Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Post-Hoc Entropy Coding for Anchor-Based 3D Gaussian Splatting with an Optional Joint-Training Path

Chi Li

lichi@dlu.edu.cn

Dalian University

Shixi Wen

Dalian University

Yuan Zhao

Dalian University

Jian Liu

Dalian University

Research Article

Keywords: 3D Gaussian Splatting, model compression, knowledge distillation, rate–distortion optimization, entropy coding

Posted Date: November 3rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11056075/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Post-Hoc Entropy Coding for Anchor-Based 3D Gaussian Splatting with an Optional Joint-Training Path

Chi Li[1*], Shixi Wen[1], Yuan Zhao[1], Jian Liu[1]

[1*]School of Information Engineering, Dalian University, Dalian,116622, Liaoning P.R.China.

*Corresponding author(s). E-mail(s): lichi@dlu.edu.cn;

Abstract

Anchor-based 3D Gaussian splatting enables real-time rendering, but trained scenes remain too large for phones, headsets, and edge servers. Existing post-hoc converters are drop-in yet saturate early, while joint-training codecs compress deeper only by retraining from scratch. We present a flexible compression framework whose primary mode is a strictly post-hoc learned entropy codec for anchor-based Gaussian splatting. The codec treats any trained model as a read-only artifact and learns a per-scene binary hash-grid entropy context with per-channel Laplace distributions. Attributes are coded into a Morton-ordered, progressively decodable bitstream, and the codec requires neither training data nor pipeline access. When the training pipeline is available, the same bitstream format optionally supports a joint-training variant, whose quantization simulation, rate–distortion optimization, and hard straight-through anchor mask align training with deployment. Across three benchmarks and thirteen scenes, the post-hoc mode substantially reduces storage relative to float16 and codebook baselines while preserving quality. It also transfers unchanged to third-party checkpoints and adds no rendering overhead under progressive decoding. The joint mode further increases compression at a modest quality cost and serves as a controlled probe of what retraining adds, though dedicated joint codecs remain stronger in the deepest regime. The framework thus turns post-hoc and joint compression into selectable rungs of one deployable format and decoder.

Keywords: 3D Gaussian Splatting, model compression, knowledge distillation, rate–distortion optimization, entropy coding

1 Introduction

Photo-realistic rendering underlies virtual reality, autonomous-driving simulation, and digital twins, all of which must run in real time on end-user hardware. 3D Gaussian Splatting [1] meets the speed requirement with an explicit primitive-based representation, but a trained scene occupies far more storage than mobile and embedded devices can afford. Anchor-based variants such as Scaffold-GS [2] restructure the representation around a sparse voxel grid of anchors, each decoded by lightweight MLPs into a small set of neural Gaussians. The structure renders more sharply from fewer primitives, yet the richer per-anchor record keeps a scene large, and pushing it to a phone, a VR/AR headset, or an edge server over a constrained link remains the step where deployment stalls.

Two families of methods address this problem, with complementary trade-offs. The first converts the trained weights. Quantization and codebook pipelines [3–6] map attributes to float16, scalar grids, or vector-quantized codebooks; feedforward codecs [7] amortize a scene-agnostic network across scenes and compress each scene in seconds. Both are drop-in and safe at deployment, yet the redundancy they exploit was frozen when training ended, so the compression ratio saturates early and distortion rises sharply at lower bit budgets. The second embeds compression into optimization itself. Rate–distortion joint training [8, 9], in particular the HAC line [10, 11], combines quantization simulation, learned entropy contexts, and anchor masking to compress an order of magnitude deeper. A model retrained for depth is no longer the model that was validated; a model kept without retraining remains at its original size.

The vast majority of deployed models cannot be retrained; for them, the post-hoc compression ceiling is a hard boundary rather than a corner case. We raise this ceiling with a flexible framework whose primary mode is a strictly post-hoc learned entropy codec. It treats any trained anchor model as a read-only artifact, consumes only its anchor schema without training data or pipeline access, and fits a per-scene entropy context in minutes on a single GPU. Unlike fixed-grid and codebook conversions, it learns each scene's attribute statistics; unlike joint-training codecs, it fits that context to a finished artifact rather than co-adapting it during optimization. The resulting bitstream is roughly half the size of float16 or codebook storage at a fraction of their distortion. It decodes progressively, and the same codec runs unchanged on third-party checkpoints. When the training pipeline is available, the identical format admits an optional joint-training rung that roughly doubles the compression ratio again. This rung shares the codec, decoder, and bitstream structure with the post-hoc mode, differing only in the training procedure, so it serves as a controlled probe that isolates the net contribution of retraining under like-for-like conditions. Deployment thus becomes a climbable ladder rather than a forced trade-off.

In summary, we make three contributions.

- We introduce a flexible compression framework that unifies post-hoc and joint-training modes within a single bitstream format. Its primary mode compresses any trained anchor checkpoint without requiring training data, distillation recipes, or pipeline access.

- We design the core mechanisms that ensure training-to-deployment fidelity for both modes. These include a binary hash-grid entropy context with per-channel Laplace distributions, and three couplings that (i) calibrate rate loss and quantization on the baked feature–gate product, (ii) export the co-adapted context unchanged, and (iii) render through a hard straight-through anchor mask.
- We validate the framework on 13 scenes across three benchmarks, with ablations that delimit the design space. The post-hoc mode achieves 6.0× average compression with a -0.03 dB quality change, runs unchanged on third-party checkpoints, and supports progressive decoding; the joint-training rung roughly doubles the ratio again, reaching 12.9× at a -0.33 dB mean change.

2 Related Work

2.1 Anchor-Based 3D Gaussian Splatting

3D Gaussian Splatting [1] renders millions of 3D Gaussians in real time, but its memory footprint remains prohibitive. Later works improve it in many ways. Mip-Splatting [12] removes aliasing, 2DGS [13] improves surface geometry, Octree-GS [14] adds level-of-detail rendering through a multi-level octree, 3DGS-MCMC [15] replaces hand-designed densification with random sampling, and EAGLES [16] and Mini-Splatting [17] cut the number of primitives and reduce training cost. Closest to our work, Scaffold-GS [2] builds a sparse voxel grid of anchors. Each anchor stores a feature vector, offsets, and scalings, and lightweight MLPs decode them into opacity, color, and scale-rotation for each offset. The design uses fewer primitives and keeps fine details, but each anchor stores more data, so the total file size grows. The field keeps growing, with new works adding language features [18] and density control for moving scenes [19]. More trained models need compression, so we build on Scaffold-GS, distilling it into a smaller student and compressing the student without hurting its rendering quality.

2.2 Post-Hoc Compression

The simplest way to deploy a trained model is to convert its weights to float16, scalar grids, or codebooks [3–6, 20], with the same design appearing for neural fields [21]. These methods do not retrain the model and are safe and easy to use. But the weight patterns were fixed when training ended, so compression ratios hit a ceiling early and quality degrades quickly at low bit budgets. This line of work stays active. MesonGS [22] transforms attributes to lower their entropy. MesonGS++ [23] adds size-aware search over pruning ratios and bit-widths. POTR [24] prunes points and recomputes lighting without training. LightGaussian [4] prunes Gaussians first and then distills from the dense model to fix quality. Other works replace codebook quantization with dictionary learning [25] or rank primitives for pruning without rendering [26]. These methods share a common limit, treating the checkpoint as a static array and coding each attribute at what fixed rules assume, not at what its entropy actually costs. Feedforward encoders [7] target vanilla Gaussians and do not support

anchor-based models. Related learned compression targets other payloads, such as point clouds [27] and generative networks [28].

2.3 Joint Training

The second group embeds compression into the training loop. ContextGS [9] and the HAC line [10, 11] add quantization simulation, learned entropy contexts, and anchor masking to training, and rate-aware self-organizing grids [8] fall in the same group. They compress an order of magnitude deeper than post-hoc methods. But they require the full pipeline and a from-scratch retrain per scene, and the deployed model no longer matches the validated one. Recent works extend this line with HEMGS [29], which pairs a hybrid entropy model with hyperprior and autoregressive contexts, and LG-HCC [30], which exploits geometric correlations between anchors. Both keep their context models inside the training loop. These methods rely on tools from learned image compression, chiefly scale hyperpriors [31, 32] and joint autoregressive priors [33]. We use the same tools but place them at a different stage. Our post-hoc mode fits the context after training, and our joint mode keeps the same codec, changing only the training procedure. Rate pressure, anchor masking, and teacher distillation [34] now act together, a combination that prior work does not isolate.

Unlike the post-hoc conversions above, our primary mode learns an entropy context from each checkpoint, raising the post-hoc ceiling without retraining. The joint pipelines above make retraining mandatory, and ours makes it optional. Both modes share one decoder and one progressively decodable bitstream.

3 Method

Figure 1 depicts the full pipeline in three stages. First, a Scaffold-GS teacher is distilled into a gated student and baked. Any third-party anchor checkpoint can also enter directly. Second, two paths feed into the shared entropy codec. The post-hoc path takes any trained anchor checkpoint as it is, needing no training data, no distillation recipe, and no pipeline access. The joint path is available when the training pipeline is owned. It retrains the student with the codec in the loop under rate–distortion optimization and a hard straight-through anchor mask, and exports the co-adapted context into the same format. Third, both paths produce one Morton-ordered bitstream and share the entropy model, decoder, and progressive structure, differing only in what happens before the bitstream is produced.

3.1 Preliminaries

A Scaffold-GS scene has N anchors on a sparse voxel grid. Each anchor stores a feature vector $\boldsymbol{f}_i \in \mathbb{R}^F$, k grid offsets $\boldsymbol{o}_i \in \mathbb{R}^{k\times 3}$, and scalings $\boldsymbol{s}_i \in \mathbb{R}^6$. Three lightweight MLPs consume the feature together with the viewing direction and camera distance. They produce per-offset opacity, color, and scale-rotation. The neural Gaussians are splatted via tile-based rasterization following 3DGS [1]. See Lu et al. [2] for the full architecture.

Our student keeps the teacher's anchor positions. It trains all attributes under photometric and distillation losses, the latter against rendered teacher images. The

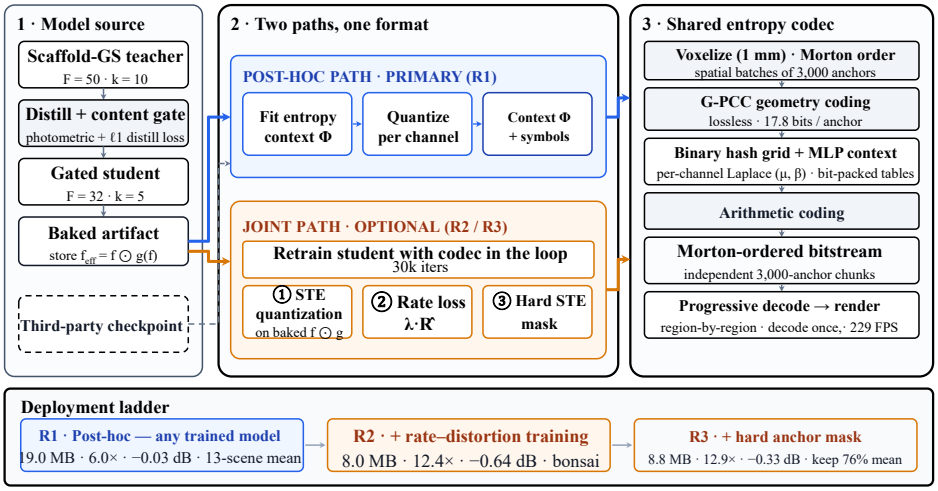


Fig. 1 The full pipeline in three stages. Left, the model source, where a Scaffold-GS teacher is distilled into a gated student and baked, and third-party checkpoints can also enter. Center, two paths over one format, the primary post-hoc path and the optional joint path with quantization simulation, rate loss, and a hard anchor mask. Right, the shared entropy codec emitting one Morton-ordered bitstream that decodes progressively. The bottom band lists the deployment ladder.

student is smaller than the teacher, using F=32 features and k=5 offsets per anchor against the teacher's F=50 and k=10.

A content-adaptive gate, inspired by channel attention [35], modulates the features before they enter the rendering MLPs. The gate is $\boldsymbol{g}(\boldsymbol{f}_i) = \sigma(\mathrm{MLP}(\boldsymbol{f}_i)) \in (0,1)^F$. It has two layers, from F to 32 hidden units and back, roughly two thousand parameters. The feature that the MLPs actually consume is

$$\boldsymbol{f}_i^{\,\mathrm{eff}} \;=\; \boldsymbol{f}_i \odot \boldsymbol{g}(\boldsymbol{f}_i). \tag{1}$$

The gate depends only on $\boldsymbol{f}_i$, so we can bake it at export, storing $\boldsymbol{f}^{\,\mathrm{eff}}$ and dropping the gate network. Rendering stays unchanged. The baked $\boldsymbol{f}^{\,\mathrm{eff}}$ is the only feature the codec sees. Its narrow dynamic range becomes important in Sec. 3.3.

3.2 Post-Hoc Entropy Coding

The post-hoc mode is the primary rung of our deployment ladder. It compresses any trained anchor model without retraining, treats the model as a read-only artifact, and fits a per-scene entropy context to its frozen attributes.

We are inspired by HAC++ [11]. Our codec follows the same design, with a hash-grid entropy context [36], a Laplace model, G-PCC geometry coding, and a Morton-ordered container. We review this shared machinery briefly. Anchors are voxelized at 1 mm resolution, sorted by Morton order, and coded with G-PCC, which yields about 17.8 bits per anchor on our scenes. Decoding returns every anchor to its quantized position, bit-exactly. Attributes are reordered into the same Morton sequence and

split into spatial batches of 3,000 anchors. The bitstream is a chain of independently decodable, spatially coherent chunks.

Within each batch, every attribute channel c is quantized on a uniform grid:

$$Q_c = \frac{x_c^{\max} - x_c^{\min}}{2^{b_c} - 1}, \tag{2}$$

where b_c is the bit budget, and $x_c^{\min}$, $x_c^{\max}$ bound the observed range of channel c in the model being coded. The post-hoc mode budgets nine bits for features and offsets, and ten bits for scalings.

The entropy model learns a per-channel Laplace distribution to minimize the code length of each quantized symbol. Symbols are arithmetic-coded [37] under this learned distribution. Its tails are heavier than Gaussian, matching the peaked, heavy-tailed residuals of the attributes.

A small MLP reads a binary hash-grid embedding of the anchor position. It outputs per-channel location μ_c and scale β_c. The code length of symbol x in its quantization bin is

$$R(x) = -\log_2\big(\mathcal{F}(x + \tfrac{Q_c}{2}) - \mathcal{F}(x - \tfrac{Q_c}{2})\big), \tag{3}$$

where $\mathcal{F}$ is the Laplace CDF. Decoding is geometry-first. G-PCC recovers the anchor positions. These positions then drive the hash grid. The entropy context needs no side information beyond the bit-packed tables. Because batches follow Morton order, a decoder can render each region as its chunk arrives. We use this for progressive decoding in Sec. 4.2.1.

The hash-grid tables are binarized to ± 1 via straight-through estimation, as in binary-weight networks [38]. They are stored bit-packed, which keeps the side information at kilobytes. Storing them in float16 would cost more than the coding savings they provide.

Given a trained anchor model, the post-hoc mode fits the entropy context directly to its frozen attributes. The schema is the only interface, and no training images, distillation recipes, or pipeline access are needed. Fitting minimizes the average code length of the model's own attributes:

$$\min_{\Phi} \frac{1}{NC} \sum_{i=1}^{N} \sum_{c=1}^{C} R_\Phi(x_{ic}), \tag{4}$$

where Φ holds the parameters of the binary hash grid and the distribution MLP, C is the number of attribute channels, and R_Φ is the code length of Eq. (3) on the frozen attribute symbol x_{ic} of anchor i in channel c. After fitting, quantization and arithmetic coding follow as described above.

3.3 Optional Joint-Training Extensions

The joint mode asks how much deeper compression can go when the model is trained with full knowledge of the codec. Three couplings align the training conditions with the deployment conditions. We add two higher rungs, with R2 retraining the model under rate–distortion loss and R3 adding a hard anchor mask on top.

3.3.1 Quantization simulation on the baked quantity.

The first coupling adds quantization noise during training, making the student robust to the grids it will face at deployment. Offsets and scalings use straight-through rounding [39, 40]:

$$\tilde{x} = x + \big(\mathrm{round}(x/Q_c)\cdot Q_c - x\big)^{\text{stop-grad}}, \tag{5}$$

applied from iteration 3,000 onward. For features, the codec sees the baked product $\boldsymbol{f}_i^{\text{eff}}$ from Eq. (1), so the quantization grids operate on this product, whose range is roughly 18× narrower than the raw feature's. Grids refresh every 2,000 iterations.

3.3.2 Rate loss and the co-adapted context.

The second coupling adds the code length to the training loss, pulling the model toward parameters that are cheap to encode. From iteration 10,000, the total loss $\mathcal{L}$ becomes

$$\mathcal{L} = \mathcal{L}_{\text{render}} + \alpha\,\mathcal{L}_{\text{distill}} + \lambda\,\widehat{\boldsymbol{R}}, \tag{6}$$

where the rendering term is the standard 3DGS photometric loss on the student render $\hat{I}$ and the ground-truth view I, an ℓ_1 term plus a D-SSIM term,

$$\mathcal{L}_{\text{render}} = (1-\gamma)\,\|\hat{I} - I\|_1 + \gamma\,\big(1 - \mathrm{SSIM}(\hat{I}, I)\big), \tag{7}$$

the distillation term is the ℓ_1 gap between the student render and the teacher render $I_{\mathcal{T}}$,

$$\mathcal{L}_{\text{distill}} = \|\hat{I} - I_{\mathcal{T}}\|_1, \tag{8}$$

and $\widehat{\boldsymbol{R}}$ is the mean code length of Eq. (3) over a 5% uniform anchor sample. The weight α scales the distillation term, γ the D-SSIM term, and λ the rate term, trading distortion for rate. The weights γ=0.2 and α=0.7 are fixed across all scenes, following the shared teacher–student recipe. Gradients of $\widehat{\boldsymbol{R}}$ pull the parameters toward entropy-cheap bins while the context co-adapts and is exported unchanged into the bitstream. The training-time estimate of 1.95 bits per symbol for features is met by the actual encoder at 1.948, a fidelity that a post-hoc re-fit cannot reach.

3.3.3 Hard straight-through anchor mask.

The third coupling prunes anchors during training, which compresses the point set itself. Each anchor i gets a logit θ_i and a gate $a_i = \sigma(\theta_i)$ that scales its neural opacity. A hinge penalty $\lambda_m \max(\bar{a} - \rho, 0)$ pushes the keep ratio toward a target ρ. The rendering loss removes anchors whose deletion does not hurt quality, in the spirit of learned sparsification [41].

The forward pass renders the thresholded gate through straight-through estimation:

$$\hat{a}_i = a_i + \big(\mathbb{1}[a_i > 0.5] - a_i\big)^{\text{stop-grad}}, \tag{9}$$

where $\mathbb{1}[\cdot]$ is the indicator function, 1 if the condition holds and 0 otherwise. The model rendered during training is the model that survives at export. Kept anchors train at full opacity, removed ones contribute nothing, and export reduces to a row filter.

We use $\lambda_m = 0.02$ and a keep target of $\rho = 0.75$ indoors and 0.85 outdoors. A soft gate creates a train–deploy gap, and the hard gate closes it.

3.4 The Deployment Ladder

The two modes define three deployment rungs over one shared bitstream format.

R1. Post-hoc coding. Any trained model is compressed without retraining, as in Sec. 3.2.

R2. Joint rate–distortion training. The first two couplings of Sec. 3.3, applied during training.

R3. Joint training with the hard mask. R2 plus the third coupling. Training renders the pruned model, and export is a threshold.

The rung is selected per scene by the rate–distortion trade-off. Inside R2 and R3, λ controls the rate weight and ρ controls the keep ratio.

4 Experiments

4.1 Setup

Dataset. We evaluate on 13 scenes from the three benchmarks commonly used in this field, nine from Mip-NeRF 360 [42, 43], two from Tanks and Temples (T&T) [44], and two from Deep Blending (DB) [45].

Implementation details. All student models are distilled from a Scaffold-GS teacher under an identical recipe of 30,000 iterations, a voxel size of 5 mm, F=32 features, k=5 offsets per anchor, and the content-adaptive gate of Sec. 3.1. Post-hoc context fitting runs 8,000 steps per scene; joint training runs 30,000 iterations with the codec in the loop (Sec. 3.3). All experiments use one NVIDIA RTX 4090D, and evaluation renders test views at 1600 px.

Evaluation protocol. Quality is PSNR, SSIM, and LPIPS on the official test splits. We render after full decode with the unmodified evaluation pipeline. Bitstream size counts every byte the decoder needs. This includes geometry, attribute codes, and coding context. All sizes are binary megabytes reported as MB. Ratios are against the float32 anchor PLY of the same model.

Baselines. We group baselines by whether the scene is retrained. The Train column of Table 1 marks the level. *None* codes a frozen model, *scratch* trains the scene from scratch and ships an uncompressed or pruned model, *pipeline* retrains within an existing pipeline such as finetuned post-training variants and our joint rung, and *joint* embeds the codec in optimization as HAC++ does. Uncompressed references ship raw PLY files, vanilla 3DGS [1], Scaffold-GS [2], our teacher, and Octree-GS [14]. We train and measure these ourselves. Post-hoc methods code a frozen model, the float16 and scalar 9-bit conversions deployed on our students, together with the no-finetuning rows of MesonGS [22] and POTR [24], transcribed from their papers. Methods that touch training either prune and distill at training time, like EAGLES [16] and LightGaussian [4], or embed compression in optimization, like HAC++ [11] and our joint rungs. We reproduce EAGLES, LightGaussian, and HAC++ on our own hardware, and the finetuned post-training variants, POTR-FT and MesonGS++ [23], belong

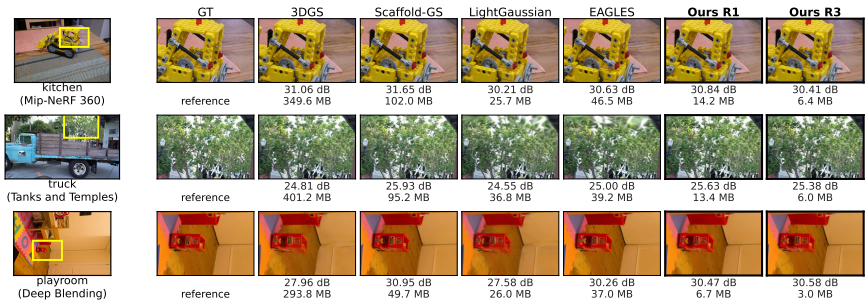


Fig. 2 Qualitative comparison on one scene per dataset, kitchen from Mip-NeRF 360, truck from T&T, and playroom from Deep Blending. For each scene the left frame is the ground truth with the crop region marked; the crops, enlarged from the rendered pixels of four baselines and both of our rungs at the operating points of Table 1, are annotated with scene PSNR and deployable size in MB. Scaffold-GS is the uncompressed teacher; Ours shows the decoded bitstream itself.

here as well. HAC++ is the strongest published codec of the joint-training line and represents that regime in Table 1.

4.2 Main Results

Table 1 compares our two rungs against the baselines. The post-hoc rung compresses any trained model without retraining.

Against float16 and scalar quantization, the bitstream is roughly half the size at a fraction of their distortion. Fixed grids and codebooks use the same rules for every scene and ignore the actual attribute distributions. Our entropy context learns these distributions and adapts the coding to the data.

Except for the joint-trained HAC++, the compression baselines are non-joint methods. None of them learns the actual entropy of the scene, but our codec does. The post-hoc rung delivers higher PSNR at a smaller size than the non-joint methods except POTR-FT, which is smaller but consistently lower in PSNR.

The uncompressed baselines show the starting point, and our bitstreams are much smaller than all of them. The quality gap comes mainly from the distilled student, half a decibel below the teacher, while the compression itself costs hundredths. HAC++ compresses deeper by training the model and codec together from scratch, which requires full retraining. Our post-hoc mode fits a frozen checkpoint without pipeline access, and the joint rung roughly doubles the compression ratio again when retraining is possible.

Figure 2 shows crops from one scene per dataset. The post-hoc decode is close to the teacher, and the joint rung shows slightly more blur. Figure 3 replots the Mip-NeRF 360 means in the rate–distortion plane. The post-hoc marker sits left of every uncompressed baseline at comparable quality, and the joint rung shifts roughly another factor of two left. The inset of Figure 3 previews the bit-depth ablation on bonsai, where the joint rung at 6 bits beats the post-hoc rung at the same depth by +0.52 dB, as Sec. 4.3 details.

Table 1 Comparison with existing methods on the three evaluation datasets, group means. Size is the mean deployable artifact in MB, the full bitstream for ours and HAC++, and raw PLY or compressed containers for the rest. A dagger marks rows transcribed from published results under the 3DGS.zip protocol [46] on vanilla 3DGS at full resolution, which are not directly comparable with the self-measured rows.

		Mip-NeRF 360				T&T				Deep Blending			
Method	Train	PSNR	SSIM	LPIPS	MB	PSNR	SSIM	LPIPS	MB	PSNR	SSIM	LPIPS	MB
3DGS [1]	scratch	27.13	0.8040	0.2409	647.1	23.14	0.8273	0.2135	372.0	27.04	0.8765	0.2909	448.0
Octree-GS [14]	scratch	27.59	0.8131	0.2226	122.3	24.49	0.8644	0.1554	78.0	30.26	0.9090	0.2489	54.9
Scaffold-GS (teacher) [2]	scratch	27.73	0.8134	0.2265	173.5	24.33	0.8553	0.1724	77.5	30.33	0.9096	0.2525	54.9
HAC++ [11]	joint	27.54	0.8054	0.2475	8.9	24.23	0.8495	0.1891	5.3	30.18	0.9073	0.2662	2.9
EAGLES [16]	scratch	27.16	0.8095	0.2337	58.3	23.29	0.8377	0.2014	29.0	29.81	0.9101	0.2462	52.5
LightGaussian [4]	scratch	26.68	0.7929	0.2578	42.2	22.85	0.8138	0.2326	24.5	26.63	0.8669	0.3105	29.2
POTR† [24]	none	27.08	0.806	0.226	26.0	23.27	0.834	0.191	11.3	29.31	0.897	0.253	16.5
POTR-FT† [24]	pipeline	27.20	0.808	0.223	21.3	23.34	0.837	0.189	9.4	29.44	0.902	0.250	11.8
MesonGS† [22]	none	26.22	0.785	0.249	29.7	22.84	0.820	0.211	17.3	28.70	0.890	0.271	29.0
MesonGS++† [23]	pipeline	26.85	0.797	0.286	36.1	23.25	0.832	0.231	17.8	29.59	0.902	0.316	31.2
MesonGS++ HR† [23]	pipeline	27.06	0.803	0.277	65.4	23.37	0.839	0.222	31.1	29.65	0.903	0.312	55.5
Student (baked)	—	27.26	0.7994	0.2523	140.5	23.89	0.8408	0.2004	62.8	29.90	0.9056	0.2645	44.5
+ float16	none	27.20	0.7974	0.2533	52.1	23.87	0.8400	0.2007	23.5	29.87	0.9048	0.2651	16.5
+ scalar 9-bit	none	27.10	0.7921	0.2554	42.1	23.43	0.8157	0.2117	19.0	29.86	0.9048	0.2651	14.0
Ours, post-hoc (R1)	none	27.23	0.7981	0.2531	23.3	23.88	0.8399	0.2009	11.2	29.88	0.9052	0.2649	7.5
Ours, joint (R3)	pipeline	26.90	0.7861	0.2682	10.8	23.47	0.8264	0.2177	5.2	29.83	0.9032	0.2715	3.4

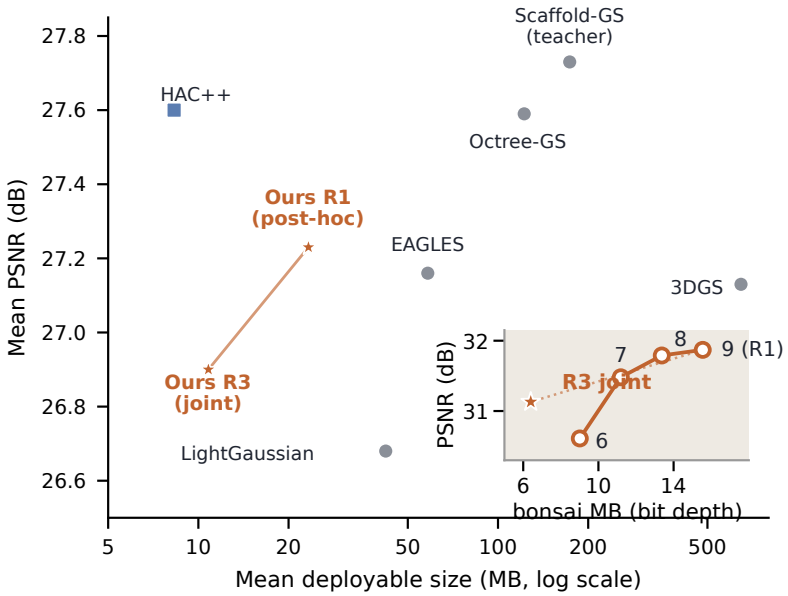


Fig. 3 Rate–distortion view of the Mip-NeRF 360 means of Table 1. Each marker is a group mean over the nine scenes, our two rungs are stars joined by the line that traces the move from R1 to R3, and the size axis is logarithmic. The inset sweeps the bonsai bit depth from 9 to 6 against the 6-bit joint rung, Table 5.

Table 2 Post-hoc coding of four checkpoints we did not train. SGS = full-size native Scaffold-GS (F=50, k=10); Δ is the paired PSNR change of decoded against original checkpoint, both rendered by our stack; MB is the decoded bitstream and the ratio is against the checkpoint's float32 anchor PLY.

Checkpoint	PLY MB	MB	ratio	PSNR	Δ dB
SGS bonsai	128.8	21.9	5.9×	32.37	−0.015
SGS bicycle	309.1	52.6	5.9×	25.11	−0.018
Octree bonsai	55.0	7.5	7.3×	30.15	−0.073
Octree bicycle	240.1	37.2	6.5×	24.86	−0.089
mean	183.3	29.8	6.4×	—	−0.049

4.2.1 Post-Hoc Analysis

The post-hoc rung adapts to each scene individually, with per-channel grids tuned to the scene's dynamic range. Scalar quantization applies a uniform rule across all channels, while entropy coding preserves reconstruction quality. Figure 4 confirms this in pixels. The post-hoc decode is indistinguishable from the baked student, above 40 dB and 0.99 SSIM per crop, while the joint rung loses some high-frequency detail.

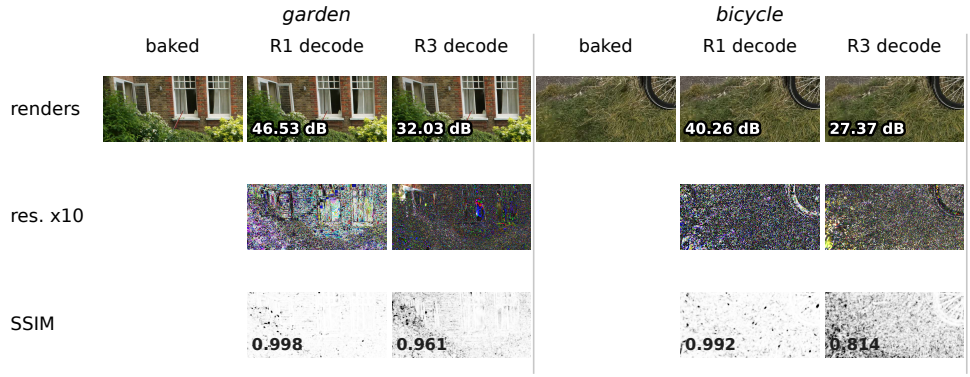


Fig. 4 Codec transparency on garden and bicycle; each lower panel sits under its source render. Top row: the baked student and the two decodes, with per-crop PSNR against it. Middle row: residuals of the two decodes against that render, amplified 10× and clipped, brighter meaning more error. Bottom row: SSIM maps against the same render, brighter meaning more similar, each contrast-stretched to its own range.

Table 3 Anatomy of the post-hoc bitstream on bonsai, 15.5 MB total.

Component	MB	share
anchor geometry (G-PCC, Morton order)	0.83	5.3%
feature codes (32×9 bit, context-coded)	9.10	58.6%
offset codes (15×9 bit)	3.03	19.5%
scaling codes (6×10 bit)	2.42	15.6%
coding context (bit-packed) + metadata	0.17	1.1%

The same codec runs unchanged on third-party checkpoints, and we tested four we did not train, two native Scaffold-GS and two Octree-GS. Their compression ratios and quality match our pipeline, requiring no adaptation, as Table 2 shows. As Table 3 indicates, the majority of the bitstream is consumed by features, whereas side information is negligible.

The bitstream supports progressive decoding, with chunks delivered in spatial order, so a client renders each region as soon as it arrives. On bonsai and bicycle, a quarter of the stream already produces a recognizable image, and quality improves as more chunks arrive, as Fig. 5 traces. Concurrent work builds progressive bitstreams for vanilla 3DGS [47]. In contrast, fixed baselines require a full download before any rendering can begin.

4.2.2 Joint Rung Analysis

The joint rungs build on R1, with R2 adding rate–distortion training and R3 the hard anchor mask, both over the same codec and bitstream format. Table 4 walks the ladder on bonsai.

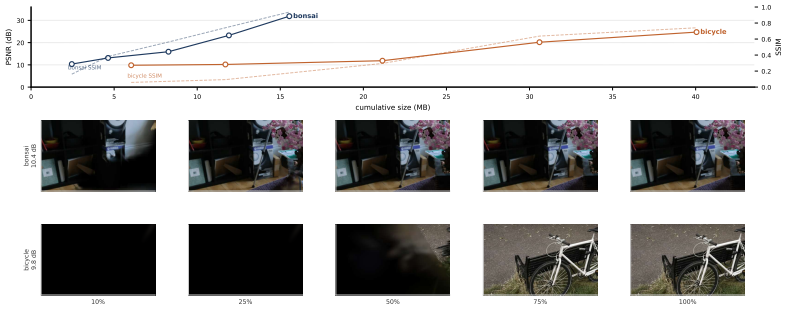


Fig. 5 Progressive decoding on bonsai and bicycle. The top panel traces whole-view PSNR, solid curves, and SSIM, dashed, against cumulative received size, counting the geometry and context header up front and the attribute stream after it; the strips below show the same crop at the five marked received fractions, bonsai above bicycle. Each arriving Morton batch refines one spatial region rather than the whole image, so quality climbs with received bytes instead of jumping only at full download.

Table 4 Deployment ladder on bonsai, float32 PLY 99.5 MB and 31.92 dB; keep is the fraction of anchors retained.

Rung	MB	ratio	PSNR	Δ	keep
baked (no codec)	99.5	1.0×	31.92	—	100%
R1 post-hoc	15.5	6.4×	31.87	−0.04	100%
R2 joint, λ=0.002	8.0	12.4×	31.27	−0.64	100%
R2 joint, λ=0.005	6.9	14.4×	31.03	−0.89	100%
R3 joint + mask	**6.4**	**15.5**×	**31.13**	−0.79	68%
13-scene means: R1 19.0 *MB,* 6.0×*,* −0.03 *dB; R3* 8.8 *MB,* 12.9×*,* −0.33 *dB, keep 76%*					

R2 isolates rate pressure alone, roughly doubling compression on bonsai at a modest quality cost.

R3 adds the hard mask across the full scene set. Indoor scenes target 75 percent of anchors, outdoor scenes 85 percent, since weak position priors make pruning costlier outside. No scene loses more than 0.8 dB, and two scenes even land above baseline as the mask removes anchors that contributed noise. Ablating the distillation term on five scenes costs 0.14 to 0.41 dB on the three scenes that clear the seed noise, counter and garden stay within it, and the no-KD models settle 4–7% smaller, so the teacher premium buys quality rather than rate.

The hard mask trains what it deploys, with kept anchors at full opacity and removed ones contributing nothing. A soft mask opens a train–deploy gap, and the hard mask closes it. Deleted anchors also speed up rendering, as Sec. 4.4 details.

These rungs require the training pipeline and offer a deeper optional path, and the same decoder handles both modes.

Table 5 Post-hoc quality vs. attribute bit budget on bonsai, with features and offsets coded at b bits and scalings at $b{+}1$ bits for $b \in \{9, 8, 7, 6\}$.

budget	MB	PSNR	Δ
9 bit, post-hoc	15.5	31.87	−0.04
8 bit, post-hoc	13.4	31.79	−0.13
7 bit, post-hoc	11.2	31.48	−0.44
6 bit, post-hoc	9.0	30.61	−1.31
6 bit, joint (R3)	6.4	**31.13**	−0.79

4.3 Ablations and Analysis

Ablations run on bonsai unless stated otherwise, reading the ladder of Table 4 and the bit budget of Table 5. The section first sweeps the post-hoc design space, then ablates the three joint couplings.

Bit budget and context value. Table 5 sweeps the attribute bit budget of the post-hoc rung. Each dropped bit frees 2.1 to 2.2 MB, but the quality cost accelerates from 0.08 dB for the first step to 0.87 dB for the third. Nine bits is the last cheap bit, and the rung ships there. The same grids also isolate the context's value. On identical 9-bit grids, scalar coding occupies 29 MB at 31.83 dB and context coding 15.5 MB at 31.87 dB. The quantizer is shared, so the 13.5 MB saved at no quality cost is the entropy model alone.

The post-hoc limit. Below 7 bits the frozen symbol statistics become the bottleneck, and a refit cannot add information the symbols no longer carry. The joint rung of Sec. 4.2.2 is built for this end, and trained against the codec it holds 31.13 dB at 6 bits where the post-hoc rung falls to 30.61 dB. The +0.52 dB gap prices what retraining buys once the artifact itself may move.

Ablation of the joint couplings. Table 6 ablates the couplings of Sec. 3.3 on bonsai. Removing the rate loss drops the recipe to 10.5× at −0.75 dB, so the term buys rate pressure, not generic regularization. Both soft-gate exports trail the hard-STE mask on both axes, and score pruning, free at 10% removed, loses another 1.28 dB at a quarter, so pruning must be trained in. Grids from raw features, before gating, make the simulated noise 18× coarser, and our first joint run thereby read a flattering 1.11 bits per symbol while the encoder paid far more, so all grids, STE operations, and rate estimates operate on the baked product $\boldsymbol{f} \odot \boldsymbol{g}$.

4.4 Time Cost

Table 7 lists the wall-clock cost per scene on one RTX 4090D. The post-hoc rung needs only the model file and about nine minutes, which parallelizes across scenes. HAC++ trains from scratch in 28.4 minutes and additionally requires the images, so for third-party checkpoints it has no path at all. Our joint rung spans 16 to 45 minutes with the teacher in the loop, and with the distillation term disabled it finishes on par

Table 6 Ablation of the joint couplings on bonsai, extending Table 4. Δ is the PSNR change against the baked model at 31.92 dB, and keep is the fraction of anchors retained. Score pruning removes anchors by accumulated opacity after training, applied here to the R2 model.

Configuration	MB	ratio	PSNR	Δ	keep
rate loss					
R2 with rate loss	8.0	12.4×	31.27	−0.64	100%
R2 without rate loss	9.46	10.5×	31.17	−0.75	100%
anchor mask					
R3 hard STE	6.4	15.5×	31.13	−0.79	68%
soft gate, default threshold	7.05	14.1×	31.01	−0.91	74%
soft gate, harder threshold	6.80	14.6×	30.95	−0.97	70%
pruning after training					
score pruning, 10% removed	7.73	12.9×	31.27	−0.65	90%
score pruning, 25% removed	7.05	14.1×	30.00	−1.92	75%

Table 7 Wall-clock cost per scene, all on one RTX 4090D. HAC++ timing is from its official-code reproduction.

Pipeline	time / scene
compressing a frozen model	
post-hoc rung (context fit + encode)	∼9 min
decoding the bitstream (once)	5–50 s
training with the codec	
HAC++ (train from scratch)	28.4 min
distillation only	∼19 min
joint rung (distill + RD + mask)	16–45 min

with HAC++, so the entire premium is the teacher's supervision, priced against the +0.14 to +0.41 dB it buys in Sec. 4.2.2. Where a teacher checkpoint already exists, the student-side cost matches a from-scratch HAC++ run. Rendering itself carries no codec overhead, with the quantization already priced into the −0.03 dB, and the mask rung raises frame rate by 18 to 43 percent on the scenes measured.

5 Conclusion

We have presented a flexible compression framework for anchor-based Gaussian splatting whose primary rung is strictly post hoc and whose joint rung is optional. Across 13 scenes and three benchmarks, the post-hoc rung roughly halves the size of float16

and codebook storage at a fraction of their distortion, and the joint rung roughly doubles the compression ratio again at a modest quality cost. Every rung ships in a single format, and the same decoder serves both modes.

Two principles survive the ablations. Fit the codec to the artifact, not the artifact to the codec, because anchor statistics survive training. Train what you ship. Most of the gains of HAC and HAC++ [10, 11] do not require retraining, though the deepest ratios remain theirs. The plateau of fixed conversions [3, 22, 24] is a choice of coding statistics, not a property of the representation. Supervision and compression complement each other, since the teacher compensates where rate pressure and masking weaken the photometric signal.

limitations. The post-hoc rung is bounded by the redundancy the model's training left behind, and the joint rung that lifts this bound requires retraining from scratch. Our quantization uses one grid per channel, and per-anchor adaptive steps as in HAC++ [11] and offset-level masks are natural next steps.

Declarations

Funding. This work was supported by the Liaoning Provincial Science and Technology Plan Joint Program (Natural Science Foundation - Doctoral Scientific Research Initiation Project, under Grant 2024-BSLH-004) and the Dalian University Research Startup Project (Grant No. 2025-XRCZK-01).

Conflict of interest. The authors have no relevant financial or non-financial interests to disclose.

Data availability. The evaluation datasets are publicly available, with Mip-NeRF 360 at https://jonbarron.info/mipnerf360/, Tanks and Temples at https://www.tanksandtemples.org/, and Deep Blending at https://github.com/Phog/DeepBlending.

Author contributions. All authors contributed to this research. Chi Li designed the research, processed the experimental data, prepared figures, and wrote the first draft of the manuscript. Shixi Wen directed the experiments and data analysis, and revised and edited the manuscript. Yuan Zhao collaborated on data processing and revised and edited the manuscript. Jian Liu collaborated on figure preparation and revised and edited the manuscript. All authors read and approved the final manuscript.

Use of large language models. During the preparation of this work the authors used ZCode and Kimi in order to assist with polishing the manuscript, experimentation code, and figure preparation. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

[1] Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G., *et al.*: 3d gaussian splatting for real-time radiance field rendering. ACM Trans. Graph. **42**(4), 139–1 (2023)

[2] Lu, T., Yu, M., Xu, L., Xiangli, Y., Wang, L., Lin, D., Dai, B.: Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 20654–20664 (2024). IEEE

[3] Lee, J.C., Rho, D., Sun, X., Ko, J.H., Park, E.: Compact 3d gaussian representation for radiance field. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 21719–21728 (2024). IEEE

[4] Fan, Z., Wang, K., Wen, K., Zhu, Z., Xu, D., Wang, Z.: Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, vol. 37, pp. 140138–140158 (2024)

[5] Niedermayr, S., Stumpfegger, J., Westermann, R.: Compressed 3d gaussian splatting for accelerated novel view synthesis. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10349–10358 (2024). IEEE

[6] Navaneet, K., Pourahmadi Meibodi, K., Abbasi Koohpayegani, S., Pirsiavash, H.: CompGS: Smaller and faster gaussian splatting with vector quantization. In: European Conference on Computer Vision, pp. 330–349 (2024). Springer

[7] Chen, Y., Wu, Q., Li, M., Lin, W., Harandi, M., Cai, J.: Fast feedforward 3d gaussian splatting compression. In: International Conference on Learning Representations, vol. 2025, pp. 74859–74872 (2025)

[8] Morgenstern, W., Barthel, F., Hilsmann, A., Eisert, P.: Compact 3d scene representation via self-organizing gaussian grids. In: European Conference on Computer Vision, pp. 18–34 (2024). Springer

[9] Wang, Y., Li, Z., Guo, L., Yang, W., Kot, A.C., Wen, B.: ContextGS: Compact 3D gaussian splatting with anchor level context model. Advances in neural information processing systems **37**, 51532–51551 (2024)

[10] Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: Hac: Hash-grid assisted context for 3d gaussian splatting compression. In: European Conference on Computer Vision, pp. 422–438 (2024). Springer

[11] Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: Hac++: Towards 100x compression of 3d gaussian splatting. IEEE Transactions on Pattern Analysis and Machine Intelligence (2025)

[12] Yu, Z., Chen, A., Huang, B., Sattler, T., Geiger, A.: Mip-splatting: Alias-free 3d gaussian splatting. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 19447–19456 (2024). IEEE

[13] Huang, B., Yu, Z., Chen, A., Geiger, A., Gao, S.: 2D gaussian splatting for geometrically accurate radiance fields. In: ACM SIGGRAPH 2024 Conference Papers, pp. 1–11 (2024)

[14] Ren, K., Jiang, L., Lu, T., Yu, M., Xu, L., Ni, Z., Dai, B.: Octree-GS: Towards consistent real-time rendering with LOD-structured 3D gaussians. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1–15 (2025)

[15] Kheradmand, S., Rebain, D., Sharma, G., Sun, W., Tseng, Y.-C., Isack, H., Kar, A., Tagliasacchi, A., Yi, K.M.: 3d gaussian splatting as markov chain monte carlo, vol. 37, pp. 80965–80986 (2024)

[16] Girish, S., Gupta, K., Shrivastava, A.: Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In: European Conference on Computer Vision, pp. 54–71 (2024). Springer

[17] Fang, G., Wang, B.: Mini-splatting: Representing scenes with a constrained number of gaussians. In: European Conference on Computer Vision, pp. 165–181 (2024). Springer

[18] Xu, R., Ji, Y., Tan, X., Ma, L.: Osh-splat: optimizable semantic hyperplanes for enhanced 3d language feature gaussian splatting. The Visual Computer **41**(13), 11127–11137 (2025)

[19] Cao, M., Liu, Z., Li, N., Zhao, H.: Spatiogs: spatiotemporal-aware density control for dynamic scene rendering with gaussian splatting. The Visual Computer **42**(5), 215 (2026)

[20] Liu, X., Wu, X., Zhang, P., Wang, S., Li, Z., Kwong, S.: CompGS: Efficient 3D scene representation via compressed gaussian splatting. In: Proceedings of the 32nd ACM International Conference on Multimedia, pp. 2936–2944 (2024)

[21] Chen, Y., Wu, Q., Harandi, M., Cai, J.: How far can we compress instant-ngp-based nerf? In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 20321–20330 (2024). IEEE

[22] Xie, S., Zhang, W., Tang, C., Bai, Y., Lu, R., Ge, S., Wang, Z.: MesonGS: Post-training compression of 3D gaussians via efficient attribute transformation. In: European Conference on Computer Vision (2024)

[23] Xie, S., Ge, J., Zhang, W., Liu, J., Tang, C., Bai, Y., Ge, S., Jiang, J., Huang, Y., Yang, F., Zhang, C., Fan, X., Wang, Z.: MesonGS++: Post-training compression of 3D gaussian splatting with hyperparameter searching (2026) arXiv:2604.26799

[24] Ramlot, B., Courteaux, M., Lambert, P., Wallendael, G.V.: Potr: Post-training 3dgs compression. IEEE Transactions on Circuits and Systems for Video Technology **36**(8), 12168–12181 (2026)

[25] Gong, J., Unger, J., Miandji, E.: Smaller and faster 3DGS via post-training dictionary learning. arXiv preprint arXiv:2605.30396 (2026)

[26] Chen, Z., Wan, S., Yu, M., Yang, F., Hou, J.: REFINE: Super-efficient 3D gaussian splatting pruning via rendering-free primitive importance. arXiv preprint arXiv:2606.09074 (2026)

[27] Tychola, K.A., Vrochidou, E., Papakostas, G.A.: Deep learning based computer vision under the prism of 3d point clouds: a systematic review. The Visual Computer **40**(11), 8287–8329 (2024)

[28] Tian, Y., Shen, L., Tian, X., Tao, D., Li, Z., Liu, W., Chen, Y.: Dgl-gan: discriminator-guided gan compression: Y. tian et al. The Visual Computer **41**(7), 4639–4660 (2025)

[29] Liu, L., Chen, Z., Jiang, W., Wang, W., Xu, D.: HEMGS: A hybrid entropy model for 3D gaussian splatting data compression. arXiv preprint arXiv:2411.18473 (2024)

[30] Deng, X., Meng, X., Man, H., Zhu, Q., Zhang, T., Zhao, D., Fan, X.: LG-HCC: Local geometry-aware hierarchical context compression for 3D gaussian splatting. arXiv preprint arXiv:2603.28431 (2026)

[31] Ballé, J., Laparra, V., Simoncelli, E.P.: End-to-end optimized image compression. arXiv preprint arXiv:1611.01704 (2016)

[32] Ballé, J., Minnen, D., Singh, S., Hwang, S.J., Johnston, N.: Variational image compression with a scale hyperprior. arXiv preprint arXiv:1802.01436 (2018)

[33] Minnen, D., Ballé, J., Toderici, G.D.: Joint autoregressive and hierarchical priors for learned image compression, vol. 31 (2018)

[34] Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)

[35] Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7132–7141 (2018)

[36] Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM transactions on graphics (TOG) **41**(4), 1–15 (2022)

[37] Witten, I.H., Neal, R.M., Cleary, J.G.: Arithmetic coding for data compression. Communications of the ACM **30**(6), 520–540 (1987)

[38] Courbariaux, M., Bengio, Y., David, J.-P.: Binaryconnect: Training deep neural networks with binary weights during propagations, vol. 28 (2015)

[39] Bengio, Y., Léonard, N., Courville, A.: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. arXiv preprint arXiv:1308.3432 (2013)

[40] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2704–2713 (2018). IEEE

[41] Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network, vol. 28 (2015)

[42] Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5460–5469 (2022). IEEE

[43] Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM **65**(1), 99–106 (2021)

[44] Knapitsch, A., Park, J., Zhou, Q.-Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction, vol. 36, pp. 1–13. ACM New York, NY, USA, ??? (2017)

[45] Hedman, P., Philip, J., Price, T., Frahm, J.-M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering. ACM Transactions on Graphics (ToG) **37**(6), 1–15 (2018)

[46] Bagdasarian, M.T., Knoll, P., Li, Y., Barthel, F., Hilsmann, A., Eisert, P., Morgenstern, W.: 3dgs. zip: A survey on 3d gaussian splatting compression methods. Computer Graphics Forum **44**(2), 70078 (2025)

[47] Chen, Y., Li, M., Wu, Q., Lin, W., Harandi, M., Cai, J.: Pcgs: Progressive compression of 3d gaussian splatting. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 40, pp. 3111–3119 (2026)