi-resolution hash embeddings with learned context models for geometry, topology, and residual prediction, CLC improves compactness while preserving reconstruction quality. Experiments on three benchmark datasets show that CLC consistently outperforms existing baselines in compression efficiency while maintaining high geometric fidelity and lossless topology.

AI use statement

Generative AI tools assisted with language editing, bibliographic metadata verification, and LaTeX formatting of this manuscript. The authors reviewed the AI-assisted work and take responsibility for

the final content, including all technical claims, experimental results, mathematical formulations, figures, tables, and references.

Ethics statement

This work studies compression methods for lattice graph representations and does not involve human subjects, personal data, or sensitive user data. The dataset descriptions and preprocessing details in the appendix support reproducible evaluation of the proposed method.

Reproducibility statement

We provide implementation details, preprocessing procedures, candidate-edge construction, notation, additional visual results, and dataset descriptions in the appendix. The benchmark data are represented as lattice graphs with node coordinates, node radii, and edge connectivity, matching the representation used throughout training, compression, decoding, and evaluation.

References

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. URL https://arxiv.org/abs/1308.3432.

Paolo Boldi and Sebastiano Vigna. The webgraph framework I: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*, pp. 595–602, 2004. doi: 10.1145/988672.988752.

Nieves R. Brisaboa, Susana Ladra, and Gonzalo Navarro. k2-trees for compact web graph representation. In *Proceedings of the 16th International Symposium on String Processing and Information Retrieval*, pp. 18–30, 2009. doi: 10.1007/978-3-642-03784-9_3.

Yihang Chen, Qianyi Wu, Mehrtash Harandi, and Jianfei Cai. How far can we compress instant-ngp-based nerf? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20321–20330, 2024. doi: 10.1109/CVPR52733.2024.01921.

Laurent Chougrani, Jean-Philippe Pernot, Philippe Véron, and Stéphane Abed. Lattice structure lightweight triangulation for additive manufacturing. *Computer-Aided Design*, 90:95–104, 2017. doi: 10.1016/j.cad.2017.05.016.

Yann Collet and Murray Kucherawy. Zstandard compression and the 'application/zstd' media type. RFC 8878, 2021. URL https://www.rfc-editor.org/info/rfc8878.

Lasse Collin and Igor Pavlov. The .xz file format, 2024. URL https://tukaani.org/xz/xz-file-format.txt. Version 1.2.1.

Guoying Dong, Yunlong Tang, and Yaoyao Fiona Zhao. A survey of modeling of lattice structures fabricated by additive manufacturing. *Journal of Mechanical Design*, 139(10):100906, 2017. doi: 10.1115/1.4037305.

Anton du Plessis, Nima Razavi, Matteo Benedetti, Simone Murchio, Martin Leary, Marcus Watson, Dhruv Bhate, and Filippo Berto. Properties and applications of additively manufactured metallic cellular materials: A review. *Progress in Materials Science*, 125:100918, 2022. doi: 10.1016/j.pmatsci.2021.100918.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgO66VKDS.

Frank Galligan. Draco bitstream specification, 2017. URL https://google.github.io/draco/spec/. Version 2.2, Google.

D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai. An overview of ongoing point cloud compression standardization activities: Video-based (V-PCC) and geometry-based (G-PCC). *APSIPA Transactions on Signal and Information Processing*, 9(1):e13, 2020. doi: 10.1017/ATSIP.2020.12.

Ashish Gupta, George Allen, and Jarek Rossignac. Exact representations and geometric queries for lattice structures with quador beams. *Computer-Aided Design*, 115:64–77, 2019a. doi: 10.1016/j.cad.2019.05.035.

Ashish Gupta, Kelsey Kurzeja, Jarek Rossignac, George Allen, Pranav Srinivas Kumar, and Suraj Musuvathy. Programmed-lattice editor and accelerated processing of parametric program-representations of steady lattices. *Computer-Aided Design*, 113:35–47, 2019b. doi: 10.1016/j.cad.2019.04.001.

Chan Soo Ha, Desheng Yao, Zhenpeng Xu, Chenang Liu, Han Liu, Daniel Elkins, Matthew Kile, Vikram Deshpande, Zhenyu Kong, Mathieu Bauchy, and Xiaoyu Rayne Zheng. Rapid inverse design of metamaterials based on prescribed mechanical behavior through machine learning. *Nature Communications*, 14:5765, 2023. doi: 10.1038/s41467-023-40854-1.

Q Youn Hong and Gershon Elber. Conformal microstructure synthesis in trimmed trivariate based v-reps. *Computer-Aided Design*, 140:103085, 2021. doi: 10.1016/j.cad.2021.103085.

Namjung Kim, Dongseok Lee, Chanyoung Kim, Dosung Lee, and Youngjoon Hong. Simple arithmetic operation in latent space can generate a novel three-dimensional graph metamaterials. *npj Computational Materials*, 10(1):236, 2024. doi: 10.1038/s41524-024-01430-3.

Kelsey Kurzeja and Jarek Rossignac. Becots: Bent corner-operated tran-similar maps and lattices. *Computer-Aided Design*, 129:102912, 2020. doi: 10.1016/j.cad.2020.102912.

Nikita Letov and Yaoyao Fiona Zhao. Beam-based lattice topology transition with function representation. *Journal of Mechanical Design*, 145(1):011704, 2023. doi: 10.1115/1.4055950.

Nikita Letov and Yaoyao Fiona Zhao. Geometric modeling of advanced cellular structures with skeletal graphs. *International Journal of Mechanical Sciences*, 270:109087, 2024. doi: 10.1016/j.ijmecsci.2024.109087.

Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, Dingwen Tao, Zizhong Chen, and Franck Cappello. Sz3: A modular framework for composing prediction-based error-bounded lossy compressors. *IEEE Transactions on Big Data*, 9(2):485–498, 2023. doi: 10.1109/TBDATA.2022.3201176.

Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014. doi: 10.1109/TVCG.2014.2346458.

Yuan Liu, Shurong Zhuo, Yining Xiao, Guolei Zheng, Guoying Dong, and Yaoyao Fiona Zhao. Rapid modeling and design optimization of multi-topology lattice structure based on unit-cell library. *Journal of Mechanical Design*, 142(9):091705, 2020. doi: 10.1115/1.4046812.

Yuan Liu, Guolei Zheng, Nikita Letov, and Yaoyao Fiona Zhao. A survey of modeling and optimization methods for multi-scale heterogeneous lattice structures. *Journal of Mechanical Design*, 143(4):040803, 2021. doi: 10.1115/1.4047917.

Marco Maurizi, Derek Xu, Yu-Tong Wang, Desheng Yao, David Hahn, Mourad Oudich, Anish Satpati, Mathieu Bauchy, Wei Wang, Yizhou Sun, Yun Jing, and Xiaoyu Rayne Zheng. Designing metamaterials with programmable nonlinear responses and geometric constraints in graph space. *Nature Machine Intelligence*, 7:1023–1036, 2025. doi: 10.1038/s42256-025-01067-x.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):102:1–102:15, 2022. doi: 10.1145/3528223.3530127.

Aamer Nazir, Kalayu Mekonen Abate, Ajeet Kumar, and Jeng-Ywan Jeng. A state-of-the-art review on types, design, optimization, and additive manufacturing of cellular structures. *The International Journal of Advanced Manufacturing Technology*, 104(9–12):3489–3510, 2019. doi: 10.1007/s00170-019-04085-3.

Jordan Noronha, Jason Dash, Martin Leary, Marcus Watson, Ma Qian, Elizabeth Kyriakou, and Milan Brandt. Additively manufactured functionally graded lattices: Design, mechanical response, deformation behavior, applications, and insights. *JOM*, 75(12):5729–5754, 2023. doi: 10.1007/s11837-023-06190-x.

Gianpaolo Savio, Roberto Meneghello, and Gianmaria Concheri. Geometric modeling of lattice structures for additive manufacturing. *Rapid Prototyping Journal*, 24(2):351–360, 2018. doi: 10.1108/RPJ-07-2016-0122.

Seungjoo Shin and Jaesik Park. Binary radiance fields. *Advances in neural information processing systems*, 36:55919–55931, 2023. doi: 10.52202/075280-2438.

Hang Si. Tetgen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software*, 41(2):11:1–11:36, 2015. doi: 10.1145/2629697.

Yunlong Tang, Guoying Dong, and Yaoyao Fiona Zhao. A hybrid geometric modeling method for lattice structures fabricated by additive manufacturing. *The International Journal of Advanced Manufacturing Technology*, 102(9–12):4011–4030, 2019. doi: 10.1007/s00170-019-03308-x.

Boris van Sosin, Daniil Rodin, Hanna Sliusarenko, Michael Bartoň, and Gershon Elber. The construction of conforming-to-shape truss lattice structures via 3d sphere packing. *Computer-Aided Design*, 132:102962, 2021. doi: 10.1016/j.cad.2020.102962.

Chaman Singh Verma, Behzad Rankouhi, and Krishnan Suresh. A combinatorial approach for constructing lattice structures. *Journal of Mechanical Design*, 142(4):041404, 2020. doi: 10.1115/1.4044521.

Sebastiano Vigna. Quasi-succinct indices. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, pp. 83–92, 2013. doi: 10.1145/2433396.2433409.

Liwei Wang, Yu-Chin Chan, Faez Ahmed, Zhao Liu, Ping Zhu, and Wei Chen. Deep generative modeling for mechanistic-based learning and design of metamaterial systems. *Computer Methods in Applied Mechanics and Engineering*, 372:113377, 2020a. doi: 10.1016/j.cma.2020.113377.

Liwei Wang, Yu-Chin Chan, Zhao Liu, Ping Zhu, and Wei Chen. Data-driven metamaterial design with laplace-beltrami spectrum as "shape-dna". *Structural and Multidisciplinary Optimization*, 61(6):2613–2628, 2020b. doi: 10.1007/s00158-020-02523-5.

Yaohong Wu, Ashish Gupta, Kelsey Kurzeja, and Jarek Rossignac. Chocc: Convex hull of cospherical circles and applications to lattices. *Computer-Aided Design*, 129:102903, 2020. doi: 10.1016/j.cad.2020.102903.

Wangzhi Zhan, Jianpeng Chen, Dongqi Fu, and Dawei Zhou. Unimate: A unified model for mechanical metamaterial generation, property prediction, and condition confirmation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 74406–74421, 2025. URL `https://proceedings.mlr.press/v267/zhan25b.html`.

Xin Zhao and Jinjie Huang. Lightweight geometric compression encoding for additive manufacturing. *Computer-Aided Design*, 165:103612, 2023. doi: 10.1016/j.cad.2023.103612.

Li Zheng, Konstantinos Karapiperis, Siddhant Kumar, and Dennis M. Kochmann. Unifying the design space and optimizing linear and nonlinear truss metamaterials by generative modeling. *Nature Communications*, 14:7563, 2023. doi: 10.1038/s41467-023-42068-x.

Qiang Zou and Guoyue Luo. Geometric modeling for microstructure design and manufacturing: A review of representations and modeling algorithms. *Computer-Aided Design*, 180:103834, 2025. doi: 10.1016/j.cad.2024.103834.

Qiang Zou, Yunzhu Gao, Guoyue Luo, and Sifan Chen. Meta-meshing and triangulating lattice structures at a large scale. *Computer-Aided Design*, 174:103732, 2024. doi: 10.1016/j.cad.2024.103732.

A More Implementation Details

A.1 Hash Encoding Modules

For the hash encoding modules H_x and H_e, we set the number of multi-resolution levels L to 12 and 16, respectively. Both modules use feature dimension $F = 4$, maximum hash table size $T = 2^{19}$, and context depth $L_c = 4$, with the maximum hash-grid resolution scaled proportionally to $n^{\frac{1}{3}}$, where n is the number of lattice nodes. Their prediction heads are implemented as 3-layer MLPs P_x and P_e. The input dimension of P_x is $L \cdot F$, whereas that of P_e is $2 \cdot L \cdot F + 1$, where $2\cdot$ accounts for the two endpoint nodes and $+1$ corresponds to their distance. The output dimension of P_x is 14, including the predictions of $\boldsymbol{d}$ and r, the mean and standard deviation of $\Delta\boldsymbol{d}$ and Δr, and the quantization refinement factors o for $\Delta\boldsymbol{d}$ and Δr. In contrast, P_e outputs a single value representing the connection probability of each candidate edge. Both P_x and P_e use hidden dimension 68, SiLU activation, and layer normalization.

A.2 Context Fusers

The context fusers C_x and C_e share the same architecture. Each fuser takes an input dimension of $F \cdot L_c + 2$, uses a hidden dimension of 32, and produces an output of dimension $2 \cdot F + 1$ with LeakyReLU activation. Here, the additional $+2$ in the input corresponds to the mean and standard deviation of the current level, which are provided as auxiliary guidance. In the output, $2 \cdot F$ represents the predicted mean and standard deviation for each feature dimension, while the remaining $+1$ corresponds to the quantization refinement factor o for the feature vector.

A.3 Asynchronous Update Strategy

Since the level-wise context models predict probability parameters and quantize hash embeddings in an autoregressive coarse-to-fine manner, performing synchronous backpropagation together with each batch of lattice nodes or edges would be time-consuming. We therefore update the level-wise context models asynchronously. During the iterations when they are not updated, hash quantization and entropy loss are computed using the cached outputs of the level-wise context models. The update interval is set to 30 iterations.

A.4 Sample Strategy

During training, feeding all embeddings into the level-wise context models in a single iteration would cause out-of-memory (OOM). To reduce memory usage while preserving sufficient training coverage, we randomly sample 20% of the feature embeddings $\boldsymbol{\theta}$ in each iteration.

B Preprocessing of Regular Grid Points

For regular grid point initialization, we first compute the PCA parameters of the lattice coordinates $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^n$ as:

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i, \\ \boldsymbol{\Sigma} &= \frac{1}{n-1}(\boldsymbol{X} - \mathbf{1}\boldsymbol{\mu}^\top)^\top(\boldsymbol{X} - \mathbf{1}\boldsymbol{\mu}^\top), \\ \boldsymbol{\Sigma} &= \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top, \\ \boldsymbol{\sigma} &= \sqrt{\mathrm{diag}(\boldsymbol{\Lambda})}, \end{aligned} \tag{19}$$

where $\boldsymbol{\mu}$, $\boldsymbol{U}$, and $\boldsymbol{\sigma}$ denote the centroid, principal directions, and standard deviations along the principal axes, respectively.

Given regular grid points $\boldsymbol{P} \in [-1,1]^{n\times 3}$ centered at the origin, we transform them to the PCA space by

$$\boldsymbol{P}' = \boldsymbol{P}\,\mathrm{diag}(\sqrt{3}\,\boldsymbol{\sigma})\,\boldsymbol{U}^\top + \mathbf{1}\boldsymbol{\mu}^\top, \tag{20}$$

where $\boldsymbol{P}'$ denotes the transformed regular grid points in the PCA space. The factor $\sqrt{3}$ compensates for the standard deviation of points uniformly sampled from $[-1,1]^3$ along each axis.

After that, we perform Hilbert sorting for both the regular grid points and the lattice nodes by their coordinates to establish a bijection between them.

C Heuristic Search for Candidate Edges

As discussed in the main text, the complete pairwise edge space contains n^2 possibilities and is therefore too large to be handled directly by the topology network. We therefore construct a much smaller candidate edge set $\boldsymbol{E}_{cand}$ using an adaptive KNN-based heuristic, and only this set is passed to the network for topology prediction. The purpose of this stage is not to fully recover the topology by itself, but to retain most ground-truth edges while keeping $\boldsymbol{E}_{cand}$ sufficiently compact for learning.

We first precompute exact Euclidean KNN lists for all lattice nodes. Based on these neighbor lists, the heuristic determines for each node how many nearby neighbors should be kept and which of them should be discarded as geometrically implausible. To this end, we estimate a local density score for node i by

$$\rho_i = \frac{1}{\frac{1}{K_\rho}\sum_{j=1}^{K_\rho} d_{ij} + \varepsilon}, \tag{21}$$

where d_{ij} denotes the distance from node i to its j-th nearest neighbor, and K_ρ is the number of nearest neighbors used for density estimation. A reference density ρ_{ref} is then obtained from a percentile of $\{\rho_i\}_{i=1}^n$. The initial adaptive neighborhood size is defined as

$$\tilde{k}_i = \mathrm{clip}\left(\mathrm{round}\left(k_0\left(\frac{\rho_i}{\rho_{\mathrm{ref}}}\right)^{\alpha}\right), 1, k_{\max}\right), \tag{22}$$

where k_0 is the base neighborhood size, α controls the strength of density adaptation, and $k_{\max}$ limits the largest allowed neighborhood. In this way, denser regions are allowed to keep more local neighbors, whereas sparser regions remain compact.

The above neighborhood is further refined in two steps. First, we detect abrupt jumps in the ordered neighbor distances. If the ratio between two consecutive distances exceeds a threshold before reaching $\tilde{k}_i$, the retained neighborhood is truncated at the first such jump, yielding the final neighborhood size k_i. Second, we compute a local geometric scale

$$s_i = \mathrm{median}(d_{i1}, \ldots, d_{iK_s}) + \varepsilon, \tag{23}$$

where K_s is the number of nearest neighbors used to estimate the local scale. A neighbor j is kept only if it satisfies

$$j < k_i \quad \text{and} \quad d_{ij} \le \tau_s s_i, \tag{24}$$

where τ_s is a distance-ratio threshold. The surviving directed neighbor pairs are finally converted to undirected edges and deduplicated to form $\boldsymbol{E}_{cand}$.

The search parameters are selected automatically by a constrained objective that explicitly balances coverage and compactness. Given the ground-truth edge set $\boldsymbol{E}_{gt}$ and a candidate set $\boldsymbol{E}_{cand}$, we compute

$$\begin{aligned} \boldsymbol{E}_{hit} &= \boldsymbol{E}_{gt} \cap \boldsymbol{E}_{cand}, \\ \boldsymbol{E}_{miss} &= \boldsymbol{E}_{gt} - \boldsymbol{E}_{cand}, \\ \boldsymbol{E}_{extra} &= \boldsymbol{E}_{cand} - \boldsymbol{E}_{gt}. \end{aligned} \tag{25}$$

Accordingly, the recall and precision are defined as

$$\mathrm{Recall} = \frac{|\boldsymbol{E}_{hit}|}{|\boldsymbol{E}_{gt}|}, \qquad \mathrm{Precision} = \frac{|\boldsymbol{E}_{hit}|}{|\boldsymbol{E}_{cand}|}. \tag{26}$$

The objective is constructed as a constrained compactness criterion. When the recall reaches a prescribed target, the optimization mainly minimizes $|\boldsymbol{E}_{cand}|$, while adding a small penalty on $|\boldsymbol{E}_{extra}|$ to suppress unnecessary redundant edges. Otherwise, the objective assigns a large infeasibility penalty and further penalizes the recall violation together with $|\boldsymbol{E}_{miss}|$. Therefore, the search first enforces sufficient coverage of the ground-truth topology, and only then pushes the candidate set to be as small as possible.

After all trials are completed, the final solution is chosen by prioritizing smaller $|\boldsymbol{E}_{cand}|$, then fewer extra edges, and finally higher recall among feasible trials. If no trial satisfies the recall requirement, the highest-recall trial is returned, with ties broken by smaller candidate size and fewer extra edges. In this way, the resulting candidate set remains compact enough for efficient learning while still covering the vast majority of ground-truth connections required for subsequent topology reconstruction.

D Data Appendix

The benchmark datasets introduced in this work are included with the supplementary material. Each dataset is represented as a lattice graph with three components: node coordinates $\boldsymbol{X} \in \mathbb{R}^{n\times 3}$, node radii $\boldsymbol{R} \in \mathbb{R}^{n\times 1}$, and edge connectivity $\boldsymbol{E} \in \mathbb{R}^{m\times 2}$. This representation matches the data structure defined in Section 3.1 and is the format used throughout training, compression, and evaluation.

For reproducibility, we provide the processed lattice data used in the experiments together with the corresponding supplementary assets needed by our pipeline. Specifically, the released files include the lattice graph in human-readable text form and compact binary form, and, when applicable, the intermediate mesh files and scripts used to construct the evaluated lattice instances. These files are sufficient to reproduce the preprocessing, encoding, decoding, and metric computation procedures reported in the paper.

Specifically, the three benchmark datasets **Tetgen**, **Voronoi**, and **Hexahedron** are generated by the following steps:

D.1 Tetgen

We first generate the tetrahedral mesh using TetGen and then subdivide its cells into smaller tetrahedra. The subdivided mesh is then converted into a lattice graph by taking mesh vertices as nodes and mesh edges as struts. Node radii are assigned from feasible intervals determined by local incident-edge lengths, so that the radii remain compatible with the surrounding strut geometry.

D.2 Voronoi

We start from a watertight STL surface and preprocess it according to the target lattice resolution. We then use TetGen with a searched element-volume parameter to obtain a tetrahedral mesh whose vertex count is close to the prescribed target, and construct a Voronoi-like lattice from its circumcenter dual, where tetrahedra become nodes and shared interior faces define struts. To keep the dual graph geometrically valid, circumcenters lying outside the shape are removed using boundary half-space tests, long dangling degree-one spikes are pruned, and only those locally anomalous short edges that would violate the radius-feasibility constraint are further discarded. Node radii are finally sampled from feasible intervals induced by the incident strut lengths, so that the local strut-to-radius ratios remain valid.

D.3 Hexahedron

We start from a watertight STL surface and fill its enclosed volume with a Cartesian hexahedral grid, where the voxel pitch is adjusted to match the target lattice scale. We then simplify the occupied fine grid into a geometry-aware adaptive octree representation, so that interior regions can be merged more aggressively while cells near the surface or high-curvature areas remain finer. The centers of the adaptive hex cells are taken as lattice nodes, and face adjacencies between them define the struts. Node radii are finally sampled from a scale-consistent interval determined by the grid pitch.

Table 3: Notation table.

Notation	Definition
$G = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{R})$	Lattice graph with node coordinates, edge connectivity, and node radii.
$\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{R}$	Node coordinates, edge set, and node radii of the lattice graph.
n, m	Numbers of lattice nodes and edges.
$\boldsymbol{x}_i$	Coordinate of the i-th lattice node.
$\boldsymbol{d}, r$	Predicted node displacement and radius in the geometric branch.
$\Delta\boldsymbol{d}, \Delta r$	Geometric residuals for displacement and radius restoration.
H_x, H_e	Geometric and topological hash encoding modules.
C_x, C_e	Level-wise context fusers for geometric and topological embeddings.
L, l	Total number of hash-grid levels and the current level index.
F, T^l	Feature dimension and embedding-table size at level l.
$\Theta^l, \boldsymbol{\theta}_i^l$	Embedding table and the i-th feature embedding at level l.
L_c	Number of preceding levels used for context modeling.
q_i^l, Q_0, o_i^l	Quantization step, its initial value, and the learned refinement factor.
$\boldsymbol{\mu}_i^l, \boldsymbol{\sigma}_i^l$	Predicted Gaussian mean and standard deviation for $\boldsymbol{\theta}_i^l$.
L_{fine}, L_{coarse}	Entropy losses for the fine and coarse hash-grid levels.
$\boldsymbol{E}_{gt}, \boldsymbol{E}_{cand}$	Ground-truth edge set and the searched candidate edge set.
$\boldsymbol{E}_{pred}, \boldsymbol{E}_{corr}$	Predicted edge set and correction edge set for lossless topology recovery.
$\boldsymbol{E}_{hit}$	Candidate edges that overlap with the ground-truth edge set.
$\boldsymbol{E}_{miss}, \boldsymbol{E}_{extra}$	Missing and extra edges induced by set comparison.
$\boldsymbol{D}_{adj}, \boldsymbol{D}_{self}$	Delta-encoded differences across adjacent correction edges and within each edge.
$\boldsymbol{y}_e$	Ground-truth labels of candidate edges for topology training.
L_{res}, L_{quan}	Residual-entropy and quantization losses in geometry restoration.
L_{geom}, L_{topo}	Geometry and topology supervision losses.
$\lambda_{hash}, \lambda_{res}$	Trade-off hyperparameters balancing rate and reconstruction fidelity.
λ_{quan}	Quantization-loss weight in geometry restoration.
$\boldsymbol{P}, \boldsymbol{P}'$	Regular grid points before and after PCA-space transformation.
$\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{U}, \boldsymbol{\Lambda}, \boldsymbol{\sigma}$	PCA centroid, covariance, principal directions, eigenvalues, and axis-wise standard deviations.
$\rho_i, \rho_{\text{ref}}$	Local density estimate of node i and the reference density for adaptation.
K_ρ, K_s	Numbers of nearest neighbors used for density and local-scale estimation.
$k_0, \tilde{k}_i, k_i, k_{\max}$	Base, initial adaptive, final, and maximum neighborhood sizes in candidate-edge search.
s_i, τ_s	Local geometric scale and distance-ratio threshold used to prune implausible neighbors.

E Notation Table and More Results

E.1 Notation Table

We summarize the main notation used throughout the main paper and appendix in Table 3.

E.2 More Results

Figure 7 provides additional visual comparisons on the three benchmark datasets used in our experiments, namely **Tetgen**, **Voronoi**, and **Hexahedron**. To make the local structural differences and deviation heatmaps easier to inspect, the lattices shown in this figure are sparsified versions derived from the original evaluation data for visualization only. This sparsification is used solely for display clarity and does not change the quantitative results reported in the main paper.

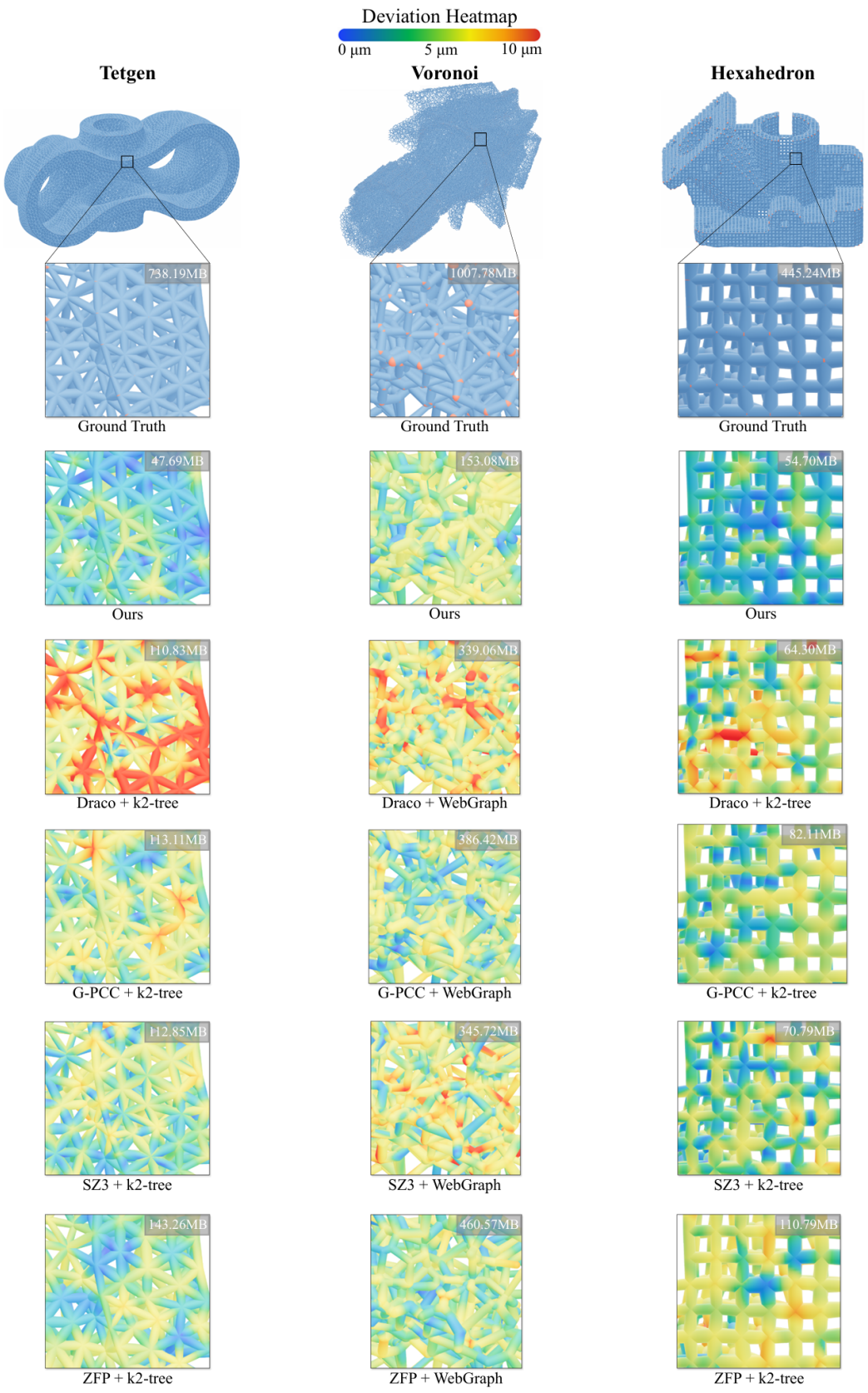


Figure 7: Additional visual comparisons on Tetgen, Voronoi, and Hexahedron.

E.3 Encoding and Decoding Time

Table 4: Logged component times for CLC on the Tetgen lattice in Figure 1.

Component	Time (s)
Geometric training + coding	1452.73
Candidate-edge KNN upper-bound search	1608.17
Topological training + coding	1575.19
Geometric decoding	81.72
KNN precomputation before topological decoding	1300.19
Topological decoding	112.53

Table 5: Encoding and decoding time comparison on the Tetgen lattice in Figure 1.

Method	Encoding time (s)	Decoding time (s)
UniTruss	4120.53	2100.1
UniMate	3680.29	1883.45
zstd	2.80	1.02
xz	2.33	1.85
ZFP	2.99	1.53
SZ3	2.23	1.34
Draco	10.35	5.21
G-PCC	14.67	8.65
WebGraph	2100.34	1034.45
k2-tree	1654.87	1281.50
Elias-Fano	2311.70	1625.51
Ours	4636.09	1494.44

F Limitations

Runtime remains a limitation of the current CLC implementation on large lattices. For the Tetgen lattice, topological network training and candidate-edge search incur substantial encoding costs, while neighborhood precomputation contributes to decoding time (Table 4). Future work could investigate parallel compression and decompression operators, including parallel candidate-edge construction and context-model evaluation, to reduce these costs.