TOWARDS PRACTICAL COMPRESSION OF 3D GAUSSIAN SPLATTING

Anonymous authors
Paper under double-blind review

ABSTRACT

3D Gaussian Splatting (3DGS) enables high-quality novel-view synthesis but requires substantial storage. Existing compression methods often rely on spatial context modeling over irregular 3D representations, increasing the complexity of training and coding. Meanwhile, floating-point context inference can introduce numerical inconsistencies across platforms, causing entropy-decoding failures. To address these practical challenges, we propose COSA-GS, which constructs **C**ontext with**O**ut **S**patial **A**ggregation through anchor-wise causal factorization. Specifically, we use geometry context derived from each anchor's coordinates to model a compact learnable anchor latent. The anchor latent is then fused with the geometry context to form an anchor context for attribute coding. The resulting context model features a simple architecture composed solely of linear transformations and activations. We train COSA-GS using rate–distortion optimization with adaptive Gaussian pruning. Further, we develop quantization-aware training and integer inference for the context model to achieve bit-exact consistency of entropy-decoded symbols across platforms. Experiments demonstrate that COSA-GS achieves state-of-the-art compression performance while retaining fast and consistent cross-platform decoding, providing a simple yet effective framework for practical 3DGS compression. Anonymized code and a demo are provided in the supplementary materials and at `https://anonymous.4open.science/r/COSA-GS`.

1 INTRODUCTION

3D Gaussian Splatting (3DGS) enables high-quality novel-view synthesis (Kerbl et al., 2023). However, its explicit scene representation often contains numerous Gaussian primitives with associated appearance and geometry parameters, resulting in substantial storage and transmission costs. To reduce this overhead, existing methods explore pruning, vector quantization, compact parameterizations, and entropy coding (Bagdasarian et al., 2025).

An important direction is to exploit spatial correlations in the scene representation. Scaffold-GS (Lu et al., 2024) parameterizes local Gaussians using anchors and learnable offsets, and predicts their view-dependent attributes from anchor features and viewing conditions, providing a compact representation for subsequent compression methods. Building on this representation, HAC and HAC++ query a learned hash grid at anchor locations to construct spatial context for attribute distribution modeling (Chen et al., 2024; 2025). ContextGS instead organizes anchors into hierarchical levels and uses already decoded coarser anchors, together with a per-anchor hyperprior, to predict finer-level attribute distributions (Wang et al., 2024b).

Although these methods improve compression performance, their reliance on spatial context extraction over irregular 3D structures complicates the coding pipeline and limits coding speed. Meanwhile, floating-point context inference can introduce numerical inconsistencies across platforms, causing entropy-decoding failures (Ballé et al., 2019; Koyuncu et al., 2022). This raises a practical question: *can we achieve strong rate–distortion performance and fast, consistent cross-platform decoding with a simple architecture?* Our intuition is that 3DGS optimization itself can reduce inter-anchor dependence when guided by an appropriate context model. Concretely, we can encourage each anchor to learn distinctive spatial information while concentrating the remaining dependencies within individual anchors, allowing complex spatial context aggregation to be replaced by simpler

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.

anchor-wise modeling. Note that *anchor-wise* does not mean discarding spatial information: the context for each anchor can still be derived from its own coordinate.

Based on this principle, we propose COSA-GS, short for **C**ontext with**o**ut **S**patial **A**ggregation for **G**aussian **S**platting. The method organizes context construction and attribute coding into a connected causal structure within each anchor. Coordinate-derived geometry context is used to model a compact learnable anchor latent, which is then fused with the geometry context to form an anchor context. The anchor context supports modeling and coding across all attribute groups. All context modeling uses only anchor-wise linear transformations and activations, yielding a simple compression architecture without spatial context aggregation. We further develop an integer inference pipeline for the context model, with quantization-aware training to adapt the networks to integer arithmetic. The pipeline maintains consistency throughout context construction and probability prediction, enabling consistent entropy decoding across platforms.

Our contributions are summarized as follows:

- We propose COSA-GS, a simple 3DGS compression framework that integrates context construction and attribute coding through anchor-wise causal factorization without spatial context aggregation.
- We develop an integer inference pipeline with quantization-aware training for the context model, enabling consistent entropy decoding across heterogeneous platforms.
- We combine the proposed model with rate–distortion optimization and adaptive masking, achieving state-of-the-art compression performance and fast decoding on Mip-NeRF360, DeepBlending, Tanks&Temples, and BungeeNeRF.

2 Related Work

2.1 3D Gaussian Splatting Compression

The explicit scene representation of 3DGS enables efficient rendering but incurs substantial storage cost (Kerbl et al., 2023). Recent work has explored compression and compaction through pruning, quantization, compact parameterization, and entropy coding (Lee et al., 2024; Fan et al., 2024; Niedermayr et al., 2024; Girish et al., 2024; Papantonakis et al., 2024; Shin et al., 2025). Post-training and training-free methods reduce storage through pruning and compact attribute coding (Xie et al., 2024; Tian et al., 2025). Predictive coding and structured feature-plane representations further reduce redundancy in Gaussian attributes (Liu et al., 2024; Lee et al., 2025). Scene-adaptive lattice vector quantization improves the rate–distortion efficiency of anchor-based compression (Xu et al., 2026). Gaussian pruning and sparsification reduce model size by removing redundant primitives (Hanson et al., 2025; Zhang et al., 2025). Differentiable rate–distortion optimization directly balances storage cost and rendering quality during scene optimization (Wang et al., 2024a). Anchor-based parameterization represents local Gaussians with anchors and learnable offsets while predicting view-dependent attributes from anchor features and viewing conditions (Lu et al., 2024), providing a compact representation for learned 3DGS compression.

2.2 Context Modeling for 3DGS Compression

Context modeling improves entropy coding by conditioning symbol probabilities on information available at the decoder, a principle widely used in learned image compression (Ballé et al., 2018; Minnen et al., 2018). Applying this idea to irregular 3D representations is challenging because neighboring elements do not lie on dense regular grids. HAC and HAC++ construct spatial context by querying a learned hash grid at anchor locations for anchor-attribute distribution modeling (Chen et al., 2024; 2025). ContextGS instead builds a coarse-to-fine anchor hierarchy and conditions finer anchors on already decoded coarser anchors, together with a low-dimensional per-anchor hyperprior (Wang et al., 2024b). PCGS builds on the HAC++ framework with progressive masking and progressive quantization for incremental bitstreams (Chen et al., 2026). These methods construct context through spatial feature fields or inter-anchor dependencies. COSA-GS instead constructs context through causal dependencies among coordinates, latents, and attribute groups within each anchor, retaining conditional context modeling without spatial aggregation.

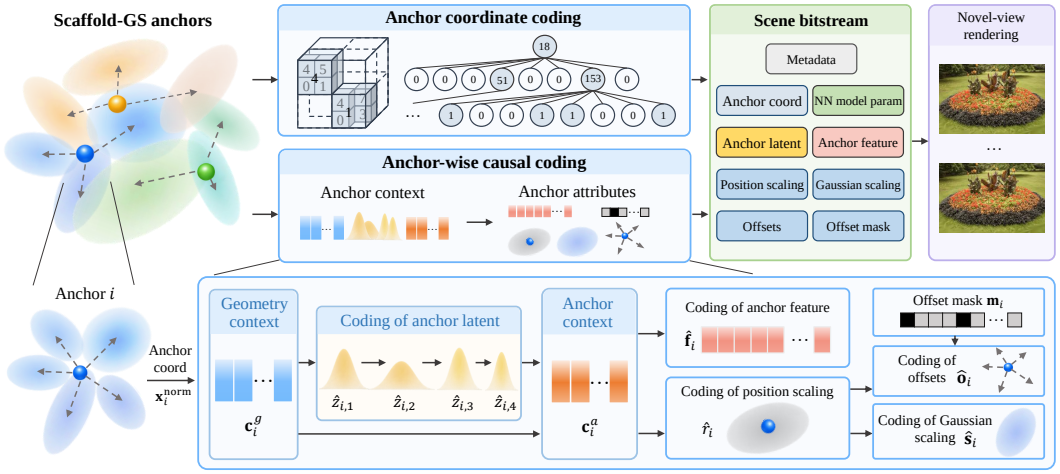

Figure 1: Main architecture of COSA-GS. The top shows the overall coding and rendering pipeline, while the bottom details the anchor-wise causal structure. Geometry context supports anchor-latent modeling and is fused with the anchor latent to form the anchor context for attribute coding.

2.3 Cross-Platform Decoding in Neural Compression

Neural compression requires numerical consistency between the encoder and decoder, as even small differences in predicted probabilities can cause entropy-decoding failures. Ballé et al. (2019) address platform-dependent floating-point behavior using integer networks for learned image compression. Koyuncu et al. (2022) further study cross-device consistency through weight and activation quantization of entropy networks. For video compression, DCVC-RT incorporates model integerization to support consistent cross-device coding (Jia et al., 2025). Integer inference has also been developed for cross-platform lossless LiDAR point cloud compression (Yu et al., 2026). An alternative approach transmits calibration information to correct numerical mismatches at the cost of additional side information (Tian et al., 2023). COSA-GS applies integer inference to the context model of anchor-based 3DGS, maintaining consistency of entropy-decoded symbols across platforms.

3 Method

3.1 Preliminaries

Scaffold-GS (Lu et al., 2024) represents a scene with anchors, each parameterizing K Gaussian primitives. For anchor i, we denote its attributes by $\mathcal{V}_i = (\mathbf{f}_i, r_i, \mathbf{o}_i, \mathbf{s}_i)$, where $\mathbf{f}_i \in \mathbb{R}^{C_f}$ is the anchor feature, r_i is the log-domain position scaling, $\mathbf{o}_i \in \mathbb{R}^{K\times 3}$ contains the Gaussian offsets, and $\mathbf{s}_i \in \mathbb{R}^3$ is the log-domain Gaussian scaling. Each anchor has a coordinate $\mathbf{x}_i \in \mathbb{R}^3$, with the k-th Gaussian centered at $\mathbf{p}_{i,k} = \mathbf{x}_i + \exp(r_i)\mathbf{o}_{i,k}$. Given a target view, shared MLPs predict view-dependent Gaussian attributes from the anchor feature and viewing conditions:

$$\{(\mathbf{c}_{i,k}, \alpha_{i,k}, \mathbf{q}_{i,k}, \mathbf{u}_{i,k})\}_{k=1}^{K} = \mathcal{D}_\theta(\mathbf{f}_i, \mathbf{v}_i), \quad \mathbf{s}^{\mathrm{G}}_{i,k} = \exp(\mathbf{s}_i) \odot \mathrm{sigmoid}(\mathbf{u}_{i,k}), \tag{1}$$

where $\mathbf{v}_i$ contains the viewing direction and distance, and $\mathcal{D}_\theta$ denotes the shared prediction MLPs and their output transformations. Here, $\mathbf{c}_{i,k}$, $\alpha_{i,k}$, and $\mathbf{q}_{i,k}$ denote color, opacity, and rotation quaternion, respectively, while $\mathbf{u}_{i,k}$ modulates the base scaling $\exp(\mathbf{s}_i)$. These attributes define the Gaussian primitives used for rendering. Scaffold-GS therefore stores anchor attributes and shared MLP weights rather than explicit attributes for every Gaussian.

3.2 Overview

As illustrated in Figure 1, COSA-GS augments the Scaffold-GS representation with learnable anchor latents and compresses it through anchor-wise causal coding. For a scene with N anchors, we define

$\mathcal{A} = \{(\mathbf{x}_i, \mathbf{z}_i, \mathcal{V}_i, \mathbf{m}_i)\}_{i=1}^{N}$, where $\mathbf{x}_i$ is the anchor coordinate, $\mathbf{z}_i \in \mathbb{R}^{C_z}$ is the anchor latent, $\mathcal{V}_i$ denotes the anchor attributes, and $\mathbf{m}_i \in \{0,1\}^K$ is a mask indicating the active Gaussian offsets.

We encode this representation using octree occupancy coding for anchor coordinates and anchor-wise causal coding for anchor latents and attributes. The anchor latent is modeled using coordinate-derived geometry context and then fused with this context to form the anchor context. The anchor context supports modeling and coding across all attribute groups. The resulting bitstream is decoded to recover the Scaffold-GS representation for novel-view rendering. Further, quantization-aware training and integer inference for the context model enable consistent cross-platform decoding of entropy-coded symbols.

3.3 Anchor Coordinate Coding

We encode anchor coordinates with octree occupancy coding based on empirical distributions. The coordinates are first quantized as $\mathbf{q}_i = \lfloor \mathbf{x}_i / \Delta_x \rceil$, where Δ_x is the coordinate quantization step and $\lfloor \cdot \rceil$ denotes rounding. We construct the octree by recursively grouping occupied cells into their parent cells. For parent j at level ℓ, $\omega_{\ell,j} \in \{1, \ldots, 255\}$ denotes the occupancy pattern of its eight children. Occupancy symbols at the same level share an empirical distribution:

$$p_\ell(v) = \frac{1}{N_\ell} \sum_{j=1}^{N_\ell} \mathbf{1}[\omega_{\ell,j} = v], \tag{2}$$

where N_ℓ denotes the number of occupancy symbols at level ℓ.

During decoding, the octree is expanded level by level to recover the finest-level integer coordinates exactly. The reconstructed coordinates $\hat{\mathbf{x}}_i = \Delta_x \mathbf{q}_i$ are subsequently used for geometry context construction.

3.4 Anchor-Latent Coding and Context Construction

We construct the anchor context by modeling the anchor latent from geometry context and then fusing the anchor latent with the geometry context. For anchor i, we first construct a *geometry context* $\mathbf{c}_i^g = E_x(\mathbf{x}_i^{\text{norm}})$, where $\mathbf{x}_i^{\text{norm}}$ is obtained by normalizing the reconstructed coordinate $\hat{\mathbf{x}}_i$, and E_x is an MLP. To enhance this coordinate-derived context, we introduce a compact anchor latent $\mathbf{z}_i$ that is optimized for each scene. Its joint distribution is factorized as:

$$p(\hat{\mathbf{z}}_i \mid \mathbf{c}_i^g) = \prod_{c=1}^{C_z} p(\hat{z}_{i,c} \mid \mathbf{c}_i^g, \hat{\mathbf{z}}_{i,<c}), \tag{3}$$

where hats denote quantized reconstructions. Each latent channel uses a separate MLP to predict its Gaussian mean and standard deviation from the geometry context and previously reconstructed latent channels. Latent channels are therefore decoded sequentially, while anchors can be processed in parallel.

We use a Gaussian distribution model for anchor latents and attributes. For a scalar variable v in group $g \in \{z, f, r, o, s\}$, the predicted mean μ and a learned quantization step $\Delta_g > 0$ determine its integer residual and reconstruction, while the predicted standard deviation σ determines its coding probability:

$$k_v = \lfloor (v - \mu)/\Delta_g \rceil, \qquad \hat{v} = \mu + \Delta_g k_v, \qquad \ell_v = -\log_2 P(k_v; \sigma), \tag{4}$$

where $P(k_v; \sigma)$ denotes the probability assigned to the residual symbol by a discretized zero-mean Gaussian with standard deviation σ, and ℓ_v is the estimated coding cost. The quantization step Δ_g is shared across anchors and components within each group. During training, rounding is replaced by additive uniform noise, yielding relaxed anchor latents $\tilde{\mathbf{z}}_i$ for subsequent channel prediction and anchor-context construction.

After all latent channels are reconstructed, we construct the *anchor context*:

$$\mathbf{c}_i^a = F_a\left([\mathbf{c}_i^g; E_z(\hat{\mathbf{z}}_i)]\right), \tag{5}$$

where $[\cdot;\cdot]$ denotes concatenation, and E_z and F_a are linear transformations with GELU activations. The resulting anchor context combines coordinate-derived and learned scene-specific information for attribute distribution modeling.

3.5 ANCHOR-ATTRIBUTE CODING

Attribute coding uses the anchor context for all attribute groups. For anchor i, we denote the attributes by $\mathcal{V}_i = (\mathbf{f}_i, r_i, \mathbf{o}_i, \mathbf{s}_i)$, representing the anchor feature, position scaling, offsets, and Gaussian scaling, respectively. Their conditional distribution is factorized as

$$p(\hat{\mathcal{V}}_i \mid \mathbf{c}_i^a, \mathbf{m}_i) = p(\hat{\mathbf{f}}_i \mid \mathbf{c}_i^a)\ p(\hat{r}_i \mid \mathbf{c}_i^a)\ p(\hat{\mathbf{o}}_i \mid \mathbf{c}_i^a, \hat{r}_i, \mathbf{m}_i)\ p(\hat{\mathbf{s}}_i \mid \mathbf{c}_i^a, \hat{r}_i), \tag{6}$$

where $\mathbf{m}_i \in \{0,1\}^K$ is a binary offset mask obtained from learnable logits $\boldsymbol{\eta}_i$ as $\mathbf{m}_i = \mathbf{1}[\mathrm{sigmoid}(\boldsymbol{\eta}_i) > \epsilon_m]$. Here, ϵ_m is the masking threshold, and straight-through gradients are used for binarization during training. The feature distribution $p(\hat{\mathbf{f}}_i \mid \mathbf{c}_i^a)$ and position-scaling distribution $p(\hat{r}_i \mid \mathbf{c}_i^a)$ are modeled directly from the anchor context $\mathbf{c}_i^a$ using separate MLPs, with all feature channels predicted in parallel. After reconstructing $\hat{r}_i$, we embed it and process the embedding with two branch-specific MLPs, and the resulting features are added to $\mathbf{c}_i^a$ to model the offset distribution $p(\hat{\mathbf{o}}_i \mid \mathbf{c}_i^a, \hat{r}_i, \mathbf{m}_i)$ and Gaussian-scaling distribution $p(\hat{\mathbf{s}}_i \mid \mathbf{c}_i^a, \hat{r}_i)$. The mask $\mathbf{m}_i$ specifies the active offsets, so that only active offsets are modeled and included in the bitstream.

All attribute groups follow the Gaussian distribution modeling in Eq. (4), with a separate quantization step for each group. In short, context construction and attribute coding form a connected causal structure within each anchor.

3.6 RATE–DISTORTION OPTIMIZATION

We optimize COSA-GS under a rate–distortion objective that balances rendering quality and the estimated coding rates of anchor latents and attributes. For anchor i, let $\ell_{i,j}^g$ denote the coding cost of the j-th scalar element of group $g \in \{z, f, r, s\}$, and let $\ell_{i,k}^o$ denote the coding cost of the k-th offset. Here, $m_{i,k}$ is the k-th element of the offset mask $\mathbf{m}_i$, and $a_i = \mathbf{1}[\sum_k m_{i,k} > 0]$ indicates whether anchor i is active. The corresponding rate terms are

$$R_g = \sum_i a_i \sum_j \ell_{i,j}^g, \quad g \in \{z, f, r, s\}, \qquad R_o = \sum_i \sum_k m_{i,k} \ell_{i,k}^o, \tag{7}$$

where the anchor-level rate terms R_g are applied only to active anchors, and the offset rate R_o is applied only to active offsets. The mask m_i suppresses inactive Gaussians during rendering, and anchors with no active offsets are pruned. The total training rate is $\mathcal{L}_{\mathrm{rate}} = R_z + R_f + R_r + R_o + R_s$.

For a training view with reference image I and rendered image $\tilde{I}$, the training objective is

$$\mathcal{L} = (1-\alpha)\mathcal{L}_1(\tilde{I}, I) + \alpha\big(1 - \mathrm{SSIM}(\tilde{I}, I)\big) + \lambda \mathcal{L}_{\mathrm{rate}}/(8 \cdot 10^6), \tag{8}$$

where $\mathcal{L}_1$ denotes the mean absolute image error, and $\alpha = 0.2$ weights the SSIM term. The parameter λ controls the rate–distortion trade-off.

3.7 INTEGER INFERENCE FOR CROSS-PLATFORM DECODING

Cross-platform entropy decoding requires consistent context modeling at the encoder and decoder. However, floating-point computations can produce platform-dependent numerical differences and lead to decoding failures. We therefore implement the context model entirely in integer arithmetic to ensure bit-exact consistency of entropy-decoded symbols across platforms.

Quantization. We quantize activations using per-tensor affine quantization and weights using per-channel symmetric quantization, both with 8-bit representations. For an activation tensor $\mathbf{x}$, the quantized representation is

$$\mathbf{q}_x = \mathrm{clip}\left(\lfloor \mathbf{x}/s_x \rceil + z_x,\, q_{\min},\, q_{\max}\right), \tag{9}$$

where s_x and z_x are the quantization scale and zero point, and $q_{\min}$ and $q_{\max}$ denote the quantization bounds. During training, we apply quantization-aware training to adapt the context model to these integer representations. The quantized activations are dequantized as $s_x(\mathbf{q}_x - z_x)$ for forward computation, with straight-through gradients through rounding. At inference, the corresponding operations are performed directly in integer arithmetic.

Integer linear. Linear transformations use 8-bit integer activations and weights with 32-bit integer accumulation:

$$\mathbf{a} = \mathbf{Q}_w(\mathbf{q}_x - z_x) + \mathbf{q}_b, \tag{10}$$

where $\mathbf{q}_x$ is the quantized input, $\mathbf{Q}_w$ and $\mathbf{q}_b$ are the quantized weight matrix and bias, z_x is the input zero point, and $\mathbf{a}$ is the accumulated integer output. The subtraction of z_x is applied element-wise. Since the weights use symmetric quantization, their zero point is zero. To convert the accumulated output to the quantization scale of the next layer, we use fixed-point rescaling. The scale conversion is represented by integer multipliers $\boldsymbol{\rho}$ and a power-of-two divisor 2^τ, giving

$$\mathbf{q}_y = \mathrm{clip}\left(\left\lfloor \frac{\mathbf{a} \odot \boldsymbol{\rho}}{2^\tau} \right\rceil + z_y,\, q_{\min},\, q_{\max}\right), \tag{11}$$

where $\boldsymbol{\rho}$ contains the precomputed integer multipliers, τ is the right-shift factor, z_y is the output zero point, and $\odot$ denotes element-wise multiplication.

Integer GELU. The GELU activation is $\mathrm{GELU}(x) = x\Phi(x)$, where Φ is the standard Gaussian cumulative distribution function. To avoid floating-point calculation of Φ, we rewrite GELU as

$$\mathrm{GELU}(x) = \max(x, 0) - h(|x|), \quad h(t) = t\,\Phi(-t). \tag{12}$$

This decomposition isolates the nonlinear term into $h(|x|)$. We approximate h using a quantized lookup table $\mathcal{T}$. Since h approaches zero for large $|x|$, the lookup table only needs to cover a bounded nonnegative range, while $\max(x, 0)$ can be evaluated directly in integer arithmetic. For a fixed-point input u, the integer GELU is computed as $\mathcal{G}(u) = \max(u, 0) - \mathcal{I}_\mathcal{T}(|u|)$, where $\mathcal{I}_\mathcal{T}$ denotes integer linear interpolation of the lookup-table values at the corresponding fixed-point scale. The correction term is set to zero outside the tabulated range.

Integer Gaussian modeling. We use precomputed integer cumulative distribution function (CDF) tables to ensure consistent Gaussian entropy coding across platforms. The tables represent the discretized zero-mean Gaussian distributions in Eq. (4). The standard-deviation index predicted by the context model is rounded and clipped using integer arithmetic to select the corresponding CDF table. For reconstruction, the residual symbol k_v is rescaled by Δ_g and added to the predicted mean μ in fixed-point arithmetic:

$$q_{\hat{v}} = q_\mu + \left\lfloor (k_v \rho_v)/2^{\tau_v} \right\rceil, \tag{13}$$

where q_μ and $q_{\hat{v}}$ are the fixed-point representations of the predicted mean μ and reconstructed value $\hat{v}$. The integer multiplier ρ_v and right-shift factor τ_v are precomputed from the quantization step Δ_g and the fixed-point scale of q_μ. Therefore, CDF selection and reconstruction for subsequent context modeling use integer arithmetic, maintaining consistency across platforms.

4 Experiments

4.1 Settings

Datasets and compared methods. Experiments are conducted on Mip-NeRF360 (Barron et al., 2022), DeepBlending (Hedman et al., 2018), Tanks&Temples (Knapitsch et al., 2017), and BungeeNeRF (Xiangli et al., 2022). We compare against ContextGS (Wang et al., 2024b), HAC++ (Chen et al., 2025), CAT-3DGS (Zhan et al., 2025), PCGS (Chen et al., 2026), and SpeedyGS (Zhang et al., 2026). We report dataset-average PSNR, SSIM, and LPIPS for novel-view rendering, together with compressed scene size in MiB. Higher PSNR and SSIM and lower LPIPS indicate better novel-view quality. The rate–distortion results of the compared methods are based on the per-scene metrics reported in the corresponding papers.

Implementation details. We initialize from a Scaffold-GS representation optimized for 15K iterations and use 30K rate–distortion optimization iterations as the default setting. The notation 15+30K therefore denotes 45K iterations in total. We also evaluate 15+20K and 15+45K settings and a 15+30K quantization-aware training (QAT) variant. Unless otherwise specified, experiments use an NVIDIA 5880 Ada GPU and an AMD EPYC 9654 CPU. More implementation details are provided in the appendix and the anonymized repository.



Figure 2: Qualitative comparison on the *flowers* scene from Mip-NeRF360. The top row shows the target view *DSC9144* and the corresponding renderings, while the bottom row shows zoomed-in views of a distinctive region. Scene sizes and coding times are reported for the full scene. PSNR is measured on the displayed target view. Coding times are measured on a platform with an NVIDIA 5880 Ada GPU and an AMD EPYC 9654 CPU.

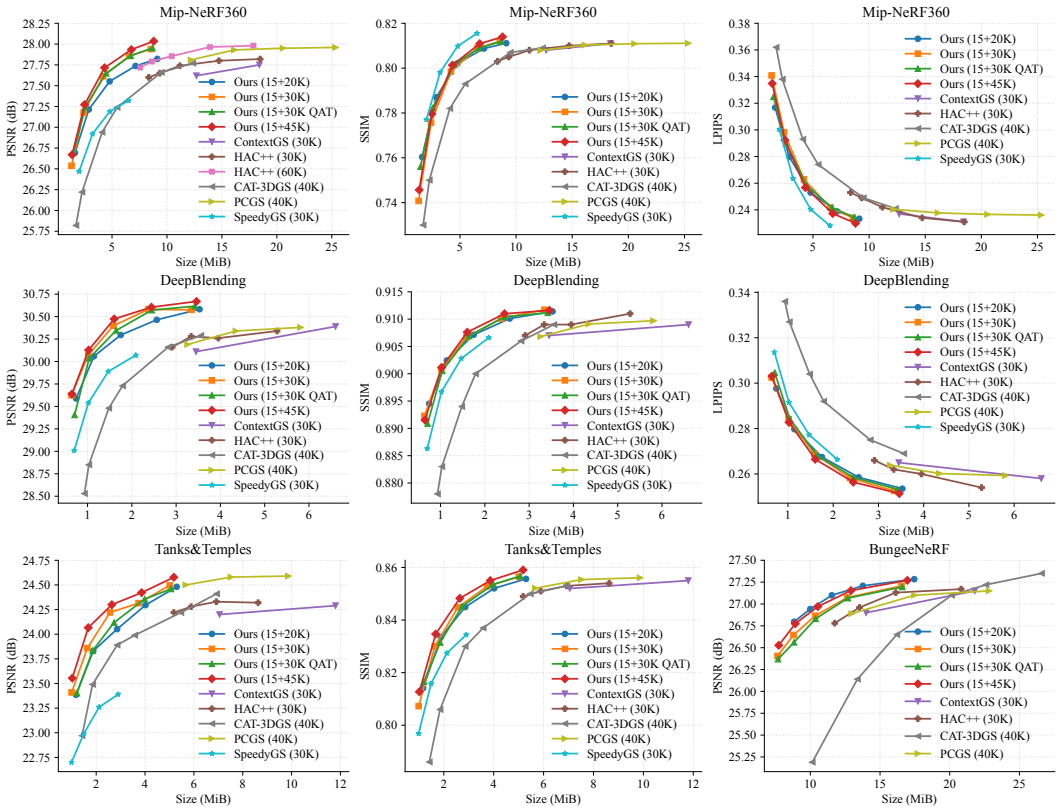

Figure 3: Rate–distortion comparison with existing methods. The first two rows show PSNR, SSIM, and LPIPS on Mip-NeRF360 and DeepBlending, respectively. The last row shows PSNR and SSIM on Tanks&Temples and PSNR on BungeeNeRF. Training steps are shown in parentheses, and QAT denotes quantization-aware training.

4.2 Performance Analysis

Rate–distortion performance. Figure 3 compares the rate–distortion performance across the four datasets. We first examine Mip-NeRF360, where the default 15+30K setting of COSA-GS achieves a favorable PSNR–size trade-off compared with existing methods, including HAC++ with 60K op-

Table 1: Novel-view quality, size, and runtime of COSA-GS (15+30K) on Mip-NeRF360.

λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)	Entropy Dec. (s)	Anchor Coord. Dec. (s)	NN Dec. (s)	Train (s) Scaffold15K	Train (s) RD 30K
.0006	27.94	0.812	0.234	8.55	0.090	0.112	0.090	0.017	0.008	479	3321
.0010	27.86	0.809	0.242	6.56	0.080	0.101	0.082	0.015	0.007		3279
.0020	27.61	0.798	0.263	4.20	0.064	0.081	0.065	0.012	0.006		3238
.0040	27.18	0.776	0.298	2.42	0.048	0.062	0.048	0.010	0.004		3206
.0080	26.54	0.741	0.341	1.32	0.036	0.044	0.034	0.007	0.003		3210
Avg.	27.42	0.787	0.276	4.61	0.063	0.080	0.064	0.012	0.005	479	3251

Table 2: Cross-platform decoding times of COSA-GS (15+30K, QAT) on Mip-NeRF360. All bitstreams are encoded on the NVIDIA 5880 Ada and AMD EPYC 9654 platform and decoded on the listed GPU–CPU configurations. Times are in seconds. The final column reports the number of bitstreams with identical entropy-decoded symbols and anchor coordinates out of 45 tested bitstreams.

GPU	CPU	Total decoding time (s)						Decoding
		$\lambda = .0006$	.001	.002	.004	.008	Average	consistency
NVIDIA 5880Ada	AMD EPYC 9654	0.121	0.110	0.096	0.076	0.061	0.093	45/45
NVIDIA 4090	AMD EPYC 7R32	0.145	0.134	0.114	0.090	0.071	0.111	45/45
NVIDIA V100	Intel Xeon Gold 6258R	0.184	0.173	0.149	0.120	0.096	0.144	45/45

timization steps. As reported in Table 1, COSA-GS covers 1.32–8.55 MiB and 26.54–27.94 dB on Mip-NeRF360, reaching 27.61 dB at 4.20 MiB. Meanwhile, the SSIM and LPIPS curves also show better overall trade-offs than ContextGS, HAC++, CAT-3DGS, and PCGS at comparable sizes. SpeedyGS achieves higher SSIM and lower LPIPS at several bitrate points, while COSA-GS provides a stronger PSNR–size trade-off. On DeepBlending, Tanks&Temples, and BungeeNeRF, the default 15+30K setting similarly achieves favorable PSNR with smaller scene sizes than the compared methods. On the *flowers* scene in Figure 2, COSA-GS achieves a decoding time of 0.1279 s, compared with 24.95 s for HAC++ and 25.49 s for PCGS, while producing a smaller bitstream and slightly better rendering quality.

Figure 3 also compares different numbers of optimization steps. Increasing the optimization steps generally improves rate–distortion performance, but the gain gradually diminishes and varies across datasets. The improvement is more evident on Mip-NeRF360, DeepBlending, and Tanks&Temples, whereas the curves on BungeeNeRF remain close across the 15+20K, 15+30K, and 15+45K settings. The QAT variant applies quantization-aware training together with the integer context-model inference described in Sec. 3.7. Its rate–distortion curves remain close to the floating-point 15+30K model over most bitrate points, with only small differences at some low-bitrate points on DeepBlending and BungeeNeRF. Overall, these results show that the proposed anchor-wise context modeling is effective across the four benchmarks.

Computational efficiency. Table 1 reports novel-view quality, scene size, runtime, and training time on Mip-NeRF360. Total decoding takes 0.044–0.112 s across the five bitrate points, averaging 0.080 s. The entropy coder is the dominant decoding component, taking 0.064 s on average, whereas anchor-coordinate decoding takes only 0.012 s. The neural context model requires only 0.003–0.008 s, indicating that the anchor-wise context modeling introduces little decoding overhead. Note that the sum of the component decoding times may slightly exceed the total decoding time, as expected with asynchronous GPU execution. For training, the shared 15K Scaffold-GS initialization takes 479 s, while the subsequent 30K rate–distortion optimization takes 3,206–3,321 s across bitrate points and accounts for most of the training time.

Cross-platform decoding. We encode the scene bitstreams on the NVIDIA 5880 Ada and AMD EPYC 9654 platform and decode the same streams on the three GPU–CPU configurations in Table 2. On the reference 5880 Ada platform, dataset-average decoding takes 0.061–0.121 s across the five bitrate points, averaging 0.093 s. The corresponding ranges are 0.071–0.145 s on the NVIDIA 4090 configuration and 0.096–0.184 s on the NVIDIA V100 configuration. Across all three platforms, decoding time decreases toward lower bitrate points, reflecting the reduced amount of coded data.

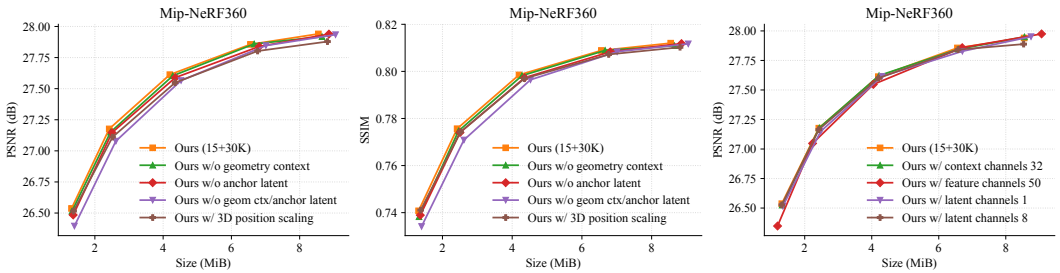

Figure 4: Ablation studies on Mip-NeRF360 under the 15+30K setting. Left and middle: the effects of geometry context, anchor latents, and position-scaling dimensionality. Right: comparisons of different numbers of context, feature, and latent channels.

We verify cross-platform consistency by checking exact equality of the entropy-decoded symbols for anchor latents, anchor attributes, offset masks, and the decoded anchor coordinates. All bitstreams of the 5 bitrate points on the 9 Mip-NeRF360 scenes are decoded consistently across platforms.

4.3 Ablation Studies

We conduct ablations on Mip-NeRF360 using the 15+30K setting to validate context construction, position-scaling dimensionality, and model capacity.

Geometry context and anchor latent. Figure 4 compares the full model with variants that remove the geometry context, the anchor latent, or both. Removing either source causes a small degradation, indicating that the two context sources provide complementary information. When both are removed, performance drops noticeably across all bitrate points, showing that anchor context is essential to the compression performance of COSA-GS. The geometry-free variant remains close to the full model, further indicating that the learnable anchor latent can provide informative context even without coordinate conditioning.

Position scaling. COSA-GS uses a scalar position scaling for each anchor to scale its offsets uniformly. To examine whether additional directional flexibility benefits compression, we replace the scalar with a three-dimensional vector. Figure 4 shows that this variant provides no overall rate–distortion improvement and slightly degrades performance at high bitrate points. The additional scaling degrees of freedom therefore increase the amount of scaling information to be coded without providing sufficient quality gains to offset the added rate.

Model capacity. The right panel of Figure 4 evaluates the number of channels used for the context, anchor feature, and anchor latent. Increasing the context channels from 24 to 32 provides little improvement. Increasing the feature channels from 32 to 50 slightly improves PSNR at the highest bitrate point but degrades performance at lower bitrates. Changing the number of latent channels from the default four to either one or eight also yields similar rate–distortion curves, with no consistent gain from using more latent channels. Overall, increasing the number of channels provides limited benefit, supporting the default configuration of COSA-GS.

5 Conclusion

We presented COSA-GS, an anchor-wise causal framework for practical 3DGS compression without spatial context aggregation. A compact anchor latent is modeled from coordinate-derived geometry context and then fused with this context to provide context for attribute coding, yielding a simple context model composed only of linear transformations and activations. Combined with rate–distortion optimization, quantization-aware training, and integer inference, COSA-GS achieves strong compression performance, fast decoding, and cross-platform consistency. Experiments across four benchmarks demonstrate the effectiveness of this simple anchor-wise architecture for practical 3DGS compression.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language editing and the implementation and debugging of code. We have reviewed the revised text and inspected and tested the AI-assisted code. AI assistance was limited to these tasks. We take responsibility for the manuscript, code, and reported results.

REPRODUCIBILITY STATEMENT

We provide anonymized source code in the supplementary materials and at `https://anonymous.4open.science/r/COSA-GS`, including model implementations, training configurations, and scripts for compression, decompression, and evaluation. The accompanying README provides environment setup instructions and commands for running the experiments. We encourage readers to inspect the implementation and reproduce the experiments. A pretrained compressed scene and a single-view evaluation demo also allow readers to test consistent cross-platform decoding and inspect rendering results without training.

REFERENCES

Milena T. Bagdasarian, Paul Knoll, Yi-Hsin Li, Florian Barthel, Anna Hilsmann, Peter Eisert, and Wieland Morgenstern. 3DGS.zip: A survey on 3D Gaussian splatting compression methods. *Computer Graphics Forum*, 44(2):e70078, 2025.

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations (ICLR)*, 2018.

Johannes Ballé, Nick Johnston, and David Minnen. Integer networks for data compression with latent-variable models. In *International Conference on Learning Representations (ICLR)*, 2019.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5470–5479, 2022.

Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. HAC: Hash-grid assisted context for 3D Gaussian splatting compression. In *European Conference on Computer Vision (ECCV)*, pp. 422–438, 2024.

Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. HAC++: Towards 100X compression of 3D Gaussian splatting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(11):10210–10226, 2025.

Yihang Chen, Mengyao Li, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. PCGS: Progressive compression of 3D Gaussian splatting. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, pp. 3111–3119, 2026.

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. LightGaussian: Unbounded 3D Gaussian compression with 15x reduction and 200+ FPS. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 140138–140158, 2024.

Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. EAGLES: Efficient accelerated 3D Gaussians with lightweight EncodingS. In *European Conference on Computer Vision (ECCV)*, pp. 54–71, 2024.

Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. PUP 3D-GS: Principled uncertainty pruning for 3D Gaussian splatting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5949–5958, 2025.

Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics*, 37(6), 2018.

Zhaoyang Jia, Bin Li, Jiahao Li, Wenxuan Xie, Linfeng Qi, Houqiang Li, and Yan Lu. Towards practical real-time neural video compression. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12543–12552, 2025.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and Temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017.

Esin Koyuncu, Timofey Solovyev, Elena Alshina, and André Kaup. Device interoperability for learned image compression with weights and activations quantization. In *Picture Coding Symposium (PCS)*, pp. 151–155, 2022.

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3D Gaussian representation for radiance field. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21719–21728, 2024.

Soonbin Lee, Fangwen Shu, Yago Sanchez, Thomas Schierl, and Cornelius Hellge. Compression of 3D Gaussian splatting with optimized feature planes and standard video codecs. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 25496–25505, 2025.

Xiangrui Liu, Xinju Wu, Pingping Zhang, Shiqi Wang, Zhu Li, and Sam Kwong. CompGS: Efficient 3D scene representation via compressed Gaussian splatting. In *ACM International Conference on Multimedia (ACM MM)*, pp. 2936–2944, 2024.

Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-GS: Structured 3D Gaussians for view-adaptive rendering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20654–20664, 2024.

David Minnen, Johannes Ballé, and George Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 10771–10780, 2018.

Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3D Gaussian splatting for accelerated novel view synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10349–10358, 2024.

Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3D Gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):1–17, 2024.

Seungjoo Shin, Jaesik Park, and Sunghyun Cho. Locality-aware gaussian compression for fast and high-quality rendering. In *International Conference on Learning Representations (ICLR)*, pp. 63–90, 2025.

Boyuan Tian, Qizhe Gao, Siran Xianyu, Xiaotong Cui, and Minjia Zhang. FlexGaussian: Flexible and cost-effective training-free compression for 3D Gaussian splatting. In *ACM International Conference on Multimedia (ACM MM)*, pp. 7287–7296, 2025.

Kuan Tian, Yonghang Guan, Jinxi Xiang, Jun Zhang, Xiao Han, and Wei Yang. Towards real-time neural video codec for cross-platform application using calibration information. In *ACM International Conference on Multimedia (ACM MM)*, pp. 7961–7970, 2023.

Henan Wang, Hanxin Zhu, Tianyu He, Runsen Feng, Jiajun Deng, Jiang Bian, and Zhibo Chen. End-to-end rate-distortion optimized 3D Gaussian representation. In *European Conference on Computer Vision (ECCV)*, pp. 76–92, 2024a.

Yufei Wang, Zhihao Li, Lanqing Guo, Wenhan Yang, Alex C. Kot, and Bihan Wen. ContextGS: Compact 3D Gaussian splatting with anchor level context model. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 51532–51551, 2024b.

Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. BungeeNeRF: Progressive neural radiance field for extreme multi-scale scene rendering. In *European Conference on Computer Vision (ECCV)*, pp. 106–122, 2022.

Shuzhao Xie, Weixiang Zhang, Chen Tang, Yunpeng Bai, Rongwei Lu, Shijia Ge, and Zhi Wang. MesonGS: Post-training compression of 3D Gaussians via efficient attribute transformation. In *European Conference on Computer Vision (ECCV)*, pp. 434–452, 2024.

Hao Xu, Xiaolin Wu, and Xi Zhang. Improving 3D Gaussian splatting compression by scene-adaptive lattice vector quantization. *IEEE Transactions on Image Processing*, 35:5864–5879, 2026.

Pengpeng Yu, Haoran Li, Runqing Jiang, Dingquan Li, Jing Wang, Liang Lin, and Yulan Guo. Towards practical lossless neural compression for LiDAR point clouds. In *European Conference on Computer Vision (ECCV)*, pp. 115–133, 2026.

Yu-Ting Zhan, Cheng-Yuan Ho, He-Bi Yang, Yi-Hsin Chen, Jui Chiu Chiang, Yu-Lun Liu, and Wen-Hsiao Peng. CAT-3DGS: A context-adaptive triplane approach to rate-distortion-optimized 3DGS compression. In *International Conference on Learning Representations (ICLR)*, pp. 26312–26334, 2025.

Junteng Zhang, Tong Chen, Yuxin Zhao, Yibo Shi, Jing Wang, and Zhan Ma. SpeedyGS: Content-aware 3D Gaussian splatting compression via two-stage optimization. In *ACM International Conference on Multimedia (ACM MM)*, 2026.

Yangming Zhang, Wenqi Jia, Wei Niu, and Miao Yin. GaussianSpa: An "optimizing-sparsifying" simplification framework for compact and high-quality 3D Gaussian splatting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26673–26682, 2025.

A APPENDIX

This appendix provides implementation and runtime details, bit allocation analysis, extended rate–distortion comparisons, additional ablation studies, and detailed results for individual scenes.

A.1 DATASET CONFIGURATION

We follow the image-resolution settings and training/test view splits used in the publicly released code of Scaffold-GS, HAC++, PCGS, and CAT-3DGS. We evaluate on nine scenes from Mip-NeRF360, two from Tanks&Temples, two from DeepBlending, and six from BungeeNeRF. Images wider than 1,600 pixels are downsampled to a width of 1,600 while preserving their aspect ratio, and smaller images retain their original resolution. Views are sorted by image filename before splitting. For Mip-NeRF360, Tanks&Temples, and DeepBlending, every eighth view, starting from the first, is held out for testing, and the remaining views are used for training. For BungeeNeRF, the first 31 views are used for testing and the remaining views for training.

A.2 BIT ALLOCATION.

Figure 5 shows the bitstream composition at three rate settings on Mip-NeRF360. Anchor features and offsets account for most of the bitstream, contributing 77.3% at $\lambda = 0.0006$ and 53.8% at $\lambda = 0.008$. Anchor latents contribute 3.4%–8.5%, providing context for multiple attribute distributions with a comparatively small cost. As the total rate decreases, the fraction of anchor coordinates increases from 5.8% to 15.4%, while the fraction of model parameters and metadata increases from 0.6% to 3.9%. The low-rate bitstreams therefore allocate a larger fraction to anchor coordinates, model parameters, and metadata as the coding costs of features and offsets decrease.

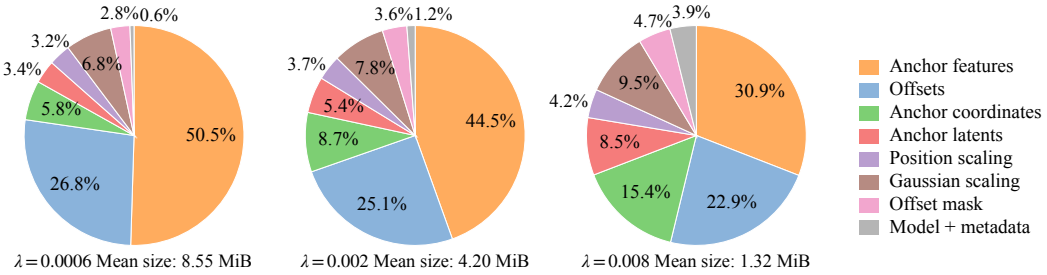

Figure 5: Bitstream composition on Mip-NeRF360 for $\lambda = 0.0006$, 0.002, and 0.008 under the 15+30K setting. The corresponding mean scene sizes are 8.55, 4.20, and 1.32 MiB, respectively.

A.3 RUNTIME MEASUREMENT

Measurement protocol. We measure encoding and decoding times after one warmup run in the same process. For encoding–decoding evaluation, we first encode the trained model and decode the resulting bitstream as a warmup, then repeat the complete procedure and record the encoding and decoding times separately. For decoding-only evaluation, we decode the same bitstream once as a warmup and measure the second decoding run. Warmup is excluded from the reported times.

Timing boundaries. We use a host wall clock and synchronize CUDA immediately before and after each measured encoding or decoding call. Encoding starts from a trained model in memory and ends with an in-memory scene bitstream. Decoding starts from that bitstream and an initialized decoder model, and ends with the coordinates, mask, attributes, and network parameters required for rendering. Both intervals include context prediction, entropy coding, CPU–GPU transfers, and in-memory serialization or parsing. Checkpoint loading, scene-file input/output, decoder-model construction, rendering, and image-quality evaluation are excluded. For the QAT variant, encoding time includes exporting the quantized networks. Decoding time excludes constructing empty integer-network modules but includes restoring and transferring their parameters.

Entropy coding and component times. We use a CPU rANS backend with 16-bit probability precision. Each Gaussian-coded stream is split into four independent blocks by default and processed by up to four CPU worker threads. Coordinate occupancy and offset mask use categorical coding with transmitted CDFs. Coordinate- and attribute-coder times are measured with host timers, without synchronization at every component boundary. Context-network time is additionally measured with CUDA events. These component times are diagnostic rather than an exclusive decomposition: a host interval may wait for GPU work already counted by the event timer, and asynchronously submitted work may finish outside that host interval.

Training time. The rate–distortion training timer starts after data and model initialization and stops after the optimization loop, before the final checkpoint save and evaluation. It includes pruning, logging, periodic evaluation, and checkpoint writing. The preceding 15K Scaffold-GS initialization is timed separately.

A.4 Deterministic entropy decoding

Fixed-point representation. The integer variant uses a shared scale $S = 2^{20}$: an int32 value u represents the real value u/S. This representation is used for context-network inputs and outputs, including reconstructed variables needed for subsequent predictions. Linear layers use int8 weights and activations in $[-127, 127]$, int32 biases and accumulators, and int64 intermediates for rescaling. For integer a and positive integer d, define

$$\mathcal{R}(a, d) = \operatorname{sgn}(a) \left\lfloor \frac{|a|}{d} + \frac{1}{2} \right\rfloor . \tag{14}$$

This denotes nearest rounding with half ties away from zero. This rounding is implemented using integer division, or a rounded right shift when the denominator is a power of two. Parameter export and source-residual quantization instead use nearest rounding with half ties to even.

Coordinates and context networks. Octree decoding recovers origin-relative anchor coordinates in the same Morton order as the attribute streams. For coordinate $c_{i,d}$ and transmitted coordinate range L_d, the normalized fixed-point input is

$$u_{i,d} = \mathcal{R}(2Sc_{i,d}, L_d - 1) - S, \qquad L_d > 1. \tag{15}$$

Geometry context is computed from these integers, without first reconstructing floating-point world coordinates.

As described in Sec. 3.7, QAT uses per-tensor affine activation quantization and per-output-channel symmetric weight quantization. Observers are frozen at iteration 25,000 in the default 30K configuration. At export, the input zero-point correction is folded into the quantized bias, and scale conversions are represented by integer multipliers and shifts. Hidden linear-layer outputs are converted to the shared fixed-point representation, passed through integer GELU, and requantized to int8 for the next layer. Final network outputs remain in fixed point. The GELU implementation tabulates $h(t) = t\Phi(-t)$ from Eq. (12) over $[0, 6]$ at spacing $1/512$, storing values with 24 fractional bits. It uses integer linear interpolation and Eq. (14) to return to scale S. The lookup table is shared by the encoder and decoder, so no floating-point evaluation of Φ is needed during entropy decoding.

Probability tables. The codec includes 128 fixed integer CDF tables for discretized zero-mean Gaussians, with standard deviations logarithmically spaced from 0.1 to 256. They are generated offline and shared across scenes. In the implementation, each distribution head predicts a mean and a continuous table index representing the standard deviation. If the predicted index is stored as $v^{(q)}$ at scale S, the decoder selects

$$\ell = \operatorname{clip}\Big(\mathcal{R}(v^{(q)}, S), 0, 127\Big) . \tag{16}$$

Thus, selecting a Gaussian distribution requires only integer rounding and clipping. Coordinate and offset-mask CDFs are transmitted in the bitstream.

Causal reconstruction. For each group $g \in \{z, f, r, o, s\}$, export represents its quantization step by a multiplier ρ_g and shift τ_g such that

$$\Delta_g^{\text{int}} = \frac{\rho_g}{S2^{\tau_g}} \approx \Delta_g. \tag{17}$$

The encoder quantizes residuals using this effective step, and both sides use the transmitted pair (ρ_g, τ_g) for reconstruction. For variables needed by later predictions, the fixed-point reconstruction is

$$q_{\hat{v}} = q_\mu + \mathcal{R}(k_v \rho_g, 2^{\tau_g}), \tag{18}$$

where q_μ is the fixed-point predicted mean and k_v is the decoded residual. This implements Eq. (13) without floating-point arithmetic. In particular, previously reconstructed latent channels are used to predict the next channel, and reconstructed position scaling conditions the offset and Gaussian-scaling distributions. Position and Gaussian scaling are coded in the logarithmic domain in the implementation. All variables fed back into context prediction therefore remain in fixed point.

Features, offsets, and log-Gaussian scaling are not used for further entropy predictions. Their final values are reconstructed in floating point as

$$\hat{v} = \frac{q_\mu}{S} + k_v \Delta_g^{\text{int}}. \tag{19}$$

These final conversions do not affect entropy decoding.

Consistency scope. The scene bitstream stores the integer network parameters, rescaling parameters, reconstruction steps, coordinate metadata, mask CDF, compressed streams, and FP16 rendering-network parameters. Decoding also requires the project source code, the model configuration files, and the shared Gaussian and GELU tables. These shared resources are not counted in the scene size. For the same bitstream, consistency follows along the causal order: identical coordinates produce identical context inputs, the integer networks produce identical means and CDF indices, and the decoded residuals yield identical causal reconstructions. Parallel entropy blocks write to fixed output slices, so their scheduling does not change the decoded sequence. This guarantees consistency of entropy-decoded symbols and their fixed-point causal states, not bitwise equality of floating-point rendering. In our experiments, we verify cross-platform consistency by checking exact equality of the entropy-decoded symbols for anchor latents, anchor attributes, offset masks, and the decoded integer anchor coordinates.

A.5 Additional Rate–Distortion Results

Figure 6 presents PSNR, SSIM, and LPIPS against scene size on all four datasets. Alongside existing methods, we compare the 15+45K, 15+30K, and 15+20K settings of COSA-GS and the 15+30K quantization-aware training (QAT) variant.

Optimization steps. Increasing the number of optimization steps generally improves the PSNR–size trade-off on Mip-NeRF360, DeepBlending, and Tanks&Temples. On Tanks&Temples, the 15+45K setting also improves SSIM and LPIPS over much of the bitrate range. On Mip-NeRF360 and DeepBlending, the SSIM–size and LPIPS–size curves remain close across the three training settings. On BungeeNeRF, the 15+20K and 15+45K settings yield similar rate–distortion performance across all three metrics.

Quantization-aware training. The 15+30K QAT variant combines quantization-aware training with integer inference for the context model. Its rate–distortion curves remain close to those of the floating-point 15+30K model across most bitrate points, with small quality losses at some low-bitrate points on DeepBlending and BungeeNeRF.

A.6 Additional Ablation Studies

Figure 7 extends the main-paper ablations to all three quality metrics under the 15+30K setting. The first row examines context sources and position scaling, while the second row examines the numbers of context, feature, and latent channels.

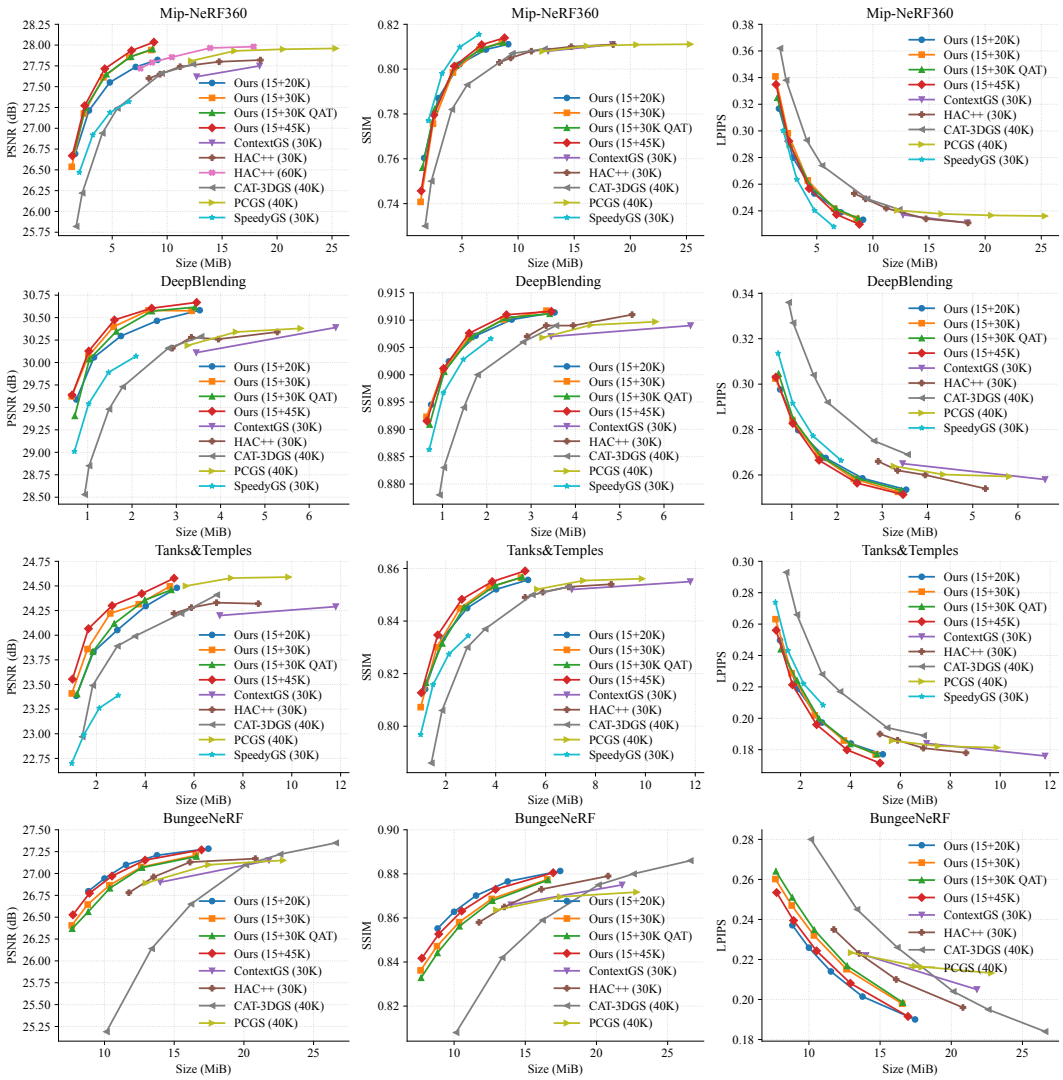

Figure 6: Rate–distortion comparison of COSA-GS and existing methods. Rows show Mip-NeRF360, DeepBlending, Tanks&Temples, and BungeeNeRF, respectively. Columns show PSNR, SSIM, and LPIPS against scene size.

Context construction and position scaling. Removing both geometry context and anchor latents produces the clearest degradation at low bitrate points, with lower PSNR and SSIM and higher LPIPS. Retaining either context source recovers much of this loss, showing that each provides informative context. Combining the two sources yields a favorable overall rate–distortion trade-off. Replacing scalar position scaling with a three-dimensional vector provides no consistent improvement across the three metrics.

Model capacity. Increasing the number of context channels from 24 to 32 has little effect on rate–distortion performance across the three metrics. Increasing the number of feature channels from 32 to 50 slightly improves PSNR at the highest bitrate point but degrades performance across all three metrics at the lowest bitrate point. Using one or eight latent channels yields rate–distortion curves close to those of the default four-channel setting. These results show no consistent benefit from increasing the number of channels, supporting the default configuration.

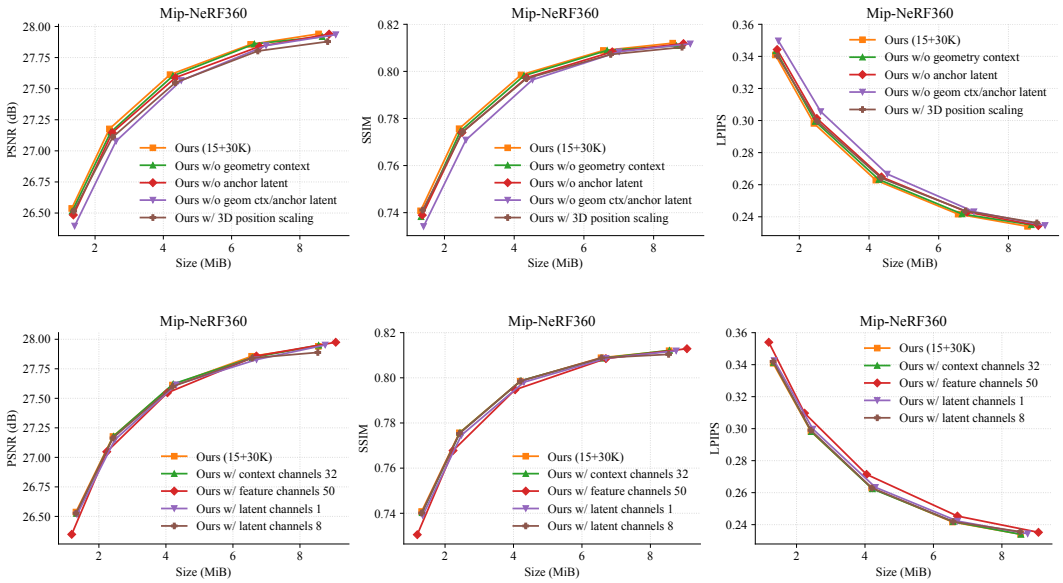

Figure 7: Additional ablation results on Mip-NeRF360 under the 15+30K setting. Top: removing geometry context, anchor latents, or both, and replacing scalar position scaling with a three-dimensional vector. Bottom: varying the numbers of context, feature, and latent channels. Columns show PSNR, SSIM, and LPIPS against scene size.

A.7 Detailed Results and Coding Times

Tables 3, 4, 5, and 6 report dataset-average reconstruction quality, compressed scene size, and total encoding and decoding times for the 15+45K, 15+30K, 15+30K QAT, and 15+20K settings, respectively. Detailed per-scene results are provided in Tables 7–10 for 15+45K, Tables 11–14 for 15+30K, Tables 15–18 for 15+30K QAT, and Tables 19–22 for 15+20K. Within each group, the tables report Mip-NeRF360, DeepBlending, Tanks&Temples, and BungeeNeRF results in that order. Runtime measurements use the reference platform with an NVIDIA 5880 Ada GPU and an AMD EPYC 9654 CPU. Table 23 lists the software versions used on this platform. Readers can consult the anonymous implementation for the configurations and evaluation procedures used in these experiments.

Detailed bitrate points. For each dataset, the 15+30K setting produces smaller bitstreams than the 15+20K setting at every tested rate weight λ. Extending the number of rate–distortion optimization steps from 30K to 45K increases PSNR at every listed λ, with a small increase in scene size. On BungeeNeRF, the 15+20K setting produces larger bitstreams and higher rendering quality than the 15+30K setting at the same λ, while the corresponding rate–distortion curves remain close.

Coding time. Across the three floating-point settings, dataset-average encoding times range from 0.0240 to 0.1752 s, and decoding times range from 0.0264 to 0.1839 s. The dataset-average encoding and decoding times remain below 0.24 s at every tabulated bitrate point.

Table 3: Dataset-average reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+45K setting. PSNR is measured in dB.

Dataset	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
Mip-NeRF360	0.0006	28.0353	0.8140	0.2299	8.7942	0.0925	0.1144
	0.0010	27.9336	0.8109	0.2373	6.7579	0.0845	0.1061
	0.0020	27.7153	0.8013	0.2567	4.3219	0.0652	0.0833
	0.0040	27.2716	0.7796	0.2924	2.5008	0.0493	0.0657
	0.0080	26.6699	0.7457	0.3348	1.3683	0.0375	0.0462
DeepBlending	0.0010	30.6693	0.9116	0.2514	3.4592	0.0451	0.0605
	0.0020	30.6049	0.9110	0.2564	2.4439	0.0425	0.0525
	0.0040	30.4742	0.9076	0.2665	1.6032	0.0394	0.0454
	0.0080	30.1275	0.9011	0.2827	1.0234	0.0295	0.0346
	0.0160	29.6381	0.8915	0.3031	0.6472	0.0240	0.0264
Tanks&Temples	0.0010	24.5780	0.8591	0.1715	5.1807	0.0603	0.0791
	0.0020	24.4219	0.8550	0.1799	3.8597	0.0499	0.0722
	0.0040	24.2998	0.8483	0.1959	2.6422	0.0449	0.0574
	0.0080	24.0663	0.8347	0.2213	1.6750	0.0353	0.0468
	0.0160	23.5543	0.8127	0.2561	1.0202	0.0313	0.0394
BungeeNeRF	0.0002	27.2707	0.8805	0.1915	16.9620	0.1626	0.1775
	0.0004	27.1533	0.8730	0.2081	12.9142	0.1445	0.1561
	0.0006	26.9704	0.8631	0.2243	10.5382	0.1223	0.1480
	0.0008	26.7743	0.8527	0.2395	8.9226	0.0980	0.1224
	0.0010	26.5272	0.8417	0.2534	7.7347	0.0927	0.1151

Table 4: Dataset-average reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K setting. PSNR is measured in dB.

Dataset	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
Mip-NeRF360	0.0006	27.9403	0.8120	0.2340	8.5471	0.0895	0.1118
	0.0010	27.8560	0.8089	0.2416	6.5575	0.0801	0.1012
	0.0020	27.6126	0.7985	0.2628	4.1960	0.0636	0.0813
	0.0040	27.1765	0.7757	0.2982	2.4203	0.0480	0.0619
	0.0080	26.5370	0.7408	0.3409	1.3158	0.0356	0.0442
DeepBlending	0.0010	30.5756	0.9117	0.2525	3.3463	0.0448	0.0595
	0.0020	30.5802	0.9101	0.2583	2.3720	0.0426	0.0553
	0.0040	30.3981	0.9067	0.2689	1.5820	0.0337	0.0411
	0.0080	30.0710	0.9008	0.2841	1.0146	0.0280	0.0326
	0.0160	29.6241	0.8923	0.3025	0.6383	0.0243	0.0308
Tanks&Temples	0.0010	24.4956	0.8565	0.1769	5.0108	0.0582	0.0815
	0.0020	24.3150	0.8528	0.1858	3.7345	0.0512	0.0698
	0.0040	24.2215	0.8448	0.2019	2.5687	0.0437	0.0561
	0.0080	23.8591	0.8300	0.2287	1.6378	0.0349	0.0438
	0.0160	23.4090	0.8072	0.2631	0.9933	0.0305	0.0331
BungeeNeRF	0.0002	27.2069	0.8775	0.1979	16.5453	0.1597	0.1747
	0.0004	27.0739	0.8686	0.2151	12.6449	0.1425	0.1500
	0.0006	26.8656	0.8581	0.2320	10.3558	0.1263	0.1317
	0.0008	26.6451	0.8472	0.2470	8.7959	0.0986	0.1182
	0.0010	26.4055	0.8362	0.2602	7.6483	0.0968	0.1111

Table 5: Dataset-average reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K QAT setting. PSNR is measured in dB.

Dataset	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
Mip-NeRF360	0.0006	27.9519	0.8119	0.2344	8.6746	0.1068	0.1206
	0.0010	27.8611	0.8089	0.2417	6.7016	0.1013	0.1099
	0.0020	27.6498	0.8016	0.2579	4.4722	0.0894	0.0958
	0.0040	27.2489	0.7822	0.2900	2.5831	0.0713	0.0762
	0.0080	26.6939	0.7560	0.3249	1.4831	0.0609	0.0607
DeepBlending	0.0010	30.6162	0.9112	0.2532	3.4205	0.0624	0.0651
	0.0020	30.5719	0.9104	0.2585	2.4474	0.0553	0.0555
	0.0040	30.3461	0.9069	0.2681	1.6462	0.0519	0.0477
	0.0080	30.0369	0.9005	0.2843	1.0378	0.0457	0.0395
	0.0160	29.4052	0.8909	0.3046	0.7079	0.0425	0.0355
Tanks&Temples	0.0010	24.4596	0.8567	0.1772	5.0638	0.0813	0.0840
	0.0020	24.3562	0.8536	0.1836	3.9832	0.0835	0.0824
	0.0040	24.1183	0.8452	0.1999	2.7325	0.0693	0.0657
	0.0080	23.8336	0.8315	0.2240	1.8563	0.0618	0.0546
	0.0160	23.4018	0.8166	0.2440	1.2088	0.0587	0.0499
BungeeNeRF	0.0002	27.1949	0.8772	0.1985	16.5945	0.1876	0.2007
	0.0004	27.0668	0.8678	0.2168	12.6852	0.1612	0.1699
	0.0006	26.8308	0.8562	0.2348	10.3753	0.1392	0.1483
	0.0008	26.5614	0.8441	0.2510	8.8307	0.1243	0.1350
	0.0010	26.3690	0.8328	0.2640	7.6827	0.1043	0.1164

Table 6: Dataset-average reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+20K setting. PSNR is measured in dB.

Dataset	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
Mip-NeRF360	0.0006	27.8221	0.8111	0.2333	9.1255	0.1024	0.1176
	0.0010	27.7373	0.8088	0.2388	7.1409	0.0884	0.1131
	0.0020	27.5516	0.8022	0.2530	4.7852	0.0728	0.0940
	0.0040	27.2134	0.7870	0.2799	2.8861	0.0577	0.0717
	0.0080	26.6940	0.7604	0.3167	1.6292	0.0435	0.0542
DeepBlending	0.0010	30.5821	0.9114	0.2535	3.5288	0.0461	0.0616
	0.0020	30.4643	0.9101	0.2585	2.5626	0.0423	0.0550
	0.0040	30.2951	0.9071	0.2675	1.7497	0.0357	0.0458
	0.0080	30.0578	0.9024	0.2797	1.1424	0.0342	0.0382
	0.0160	29.5879	0.8945	0.2976	0.7423	0.0264	0.0309
Tanks&Temples	0.0010	24.4810	0.8557	0.1770	5.2962	0.0698	0.0821
	0.0020	24.2953	0.8521	0.1839	4.0183	0.0531	0.0773
	0.0040	24.0520	0.8449	0.1973	2.8573	0.0498	0.0610
	0.0080	23.8300	0.8335	0.2188	1.8784	0.0397	0.0516
	0.0160	23.3851	0.8141	0.2497	1.1736	0.0317	0.0381
BungeeNeRF	0.0002	27.2837	0.8813	0.1900	17.4554	0.1752	0.1839
	0.0004	27.2082	0.8765	0.2015	13.7652	0.1532	0.1696
	0.0006	27.0993	0.8701	0.2140	11.5439	0.1410	0.1540
	0.0008	26.9425	0.8628	0.2259	9.9990	0.1226	0.1346
	0.0010	26.7970	0.8552	0.2371	8.8402	0.1098	0.1343

Table 7: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+45K setting on Mip-NeRF360. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
bicycle	0.0006	25.2835	0.7478	0.2630	12.0663	0.1225	0.1521
bonsai		33.2550	0.9499	0.1793	5.3596	0.0730	0.0851
counter		30.1227	0.9201	0.1821	5.9570	0.0600	0.0804
flowers		21.4197	0.5845	0.3740	10.2681	0.1022	0.1288
garden		27.4425	0.8475	0.1485	12.0146	0.1177	0.1481
kitchen		32.1876	0.9330	0.1196	6.8020	0.0732	0.0881
room		32.6127	0.9315	0.1897	4.5917	0.0527	0.0732
stump		26.7639	0.7668	0.2622	9.8609	0.1029	0.1239
treehill		23.2300	0.6445	0.3508	12.2276	0.1285	0.1496
bicycle	0.0010	25.2325	0.7435	0.2736	9.1301	0.1160	0.1732
bonsai		33.1168	0.9481	0.1831	4.1727	0.0620	0.0806
counter		29.9507	0.9163	0.1901	4.8722	0.0531	0.0730
flowers		21.3906	0.5809	0.3804	7.6977	0.0887	0.1148
garden		27.2319	0.8383	0.1669	8.8459	0.0972	0.1209
kitchen		32.0005	0.9307	0.1236	5.3657	0.0791	0.0823
room		32.5008	0.9295	0.1951	3.6470	0.0475	0.0656
stump		26.7780	0.7671	0.2658	7.8484	0.1101	0.1133
treehill		23.2004	0.6437	0.3568	9.2416	0.1066	0.1309
bicycle	0.0020	25.0605	0.7246	0.3047	5.5231	0.0939	0.1085
bonsai		32.6956	0.9429	0.1927	2.8403	0.0523	0.0658
counter		29.8797	0.9117	0.1996	3.4529	0.0469	0.0613
flowers		21.3864	0.5759	0.3933	4.9822	0.0692	0.0891
garden		26.6755	0.8081	0.2223	5.1080	0.0695	0.0978
kitchen		31.6032	0.9247	0.1352	3.4629	0.0559	0.0702
room		32.2424	0.9234	0.2113	2.4427	0.0421	0.0549
stump		26.7188	0.7610	0.2786	5.2909	0.0692	0.0937
treehill		23.1756	0.6395	0.3727	5.7940	0.0874	0.1084
bicycle	0.0040	24.5539	0.6813	0.3533	3.0359	0.0673	0.0838
bonsai		31.9820	0.9333	0.2108	1.8443	0.0424	0.0706
counter		29.3429	0.8968	0.2262	2.2085	0.0380	0.0480
flowers		21.2963	0.5601	0.4171	3.0681	0.0511	0.0747
garden		25.7611	0.7464	0.3083	2.5326	0.0465	0.0605
kitchen		31.0339	0.9140	0.1567	2.0040	0.0431	0.0584
room		31.7583	0.9127	0.2391	1.4926	0.0324	0.0406
stump		26.5431	0.7482	0.3108	3.1088	0.0527	0.0693
treehill		23.1733	0.6237	0.4088	3.2123	0.0697	0.0856
bicycle	0.0080	24.0106	0.6258	0.4050	1.6033	0.0517	0.0644
bonsai		31.1515	0.9176	0.2372	1.1353	0.0345	0.0412
counter		28.7681	0.8781	0.2572	1.2999	0.0306	0.0368
flowers		21.0336	0.5309	0.4489	1.7634	0.0392	0.0499
garden		24.6643	0.6616	0.3909	1.1751	0.0300	0.0358
kitchen		30.1894	0.8967	0.1910	1.1434	0.0333	0.0401
room		31.3093	0.8999	0.2666	0.8865	0.0260	0.0296
stump		25.9752	0.7087	0.3624	1.6359	0.0396	0.0514
treehill		22.9273	0.5919	0.4542	1.6716	0.0524	0.0671

Table 8: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+45K setting on DeepBlending. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
drjohnson	0.0010	30.0368	0.9088	0.2509	3.8020	0.0475	0.0641
playroom		31.3019	0.9144	0.2519	3.1163	0.0427	0.0568
drjohnson	0.0020	30.0120	0.9087	0.2544	2.6861	0.0489	0.0579
playroom		31.1977	0.9133	0.2584	2.2017	0.0362	0.0471
drjohnson	0.0040	29.8398	0.9043	0.2659	1.7732	0.0394	0.0514
playroom		31.1085	0.9109	0.2671	1.4331	0.0395	0.0395
drjohnson	0.0080	29.4591	0.8959	0.2846	1.1316	0.0309	0.0378
playroom		30.7960	0.9064	0.2808	0.9152	0.0282	0.0314
drjohnson	0.0160	28.9196	0.8841	0.3059	0.7131	0.0257	0.0284
playroom		30.3566	0.8989	0.3002	0.5814	0.0224	0.0244

Table 9: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+45K setting on Tanks&Temples. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
train	0.0010	22.9447	0.8288	0.2014	5.2069	0.0588	0.0759
truck		26.2113	0.8893	0.1415	5.1546	0.0618	0.0823
train	0.0020	22.7386	0.8246	0.2082	4.0864	0.0472	0.0708
truck		26.1053	0.8854	0.1516	3.6329	0.0525	0.0737
train	0.0040	22.7198	0.8194	0.2206	2.9058	0.0443	0.0590
truck		25.8799	0.8771	0.1713	2.3787	0.0455	0.0558
train	0.0080	22.5913	0.8067	0.2421	1.8879	0.0376	0.0469
truck		25.5414	0.8626	0.2005	1.4621	0.0329	0.0467
train	0.0160	22.1488	0.7839	0.2745	1.1577	0.0314	0.0439
truck		24.9598	0.8415	0.2377	0.8828	0.0311	0.0348

Table 10: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+45K setting on BungeeNeRF. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
amsterdam	0.0002	27.4387	0.9010	0.1643	18.2302	0.1678	0.2007
bilbao		27.8780	0.8871	0.1782	16.9360	0.1592	0.1693
hollywood		24.8905	0.7974	0.2851	17.0617	0.1647	0.1800
pompidou		25.8096	0.8631	0.2114	17.6858	0.1620	0.1643
quebec		30.6954	0.9450	0.1333	14.3947	0.1385	0.1504
rome		26.9121	0.8895	0.1770	17.4639	0.1832	0.2005
amsterdam	0.0004	27.2421	0.8889	0.1899	13.5120	0.1410	0.1621
bilbao		27.7605	0.8766	0.2025	12.8203	0.1406	0.1626
hollywood		24.8016	0.7900	0.2995	13.5226	0.1394	0.1611
pompidou		25.7204	0.8563	0.2253	13.2813	0.1643	0.1500
quebec		30.5626	0.9426	0.1413	11.1445	0.1223	0.1484
rome		26.8327	0.8834	0.1901	13.2042	0.1598	0.1526
amsterdam	0.0006	27.0096	0.8741	0.2151	10.8353	0.1348	0.1480
bilbao		27.5435	0.8626	0.2248	10.3477	0.1199	0.1300
hollywood		24.7597	0.7820	0.3115	11.2703	0.1174	0.1302
pompidou		25.5063	0.8465	0.2400	10.8017	0.1275	0.1497
quebec		30.4673	0.9399	0.1488	9.2429	0.0957	0.1327
rome		26.5357	0.8735	0.2055	10.7311	0.1388	0.1976
amsterdam	0.0008	26.7508	0.8588	0.2370	9.0223	0.1022	0.1307
bilbao		27.3364	0.8510	0.2416	8.6649	0.0995	0.1146
hollywood		24.6343	0.7711	0.3256	9.6504	0.0914	0.1302
pompidou		25.1719	0.8339	0.2556	9.1466	0.0987	0.1313
quebec		30.3722	0.9373	0.1564	7.9632	0.0838	0.1059
rome		26.3801	0.8639	0.2205	9.0883	0.1123	0.1214
amsterdam	0.0010	26.4696	0.8426	0.2577	7.7320	0.1017	0.1194
bilbao		27.0632	0.8405	0.2552	7.4388	0.1043	0.1202
hollywood		24.4624	0.7586	0.3401	8.4083	0.0896	0.1093
pompidou		24.9057	0.8216	0.2679	7.9541	0.0832	0.1082
quebec		30.1583	0.9331	0.1651	6.9761	0.0852	0.1102
rome		26.1039	0.8538	0.2346	7.8988	0.0919	0.1230

Table 11: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K setting on Mip-NeRF360. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
bicycle	0.0006	25.1977	0.7457	0.2677	11.6525	0.1280	0.1498
bonsai		33.1701	0.9493	0.1813	5.2926	0.0691	0.0865
counter		29.9499	0.9185	0.1851	5.8135	0.0579	0.0781
flowers		21.4071	0.5816	0.3784	9.9852	0.1009	0.1279
garden		27.3482	0.8442	0.1551	11.6989	0.1169	0.1352
kitchen		32.0032	0.9306	0.1225	6.6229	0.0757	0.0878
room		32.4597	0.9301	0.1934	4.4935	0.0525	0.0717
stump		26.7547	0.7652	0.2668	9.5308	0.0905	0.1272
treehill		23.1717	0.6428	0.3560	11.8340	0.1144	0.1422
bicycle	0.0010	25.1798	0.7404	0.2795	8.6910	0.1144	0.1293
bonsai		33.0185	0.9472	0.1854	4.1009	0.0597	0.0766
counter		29.8207	0.9158	0.1908	4.7608	0.0531	0.0940
flowers		21.3845	0.5791	0.3847	7.4766	0.0872	0.1108
garden		27.1379	0.8340	0.1758	8.6039	0.1004	0.1174
kitchen		31.8189	0.9284	0.1268	5.2713	0.0705	0.0825
room		32.3367	0.9280	0.1990	3.5764	0.0488	0.0643
stump		26.7654	0.7650	0.2706	7.5947	0.0837	0.1082
treehill		23.2418	0.6420	0.3623	8.9419	0.1026	0.1273
bicycle	0.0020	24.9983	0.7206	0.3118	5.3526	0.0848	0.1045
bonsai		32.5349	0.9418	0.1959	2.7935	0.0511	0.0659
counter		29.6267	0.9091	0.2042	3.3737	0.0471	0.0608
flowers		21.3542	0.5714	0.3999	4.8043	0.0657	0.0928
garden		26.5740	0.8022	0.2339	4.9823	0.0727	0.0878
kitchen		31.4120	0.9216	0.1395	3.3833	0.0555	0.0692
room		32.1010	0.9219	0.2156	2.3999	0.0414	0.0546
stump		26.7142	0.7614	0.2841	5.1209	0.0681	0.0915
treehill		23.1985	0.6365	0.3807	5.5531	0.0856	0.1049
bicycle	0.0040	24.5236	0.6766	0.3611	2.8992	0.0641	0.0801
bonsai		31.8558	0.9315	0.2146	1.8110	0.0413	0.0519
counter		29.2352	0.8957	0.2284	2.1616	0.0372	0.0471
flowers		21.2158	0.5535	0.4240	2.9386	0.0492	0.0648
garden		25.6422	0.7385	0.3186	2.4537	0.0456	0.0673
kitchen		30.8350	0.9111	0.1611	1.9582	0.0423	0.0547
room		31.6762	0.9111	0.2426	1.4816	0.0318	0.0392
stump		26.5135	0.7436	0.3174	3.0097	0.0534	0.0683
treehill		23.0915	0.6193	0.4164	3.0693	0.0672	0.0833
bicycle	0.0080	23.8629	0.6172	0.4140	1.4905	0.0469	0.0600
bonsai		30.9042	0.9145	0.2424	1.1146	0.0323	0.0395
counter		28.5802	0.8752	0.2620	1.2757	0.0285	0.0337
flowers		20.9496	0.5234	0.4564	1.6807	0.0369	0.0459
garden		24.5411	0.6536	0.3978	1.1307	0.0300	0.0345
kitchen		30.0107	0.8938	0.1951	1.1223	0.0328	0.0383
room		31.1372	0.8982	0.2697	0.8780	0.0258	0.0288
stump		25.9571	0.7038	0.3691	1.5647	0.0375	0.0536
treehill		22.8900	0.5873	0.4613	1.5848	0.0498	0.0635

Table 12: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K setting on DeepBlending. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
drjohnson	0.0010	29.9732	0.9093	0.2519	3.6608	0.0482	0.0638
playroom		31.1780	0.9142	0.2532	3.0318	0.0414	0.0553
drjohnson	0.0020	29.9998	0.9077	0.2574	2.6012	0.0463	0.0545
playroom		31.1607	0.9125	0.2591	2.1429	0.0389	0.0561
drjohnson	0.0040	29.8081	0.9040	0.2671	1.7314	0.0347	0.0444
playroom		30.9881	0.9094	0.2708	1.4326	0.0327	0.0378
drjohnson	0.0080	29.4251	0.8962	0.2856	1.1180	0.0303	0.0351
playroom		30.7169	0.9054	0.2827	0.9112	0.0257	0.0301
drjohnson	0.0160	28.9561	0.8864	0.3045	0.6970	0.0262	0.0343
playroom		30.2921	0.8982	0.3005	0.5795	0.0224	0.0272

Table 13: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K setting on Tanks&Temples. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
train	0.0010	22.8962	0.8260	0.2069	4.9931	0.0581	0.0723
truck		26.0951	0.8871	0.1469	5.0286	0.0582	0.0907
train	0.0020	22.6636	0.8229	0.2137	3.9408	0.0481	0.0696
truck		25.9664	0.8827	0.1580	3.5281	0.0542	0.0701
train	0.0040	22.7186	0.8160	0.2267	2.8295	0.0438	0.0569
truck		25.7244	0.8736	0.1770	2.3080	0.0436	0.0552
train	0.0080	22.3242	0.8011	0.2500	1.8518	0.0353	0.0450
truck		25.3941	0.8589	0.2074	1.4238	0.0346	0.0426
train	0.0160	21.9496	0.7782	0.2809	1.1275	0.0296	0.0348
truck		24.8685	0.8363	0.2454	0.8591	0.0313	0.0313

Table 14: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K setting on BungeeNeRF. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
amsterdam	0.0002	27.2770	0.8953	0.1749	17.7631	0.1830	0.1868
bilbao		27.8415	0.8852	0.1827	16.5058	0.1532	0.1831
hollywood		24.8069	0.7917	0.2969	16.5321	0.1488	0.1635
pompidou		25.6741	0.8595	0.2166	17.2445	0.1768	0.1688
quebec		30.7004	0.9445	0.1355	14.1288	0.1334	0.1486
rome		26.9417	0.8884	0.1809	17.0978	0.1628	0.1975
amsterdam	0.0004	27.0565	0.8819	0.2014	13.2343	0.1689	0.1572
bilbao		27.7092	0.8728	0.2095	12.5271	0.1567	0.1611
hollywood		24.7570	0.7824	0.3096	13.0967	0.1328	0.1384
pompidou		25.5832	0.8524	0.2307	13.0112	0.1349	0.1501
quebec		30.5751	0.9417	0.1443	10.9900	0.1202	0.1450
rome		26.7623	0.8807	0.1954	13.0101	0.1412	0.1485
amsterdam	0.0006	26.7930	0.8660	0.2265	10.6481	0.1243	0.1434
bilbao		27.4957	0.8594	0.2315	10.1682	0.1203	0.1258
hollywood		24.6408	0.7731	0.3237	10.9535	0.1191	0.1342
pompidou		25.3676	0.8415	0.2461	10.6087	0.1457	0.1322
quebec		30.3562	0.9381	0.1531	9.1527	0.1242	0.1230
rome		26.5402	0.8703	0.2112	10.6036	0.1239	0.1319
amsterdam	0.0008	26.5223	0.8492	0.2484	8.9210	0.1096	0.1193
bilbao		27.2346	0.8469	0.2487	8.5423	0.0930	0.1163
hollywood		24.5018	0.7618	0.3378	9.4184	0.1023	0.1252
pompidou		25.0463	0.8299	0.2595	9.0049	0.0976	0.1160
quebec		30.2854	0.9352	0.1614	7.8817	0.0795	0.1108
rome		26.2800	0.8599	0.2261	9.0068	0.1099	0.1218
amsterdam	0.0010	26.2470	0.8335	0.2675	7.6850	0.0988	0.1225
bilbao		27.0009	0.8359	0.2607	7.3670	0.1041	0.1049
hollywood		24.3468	0.7488	0.3510	8.2345	0.0970	0.1074
pompidou		24.7458	0.8186	0.2724	7.8418	0.0977	0.1049
quebec		30.0668	0.9311	0.1700	6.9370	0.0844	0.1069
rome		26.0255	0.8492	0.2396	7.8243	0.0989	0.1198

Table 15: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K QAT setting on Mip-NeRF360. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
bicycle	0.0006	25.2507	0.7457	0.2681	11.9511	0.1469	0.1661
bonsai		33.1603	0.9491	0.1814	5.4509	0.0845	0.0908
counter		29.9500	0.9183	0.1854	5.8636	0.0770	0.0864
flowers		21.3977	0.5812	0.3790	10.0742	0.1139	0.1361
garden		27.3605	0.8438	0.1561	11.7687	0.1410	0.1487
kitchen		31.9857	0.9307	0.1227	6.7252	0.0857	0.0986
room		32.4843	0.9303	0.1936	4.5560	0.0682	0.0780
stump		26.7487	0.7652	0.2669	9.6638	0.1113	0.1265
treehill		23.2290	0.6427	0.3566	12.0185	0.1323	0.1547
bicycle	0.0010	25.1912	0.7414	0.2782	9.0373	0.1434	0.1516
bonsai		33.0202	0.9471	0.1851	4.2498	0.0782	0.0879
counter		29.8277	0.9156	0.1910	4.8076	0.0707	0.0797
flowers		21.4083	0.5802	0.3834	7.7443	0.1167	0.1241
garden		27.1281	0.8329	0.1780	8.6199	0.1194	0.1277
kitchen		31.8269	0.9284	0.1269	5.4207	0.0803	0.0874
room		32.3404	0.9279	0.1997	3.6750	0.0645	0.0696
stump		26.7697	0.7653	0.2709	7.6331	0.1079	0.1146
treehill		23.2374	0.6417	0.3620	9.1268	0.1308	0.1468
bicycle	0.0020	25.0996	0.7349	0.2895	6.3999	0.1504	0.1545
bonsai		32.5312	0.9417	0.1970	2.8011	0.0687	0.0762
counter		29.6193	0.9082	0.2060	3.4045	0.0602	0.0641
flowers		21.3893	0.5755	0.3920	5.2168	0.1047	0.1121
garden		26.7629	0.8111	0.2183	5.5386	0.0974	0.1074
kitchen		31.4370	0.9230	0.1378	3.6384	0.0724	0.0772
room		32.1163	0.9212	0.2176	2.4695	0.0557	0.0559
stump		26.7178	0.7614	0.2847	5.0807	0.0815	0.0933
treehill		23.1746	0.6377	0.3786	5.7007	0.1136	0.1210
bicycle	0.0040	24.6703	0.6887	0.3517	2.9253	0.0882	0.0980
bonsai		31.9827	0.9345	0.2104	1.8782	0.0642	0.0673
counter		28.9959	0.8922	0.2314	2.3324	0.0549	0.0541
flowers		21.2589	0.5581	0.4180	3.0512	0.0760	0.0837
garden		26.0143	0.7646	0.2886	2.8767	0.0729	0.0818
kitchen		30.8882	0.9144	0.1542	2.2667	0.0640	0.0669
room		31.6801	0.9110	0.2429	1.5440	0.0498	0.0459
stump		26.6361	0.7496	0.3107	3.0575	0.0734	0.0810
treehill		23.1136	0.6267	0.4021	3.3155	0.0984	0.1073
bicycle	0.0080	24.2712	0.6677	0.3687	1.9749	0.0911	0.1009
bonsai		30.9077	0.9141	0.2434	1.1040	0.0503	0.0443
counter		28.4934	0.8697	0.2698	1.2894	0.0450	0.0378
flowers		20.9827	0.5290	0.4494	1.7362	0.0628	0.0629
garden		25.1733	0.7046	0.3555	1.4526	0.0563	0.0551
kitchen		30.2317	0.9014	0.1781	1.3275	0.0556	0.0544
room		31.1220	0.9021	0.2601	1.0212	0.0456	0.0408
stump		26.1180	0.7144	0.3589	1.5758	0.0583	0.0574
treehill		22.9447	0.6012	0.4397	1.8660	0.0836	0.0932

Table 16: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K QAT setting on DeepBlending. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
drjohnson	0.0010	29.9794	0.9083	0.2535	3.7563	0.0651	0.0693
playroom		31.2529	0.9140	0.2530	3.0848	0.0597	0.0610
drjohnson	0.0020	29.9069	0.9077	0.2580	2.6702	0.0576	0.0592
playroom		31.2368	0.9131	0.2591	2.2246	0.0529	0.0518
drjohnson	0.0040	29.7257	0.9040	0.2668	1.8523	0.0536	0.0522
playroom		30.9665	0.9099	0.2694	1.4402	0.0502	0.0431
drjohnson	0.0080	29.4492	0.8966	0.2849	1.1217	0.0460	0.0401
playroom		30.6247	0.9045	0.2838	0.9539	0.0454	0.0390
drjohnson	0.0160	28.7005	0.8844	0.3080	0.7685	0.0434	0.0380
playroom		30.1098	0.8974	0.3013	0.6472	0.0415	0.0330

Table 17: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K QAT setting on Tanks&Temples. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
train	0.0010	22.8444	0.8267	0.2069	5.0423	0.0754	0.0799
truck		26.0748	0.8867	0.1475	5.0854	0.0873	0.0882
train	0.0020	22.7016	0.8231	0.2138	4.0165	0.0739	0.0716
truck		26.0107	0.8840	0.1534	3.9499	0.0931	0.0932
train	0.0040	22.4999	0.8148	0.2286	2.8787	0.0633	0.0615
truck		25.7366	0.8757	0.1712	2.5863	0.0754	0.0698
train	0.0080	22.2251	0.7986	0.2544	2.0732	0.0601	0.0502
truck		25.4420	0.8644	0.1936	1.6395	0.0636	0.0589
train	0.0160	21.9164	0.7840	0.2691	1.2876	0.0559	0.0467
truck		24.8872	0.8492	0.2189	1.1300	0.0616	0.0531

Table 18: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+30K QAT setting on BungeeNeRF. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
amsterdam	0.0002	27.2753	0.8947	0.1763	17.8597	0.2041	0.2331
bilbao		27.8726	0.8854	0.1836	16.5345	0.1899	0.2020
hollywood		24.7648	0.7916	0.2983	16.5427	0.1871	0.1848
pompidou		25.6674	0.8592	0.2163	17.2884	0.2035	0.2068
quebec		30.6520	0.9441	0.1357	14.1433	0.1546	0.1810
rome		26.9374	0.8882	0.1809	17.1984	0.1861	0.1967
amsterdam	0.0004	27.0388	0.8802	0.2038	13.2845	0.1747	0.1726
bilbao		27.7211	0.8711	0.2127	12.5554	0.1674	0.1758
hollywood		24.7720	0.7831	0.3102	13.1135	0.1521	0.1821
pompidou		25.5549	0.8511	0.2319	13.0397	0.1586	0.1808
quebec		30.5611	0.9417	0.1448	11.0298	0.1329	0.1418
rome		26.7528	0.8796	0.1973	13.0882	0.1819	0.1659
amsterdam	0.0006	26.7861	0.8641	0.2297	10.6702	0.1434	0.1612
bilbao		27.4146	0.8569	0.2350	10.1692	0.1486	0.1498
hollywood		24.5272	0.7703	0.3288	10.9451	0.1592	0.1548
pompidou		25.2770	0.8386	0.2483	10.6352	0.1367	0.1546
quebec		30.4669	0.9386	0.1536	9.1600	0.1145	0.1260
rome		26.5130	0.8689	0.2134	10.6721	0.1325	0.1433
amsterdam	0.0008	26.4677	0.8456	0.2534	8.9524	0.1218	0.1378
bilbao		27.1479	0.8419	0.2539	8.5982	0.1196	0.1373
hollywood		24.3167	0.7578	0.3440	9.4598	0.1256	0.1326
pompidou		24.9986	0.8273	0.2631	9.0401	0.1379	0.1399
quebec		30.1874	0.9339	0.1634	7.9053	0.1103	0.1113
rome		26.2499	0.8582	0.2281	9.0283	0.1307	0.1507
amsterdam	0.0010	26.2012	0.8296	0.2721	7.6868	0.1022	0.1180
bilbao		26.9206	0.8302	0.2674	7.3979	0.1117	0.1106
hollywood		24.3277	0.7456	0.3531	8.2094	0.1092	0.1194
pompidou		24.7373	0.8150	0.2764	7.8578	0.0964	0.1233
quebec		30.0979	0.9308	0.1712	7.0965	0.0926	0.1082
rome		25.9294	0.8457	0.2441	7.8478	0.1134	0.1189

Table 19: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+20K setting on Mip-NeRF360. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
bicycle	0.0006	25.0466	0.7421	0.2671	12.5481	0.1484	0.1565
bonsai		32.7906	0.9477	0.1823	5.6182	0.0715	0.0896
counter		29.7514	0.9173	0.1867	6.0469	0.0613	0.0812
flowers		21.3726	0.5797	0.3773	10.9682	0.1229	0.1469
garden		27.4057	0.8461	0.1495	12.7902	0.1296	0.1450
kitchen		31.8157	0.9295	0.1246	6.9196	0.0906	0.0904
room		32.2352	0.9297	0.1925	4.6814	0.0567	0.0747
stump		26.7457	0.7645	0.2661	10.0052	0.1228	0.1287
treehill		23.2356	0.6433	0.3540	12.5517	0.1177	0.1456
bicycle	0.0010	25.0039	0.7385	0.2756	9.6153	0.1242	0.1889
bonsai		32.5486	0.9460	0.1858	4.4234	0.0780	0.0820
counter		29.6522	0.9153	0.1905	5.0640	0.0555	0.0814
flowers		21.3396	0.5780	0.3812	8.3524	0.0973	0.1206
garden		27.2232	0.8393	0.1628	9.7053	0.1124	0.1371
kitchen		31.6673	0.9275	0.1282	5.6099	0.0739	0.0874
room		32.2026	0.9279	0.1969	3.8221	0.0515	0.0691
stump		26.7650	0.7646	0.2687	8.1184	0.0888	0.1156
treehill		23.2331	0.6417	0.3595	9.5571	0.1139	0.1362
bicycle	0.0020	24.8839	0.7264	0.2976	6.2107	0.0989	0.1195
bonsai		32.3037	0.9419	0.1932	3.0926	0.0560	0.0725
counter		29.5086	0.9105	0.2004	3.9077	0.0488	0.0662
flowers		21.3271	0.5736	0.3913	5.4901	0.0772	0.1230
garden		26.8446	0.8191	0.2015	6.0172	0.0851	0.1087
kitchen		31.2934	0.9225	0.1369	3.7616	0.0630	0.0777
room		31.8636	0.9231	0.2088	2.6669	0.0449	0.0578
stump		26.7331	0.7629	0.2771	5.6846	0.0849	0.0993
treehill		23.2063	0.6393	0.3702	6.2357	0.0962	0.1213
bicycle	0.0040	24.5874	0.6997	0.3340	3.6124	0.0849	0.0923
bonsai		31.7084	0.9344	0.2076	2.0495	0.0478	0.0600
counter		29.1638	0.9006	0.2187	2.4773	0.0413	0.0537
flowers		21.2497	0.5613	0.4106	3.4293	0.0592	0.0780
garden		26.0996	0.7760	0.2710	3.2302	0.0612	0.0773
kitchen		30.7970	0.9135	0.1537	2.2629	0.0503	0.0628
room		31.5552	0.9146	0.2300	1.7211	0.0368	0.0467
stump		26.6445	0.7546	0.2977	3.5801	0.0610	0.0795
treehill		23.1149	0.6287	0.3954	3.6120	0.0773	0.0947
bicycle	0.0080	24.1139	0.6546	0.3795	1.9466	0.0587	0.0741
bonsai		30.9574	0.9217	0.2296	1.3113	0.0387	0.0475
counter		28.6179	0.8841	0.2460	1.5217	0.0330	0.0408
flowers		21.0604	0.5383	0.4375	2.0246	0.0461	0.0580
garden		25.1470	0.7084	0.3489	1.5821	0.0403	0.0506
kitchen		30.1355	0.9000	0.1801	1.3223	0.0390	0.0481
room		31.0213	0.9021	0.2584	1.0443	0.0283	0.0340
stump		26.2552	0.7283	0.3370	1.9734	0.0478	0.0590
treehill		22.9379	0.6059	0.4337	1.9366	0.0596	0.0755

Table 20: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+20K setting on DeepBlending. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
drjohnson	0.0010	29.9249	0.9089	0.2528	3.8178	0.0479	0.0648
playroom		31.2393	0.9139	0.2543	3.2398	0.0442	0.0584
drjohnson	0.0020	29.8601	0.9074	0.2576	2.7912	0.0450	0.0593
playroom		31.0684	0.9128	0.2595	2.3341	0.0396	0.0507
drjohnson	0.0040	29.6666	0.9036	0.2674	1.9119	0.0372	0.0486
playroom		30.9235	0.9107	0.2676	1.5874	0.0341	0.0429
drjohnson	0.0080	29.3999	0.8984	0.2795	1.2449	0.0332	0.0411
playroom		30.7157	0.9065	0.2799	1.0398	0.0353	0.0354
drjohnson	0.0160	28.9911	0.8897	0.2988	0.8064	0.0280	0.0327
playroom		30.1847	0.8994	0.2965	0.6782	0.0248	0.0290

Table 21: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+20K setting on Tanks&Temples. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
train	0.0010	22.9435	0.8250	0.2083	5.2135	0.0646	0.0766
truck		26.0186	0.8863	0.1458	5.3790	0.0751	0.0876
train	0.0020	22.6356	0.8207	0.2141	4.1309	0.0490	0.0724
truck		25.9549	0.8834	0.1537	3.9056	0.0572	0.0823
train	0.0040	22.3971	0.8142	0.2250	3.1066	0.0474	0.0586
truck		25.7069	0.8756	0.1695	2.6080	0.0521	0.0634
train	0.0080	22.2610	0.8029	0.2443	2.0930	0.0400	0.0537
truck		25.3989	0.8642	0.1932	1.6638	0.0394	0.0495
train	0.0160	21.8626	0.7827	0.2729	1.3114	0.0328	0.0384
truck		24.9076	0.8456	0.2265	1.0359	0.0305	0.0379

Table 22: Scene-wise reconstruction quality, compressed scene size, and coding time of COSA-GS under the 15+20K setting on BungeeNeRF. PSNR is measured in dB.

Scene	λ	PSNR	SSIM	LPIPS	Size (MiB)	Total Enc. (s)	Total Dec. (s)
amsterdam	0.0002	27.4180	0.9009	0.1634	18.9424	0.1912	0.2115
bilbao		27.9942	0.8909	0.1706	17.4101	0.1664	0.1690
hollywood		24.8852	0.7978	0.2853	17.2021	0.1672	0.1867
pompidou		25.8569	0.8643	0.2100	18.3461	0.1986	0.1788
quebec		30.6481	0.9445	0.1343	14.8068	0.1484	0.1563
rome		26.8999	0.8894	0.1765	18.0249	0.1794	0.2009
amsterdam	0.0004	27.2869	0.8930	0.1811	14.6458	0.1692	0.1794
bilbao		27.9690	0.8853	0.1868	13.6581	0.1378	0.1545
hollywood		24.8806	0.7934	0.2937	14.0474	0.1410	0.1710
pompidou		25.7742	0.8603	0.2204	14.2709	0.1539	0.1829
quebec		30.5580	0.9427	0.1405	11.8415	0.1427	0.1400
rome		26.7806	0.8846	0.1864	14.1276	0.1749	0.1899
amsterdam	0.0006	27.1176	0.8831	0.1996	12.1007	0.1622	0.1517
bilbao		27.8354	0.8767	0.2053	11.3744	0.1574	0.1547
hollywood		24.7834	0.7873	0.3044	12.0253	0.1318	0.1675
pompidou		25.6808	0.8542	0.2309	11.8899	0.1484	0.1405
quebec		30.4814	0.9406	0.1471	10.0647	0.1032	0.1402
rome		26.6969	0.8789	0.1968	11.8082	0.1430	0.1694
amsterdam	0.0008	26.9124	0.8721	0.2179	10.3754	0.1279	0.1364
bilbao		27.6654	0.8676	0.2208	9.7682	0.1166	0.1437
hollywood		24.7055	0.7801	0.3142	10.5579	0.1393	0.1287
pompidou		25.4862	0.8469	0.2417	10.2658	0.1212	0.1495
quebec		30.3764	0.9383	0.1528	8.8132	0.1058	0.1151
rome		26.5095	0.8719	0.2081	10.2134	0.1246	0.1340
amsterdam	0.0010	26.7373	0.8613	0.2332	9.1042	0.1174	0.1563
bilbao		27.4647	0.8582	0.2353	8.5706	0.0933	0.1328
hollywood		24.6230	0.7724	0.3248	9.4148	0.1164	0.1323
pompidou		25.3589	0.8391	0.2515	9.0666	0.1066	0.1240
quebec		30.2548	0.9359	0.1590	7.8628	0.1084	0.1216
rome		26.3430	0.8644	0.2187	9.0224	0.1168	0.1386

Table 23: Software versions of the default platform with an NVIDIA 5880 Ada GPU and an AMD EPYC 9654 CPU.

Dependency	Version
Runtime and numerical libraries	
Python	3.10.16
PyTorch	2.9.1+cu126
Torchvision	0.24.1+cu126
NumPy	1.26.4
SciPy	1.15.3
einops	0.8.2
PyYAML	6.0.3
Pillow	11.3.0
plyfile	1.1.3
OpenCV (opencv-python)	4.11.0.86
LPIPS	0.1.4
tqdm	4.67.3
colorama	0.4.6
Compilation and CUDA extensions	
CUDA Toolkit (nvcc)	12.6.77
GCC/G++	12.3.0
setuptools	80.10.2
Ninja	1.13.0
CUTLASS*	`291300ffffa3`
GLM	0.9.9.9

*CUTLASS is identified by its Git commit.