HAC++: Towards 100X Compression of 3D Gaussian Splatting

Yihang Chen, Qianyi Wu†, Weiyao Lin†, Mehrtash Harandi, Jianfei Cai, *Fellow, IEEE*

***Abstract*—3D Gaussian Splatting (3DGS) has emerged as a promising framework for novel view synthesis, boasting rapid rendering speed with high fidelity. However, the substantial Gaussians and their associated attributes necessitate effective compression techniques. Nevertheless, the sparse and unorganized nature of the point cloud of Gaussians (or anchors in our paper) presents challenges for compression. To achieve a compact size, we propose HAC++, which leverages the relationships between unorganized anchors and a structured hash grid, utilizing their mutual information for context modeling. Additionally, HAC++ captures intra-anchor contextual relationships to further enhance compression performance. To facilitate entropy coding, we utilize Gaussian distributions to precisely estimate the probability of each quantized attribute, where an adaptive quantization module is proposed to enable high-precision quantization of these attributes for improved fidelity restoration. Moreover, we incorporate an adaptive masking strategy to eliminate invalid Gaussians and anchors. Overall, HAC++ achieves a remarkable size reduction of over $100\times$ compared to vanilla 3DGS when averaged on all datasets, while simultaneously improving fidelity. It also delivers more than $20\times$ size reduction compared to Scaffold-GS. Our code is available at https://github.com/YihangChen-ee/HAC-plus.**

***Index Terms*—3D Gaussian Splatting, Compression, Context model.**

I. Introduction

OVER the past few years, significant advancements have been made in novel view synthesis. Neural Radiance Field (NeRF) [1] introduced a groundbreaking approach for establishing 3D representations from input images by rendering colors through the accumulation of RGB values along sampling rays using an implicit Multilayer Perceptron (MLP). However, the large volume of the MLP and the extensive ray point sampling have become bottlenecks, hindering both training and rendering speeds. Advancements in NeRF [2]–[4] introduce feature grids to enhance the rendering process by involving learnable explicit representations, facilitating faster rendering speeds using swifter MLPs. Nevertheless, these methods still suffer from relatively slow rendering due to the dense ray point sampling.

In this context, a new type of 3D representation, 3D Gaussian Splatting (3DGS) [5], has recently emerged. 3DGS describes 3D scenes using learnable explicit Gaussians. These Gaussians are derived from Structure-from-Motion (SfM) [6], which reconstructs 3D structures from images and views. Enhanced with learnable shape and appearance parameters, the Gaussians can be efficiently splatted onto 2D planes using tile-based rasterization [7], enabling rapid and differentiable rendering. By eliminating the computational burden of the dense point sampling, 3DGS achieves a significant boost in rendering speed. Its advantages of fast rendering and high photo-realistic fidelity have driven its rapid adoption.

Despite these advancements, 3DGS faces the challenge of massive Gaussian primitives (*e.g.*, millions of Gaussians for city-scale scenes), resulting in large storage requirements (*e.g.*, a few GigaBytes (GB) to store the Gaussian attributes for each scene [8], [9]). This issue motivates the exploration of effective compression techniques for 3DGS.

However, compressing 3D Gaussians is inherently challenging [10]–[12] due to their sparse and unorganized nature. Various methods have been proposed to address this issue [13]–[16] by utilizing Gaussian pruning or codebook-based vector quantization, as illustrated in Fig. 1. However, they share a common limitation: they focus on parameter "values" while neglecting the redundancies inherent in structural relations. Structural relations play a crucial role in uncovering redundancies among parameters, which is a principle that has been effectively demonstrated in image compression [17], [18]. While such relations are relatively easier to identify for images, leveraging them for 3D Gaussians remains a challenge. To address this, Scaffold-GS [19] introduces anchors to cluster related 3D Gaussians and uses neural predictions to infer their attributes from attributes of the anchor, achieving substantial storage savings. Despite advancements, Scaffold-GS treats each anchor independently, leaving anchors sparse, unorganized, and difficult to compress due to their point-cloud-like nature.

We draw inspiration from the NeRF series [1], which leverage well-organized feature grids [2], [3] to represent 3D space. We pose the question: *Are there inherent relations between the attributes of unorganized anchors in Scaffold-GS and the structured feature grids?* Our answer is affirmative since we observe large mutual information between anchor attributes and the hash grid features. Based on this insight, we propose our HAC++ method. The core idea is a Hash-grid Assisted Context (HAC) framework to jointly learn a structured and compact hash grid (with binarized hash parameters), and utilize it for context modeling of anchor attributes. Specifically, with Scaffold-GS [19] as our base model, for each anchor, we query the hash grid by the anchor location to obtain an interpolated hash feature, which is then used to predict the value distributions of anchor attributes.

Yihang Chen, Qianyi Wu, Mehrtash Harandi, Jianfei Cai are with Monash University. Email: {yihang.chen, qianyi.wu, mehrtash.harandi, jianfei.cai}@monash.edu

Weiyao Lin is with Shanghai Jiao Tong University. Email: wylin@sjtu.edu.cn

Yihang Chen is also with Shanghai Jiao Tong University.

†Corresponding authors: Qianyi Wu and Weiyao Lin.

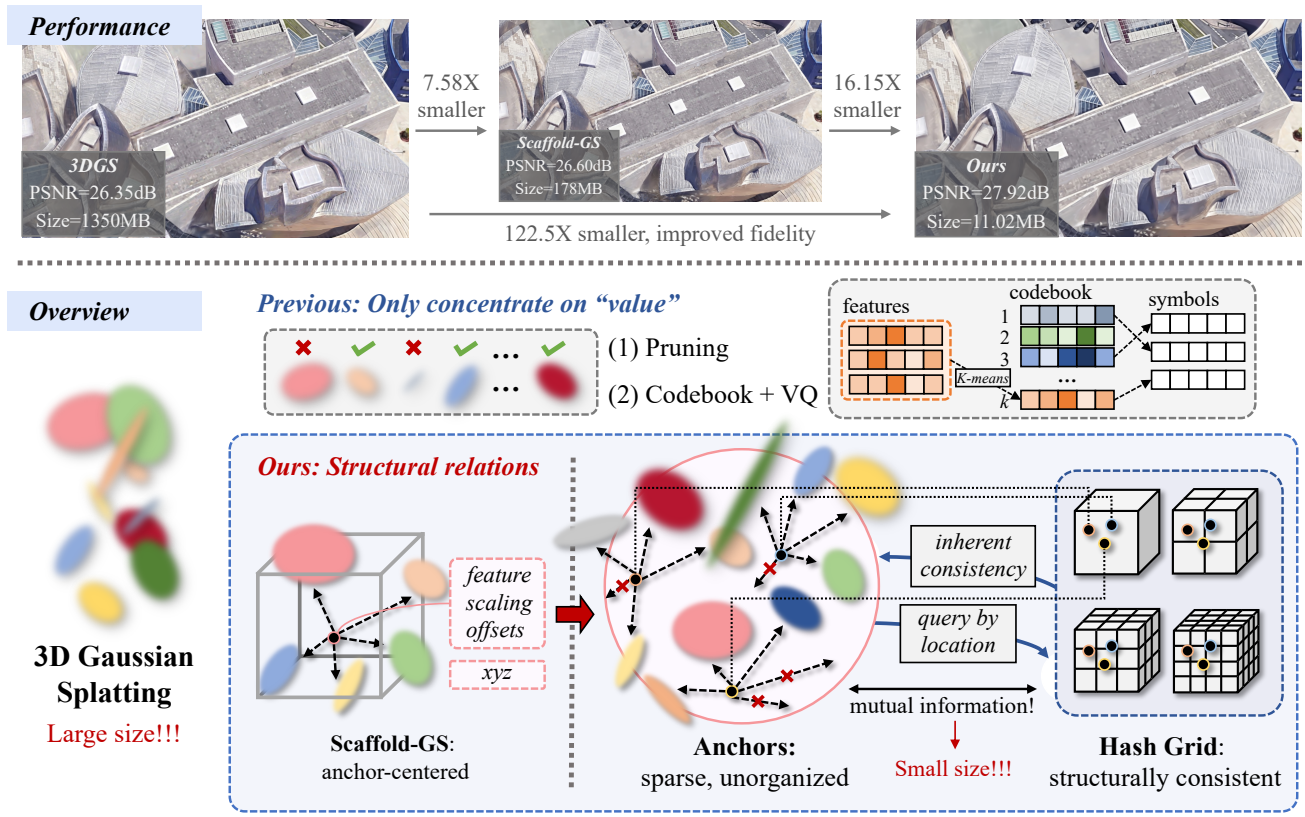

Fig. 1. **Top**: A toy example showcasing the effectiveness of our method, which reduces the size of the vanilla 3D Gaussian Splatting (3DGS) model by 122.5× (or 16.15× compared to the SoTA Scaffold-GS [19]), with similar or better fidelity. **Bottom**: Most existing 3DGS compression methods typically focus on parameter "values" through pruning or vector quantization, overlooking structural relations among Gaussians. Scaffold-GS [19] introduces anchors to cluster Gaussians and neural-predict their attributes but treats each anchor independently. In contrast, our method leverages the inherent consistencies among anchors via a structured hash grid, enabling a significantly more compressed 3DGS representation.

Beyond HAC modeling, we integrate an intra-anchor context model to effectively capture the internal contextual relationships of anchors, which provides auxiliary information that further enhances the context accuracy of HAC. It leverages channel-wise redundancies in the anchor feature by employing a chunk-by-chunk prediction flow under a causal process. This intra-anchor context is combined with HAC using a Gaussian Mixture Model (GMM) to provide auxiliary information, which facilitates entropy coding such as Arithmetic Coding (AE) [20] for a highly compressed representation. Note that we opt for Scaffold-GS as our base model due to its excellent characteristics of high fidelity and efficient speed, particularly in real-world large-scale datasets. The compression performance also benefits from its feature-based design to achieve a much smaller size.

Additionally, we introduce an Adaptive Quantization Module (AQM) that dynamically adjusts quantization step sizes for different anchor attributes, achieving a balance between retaining original information and reducing entropy. Learnable masks are employed to exclude invalid Gaussians and anchors, further improving the compression ratio. Specifically, HAC++ incorporates the mask information directly into the rate calculation in a differentiable manner, eliminating the need for an extra loss term that may otherwise impact the optimal RD trade-off. Furthermore, anchor-level masks are explicitly deduced from offset masks to proactively reduce the number of anchors. This approach adaptively determines the optimal mask ratios across different bit rate points.

This paper is an extension of our *ECCV*'2024 work, HAC [21]. By revisiting the original design and incorporating more advanced techniques, we have significantly enhanced its performance. Moreover, this paper offers a comprehensive analysis of the proposed approach. We believe this work can push the performance of 3DGS compression a step forward.

Our main contributions can be summarized as follows:

1) We propose HAC++, an innovative approach that bridges the relationship between the structured hash grid and unorganized 3D Gaussians (or anchors in Scaffold-GS), achieving effective context modeling for compression.
2) To facilitate efficient entropy encoding of anchor attributes, we propose to use the interpolated hash feature to neural-predict the value distribution of anchor attributes as well as neural-predicting quantization step refinement with AQM. The intra-anchor context model is further utilized to exploit internal information of anchors to improve the context accuracy. We also incorporate adaptive offset masks to prune out ineffective Gaussians and anchors, further slimming the model.
3) Extensive experiments on five datasets demonstrate the effectiveness of HAC++ and its individual technical components. Our method achieves an average

compression ratio of over $100\times$ compared to the vanilla 3DGS model when averaged over all datasets and $20\times$ compared to the base model Scaffold-GS, while maintaining or even improving fidelity. Comprehensive analyses are conducted to provide in-depth technical insights.

II. Related Work

Neural Radiance Field and Its Compression. Neural Radiance Field (NeRF) [1] has promoted the novel view synthesis by using learnable implicit MLPs to render 3D scenes via α-composed accumulation of RGB values along a ray. However, the dense sampling of query points and reliance on large MLPs hinder the real-time rendering ability. To address this issue, methods such as Instant-NGP [2], TensoRF [3], K-planes [4], and DVGO [22] adopt explicit grid-based representations, reducing the size of the MLPs to improve the training and rendering efficiency, which, however, come at the cost of increased storage requirements.

To alleviate storage demands, compression techniques for explicit representations have been extensively developed, which fall into two main categories: *value-based* and *structural-relation-based*. Value-based methods, such as pruning [23], [24] and codebooks [23], [25], aim to reduce the number of parameters and streamline the model. Pruning removes insignificant parameters based on threshold comparisons, while codebooks cluster parameters with similar values into shared codewords for more efficient storage. Additionally, quantization [26] and entropy constraints [27] reduce the bit-length for parameter representation by minimizing their information content. However, these approaches treat parameters independently, overlooking the mutual relationships that could further enhance compression efficiency. In contrast, structural-relation-based methods leverage spatial redundancies in well-structured grids to improve the compression performance. Techniques such as wavelet decomposition [28], rank-residual decomposition [29], and spatial prediction [30] have shown great promise. Built on Instant-NGP [2], CNC [31] demonstrates the potential of leveraging structural information to achieve substantial RD performance gains. Similarly, NeRFCodec [32] represents neural fields using regular planes and vectors, applying entropy constraints for compression. CodecNeRF [33] employs a feed-forward approach to directly compress the neural radiance fields.

3D Gaussian Splatting and its compression. 3DGS [5] has innovatively addressed the challenge of slow training and rendering in NeRF while maintaining high-fidelity quality by representing 3D scenes with explicit 3D Gaussians endowed with learnable shape and appearance attributes. By adopting differentiable splatting and tile-based rasterization [7], 3D Gaussians are optimized during training to best fit their local 3D regions. Despite its advantages, the substantial Gaussians and their associated attributes necessitate effective compression techniques.

Unlike the well-structured feature grids in NeRF-based representations, Gaussians in 3DGS are inherently sparse and unorganized, posing challenges in establishing structural relationships [12], [34]. Consequently, most compression methods follow the NeRF compression pipeline and primarily focus on reducing the number of parameters through *value-based* techniques. As redundancies exist among Gaussians, pruning approaches have been extensively explored [13], [16], [35]–[37], by employing strategies like trainable masks, gradient-informed thresholds, view-dependent metrics, and other importance-evaluation mechanisms. Beyond the entire Gaussians, Gaussian attributes such as Sphere Harmonics (SH) coefficients can also be partially adjusted or pruned, as demonstrated in [36], [38]. Additionally, codebooks [13], [14], [16], [35], [39] and entropy constraints [15] have also been widely utilized for parameter reduction and compression. On the other hand, *structural-relation-based* techniques have gained much attention. [38] employs dimensional collapsing to organize Gaussians into a structured 2D grid for compression. Combining pruning with structural relations, SUNDAE [40] leverages spectral graph modeling, while Mini-Splatting [41] utilizes spatial relation-aware spawn techniques to achieve compact representations. IGS [42] predicts attributes of unstructured 3D locations using a multi-level grid for a compact 3D modeling. Notably, Scaffold-GS [19] introduces an anchor-centered framework that leverages features for neural prediction of Gaussian attributes, achieving parameter reduction with improved fidelity. Built on Scaffold-GS, HAC [21] explores inherent spatial redundancies among anchors using compact binarized hash grids for parameter quantization and entropy modeling. Concurrently, ContextGS [43] and CompGS [44] both utilize context-aware designs by modeling hierarchical anchor relations or anchor-Gaussian relations.

While existing methods effectively establish relationships among anchors, they often overlook the internal redundancies within anchors. Additionally, the unique anchor-based structure of Scaffold-GS presents opportunities for designing optimized pruning strategies. To address these limitations, we adopt Scaffold-GS as our base model, leveraging both inter- and intra-anchor contexts combined with an improved pruning strategy to enable a more compact and efficient 3DGS representation.

III. Methods

In Fig. 2, we conceptualize our HAC++ framework. In particular, HAC++ is built based on Scaffold-GS [19] (Fig. 2 **left**), which introduces anchors with their attributes $\mathcal{A}$ (feature, scaling and offsets) to cluster and neural-predict 3D Gaussian attributes (opacity, RGB, scale, and quaternion). At the core of our HAC++, it consists of a Hash-grid Assisted Context (HAC) (Fig. 2 **right**) for modeling of inter-anchor relations and an intra-anchor context model (Fig. 2 **right**) that exploits internal redundancies of anchors. Specifically: **(1) Hash-grid Assisted Context (HAC):** HAC jointly learns a structured compact hash grid (binarized for each parameter), which can be queried at any anchor location to obtain an interpolated hash feature $\boldsymbol{f}^h$ (Fig. 2 **right**). Instead of directly

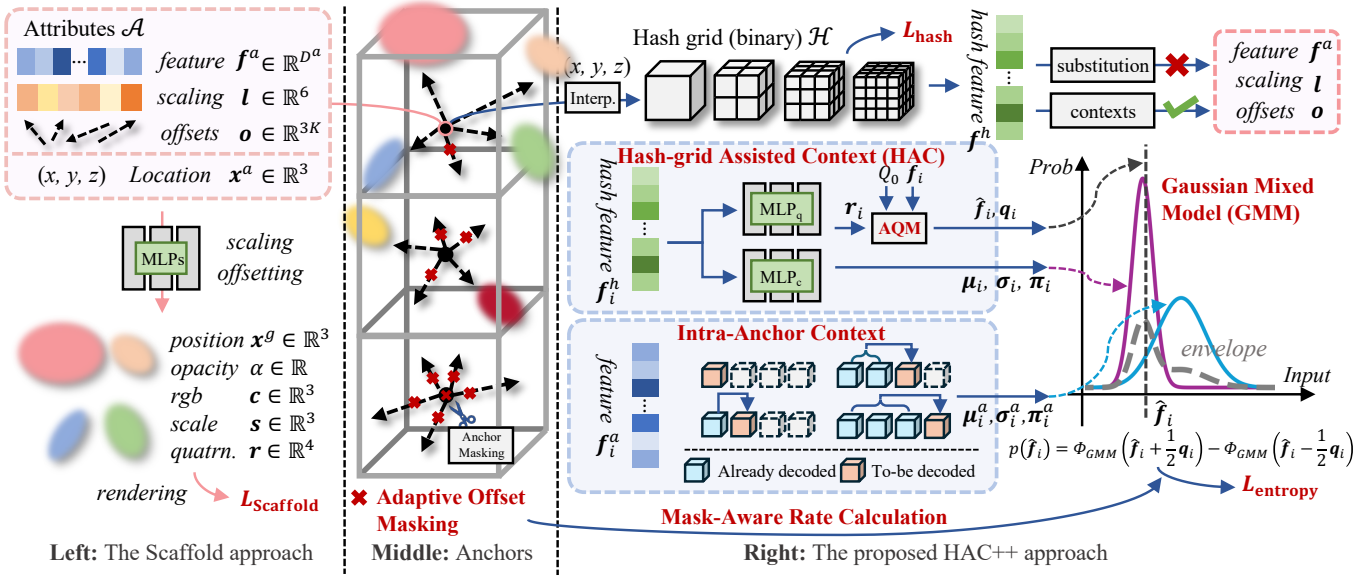

Fig. 2. **Overview of our HAC++ framework.** Built upon Scaffold-GS [19] (**left**), which introduces anchors and their attributes to neural-predict 3D Gaussian attributes, HAC++ enhances compression performance by modeling both inter- and intra-anchor relations. **Right**: HAC++ consists of a Hash-grid Assisted Context (HAC) and an Intra-Anchor Context. HAC learns a structured and compact hash grid (binarized for each parameter), which is queried at anchor locations to generate interpolated hash features $\boldsymbol{f}^h$. Instead of directly replacing anchor features, $\boldsymbol{f}^h$ serves as context for predicting the value distributions of anchor attributes, which is crucial for entropy coding. The intra-anchor context further improves prediction accuracy by eliminating internal redundancies of anchors. Additionally, HAC outputs $\boldsymbol{r}$ for the Adaptive Quantization Module (AQM), which quantizes anchor attribute values into a finite set for entropy coding. An Adaptive Offset Masking strategy (**middle**) is integrated to prune redundant Gaussians and anchors, enhancing the model's pruning efficiency across different rate points.

replacing the anchor features, $\boldsymbol{f}^h$ serves as context to predict the value distributions of anchor attributes, which is essential for subsequent entropy coding via Arithmetic Coding (AE). Additionally, HAC takes $\boldsymbol{f}^h$ as input and outputs $\boldsymbol{r}$ for the adaptive quantization module (AQM) (quantizes anchor attribute values into a finite set) and the Gaussian parameters ($\boldsymbol{\mu}^s$ and $\boldsymbol{\sigma}^s$) for modeling the value distributions of anchor attributes, from which we can compute the probability of each quantized attribute value for AE. **(2) Intra-Anchor Context:** The intra-anchor context eliminates internal redundancies of anchors, serving as auxiliary information to enhance the prediction accuracy of HAC's value distributions, thereby improving overall compression performance. **(3) Adaptive Offset Masking:** This module is utilized to prune redundant Gaussians and anchors. Specifically, the adaptive offset masking is directly integrated into the rate calculation in a differentiable manner (Fig. 2 **middle**), removing the need for an additional regularization loss term. This design enables the framework to achieve an optimal masking ratio for different rate points. In the following sections, we provide a background overview and then detail each technical component of our HAC++ framework.

A. Preliminaries

3D Gaussian Splatting (3DGS) [5] represents a 3D scene using numerous Gaussians and renders viewpoints through a differentiable splatting and tile-based rasterization. Each Gaussian is initialized from SfM and defined by a 3D covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times3}$ and location (mean) $\boldsymbol{\mu} \in \mathbb{R}^3$,

$$G(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^3$ is a random 3D point, and $\boldsymbol{\Sigma}$ is defined by a diagonal matrix $\boldsymbol{S} \in \mathbb{R}^{3\times3}$ representing scaling and rotation matrix $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ to guarantee its positive semi-definite characteristics, such that $\boldsymbol{\Sigma} = \boldsymbol{RSS}^\top\boldsymbol{R}^\top$. To render an image from a random viewpoint, 3D Gaussians are first splatted to 2D, and render the pixel value $\boldsymbol{C} \in \mathbb{R}^3$ using α-composed blending,

$$\boldsymbol{C} = \sum_{i\in I} \boldsymbol{c}_i \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j) \tag{2}$$

where $\alpha \in \mathbb{R}$ measures the opacity of each Gaussian after 2D projection, $\boldsymbol{c} \in \mathbb{R}^3$ is view-dependent color modeled by Spherical Harmonic (SH) coefficients, and I is the number of sorted Gaussians contributing to the rendering.

Scaffold-GS [19] adheres to the framework of 3DGS and introduces a more storage-friendly and fidelity-satisfying anchor-based approach. It utilizes anchors to cluster Gaussians and deduce their attributes from the attributes of attached anchors through MLPs, rather than directly storing them. Specifically, each anchor consists of a location $\boldsymbol{x}^a \in \mathbb{R}^3$ and anchor attributes $\mathcal{A} = \{\boldsymbol{f}^a \in \mathbb{R}^{D^a}, \boldsymbol{l} \in \mathbb{R}^6, \boldsymbol{o} \in \mathbb{R}^{3K}\}$, where each component represents anchor feature, scaling and offsets, respectively. During rendering, $\boldsymbol{f}^a$ is inputted into MLPs to generate attributes for Gaussians, whose locations are determined by adding $\boldsymbol{x}^a$ and $\boldsymbol{o}$, where $\boldsymbol{l}$ is utilized to regularize both locations and shapes of the Gaussians.

While Scaffold-GS has demonstrated effectiveness via this anchor-centered design, we contend there is still significant redundancy among inherent consistencies of anchors that we can fully exploit for a more compact 3DGS representation.

B. Bridging Anchors and Hash Grid

We begin the analysis by intuitively considering neighboring Gaussians share similar parameters inferred from anchor attributes. This initial perception leads us to assume anchor attributes are also consistent in space. Our main idea is to leverage the well-structured hash grid to unveil the inherent spatial consistencies of the unorganized anchors. Please also refer to experiments in 9 to observe this consistency. To verify mutual information between the hash grid and anchors, we first explore substituting anchor features $\boldsymbol{f}^a$ with hash features $\boldsymbol{f}^h$ that are acquired by interpolation using the anchor location $\boldsymbol{x}^a$ on the hash grid $\mathcal{H}$, defined as $\boldsymbol{f}^h := \text{Interp}(\boldsymbol{x}^a, \mathcal{H})$. Here, $\mathcal{H} = \{\boldsymbol{\theta}_i^l \in \mathbb{R}^{D^h} | i = 1, \dots, T^l | l = 1, \dots, L\}$ represents the hash gird, where D^h is the dimension of vector $\boldsymbol{\theta}_i^l$, T^l is the table size of the grid for level l, and L is the number of levels. We conduct a preliminary experiment on the Synthetic-NeRF dataset [1] to assess its performance, as shown in Tab. I. Direct substitution using hash features appears to yield inferior fidelity and introduces drawbacks such as unstable training (due to its impact on anchor spawning processes) and decreased testing FPS (owing to the extra interpolation operation). These results may further degrade if $\boldsymbol{l}$ and $\boldsymbol{o}$ are also substituted for a more compact model. Nonetheless, we find the fidelity degradation remains moderate, suggesting the existence of rich mutual information between $\boldsymbol{f}^h$ and $\boldsymbol{f}^a$. This prompts us to ask: *Can we exploit such mutual relation and use the compact hash features to model the context of anchor attributes $\mathcal{A}$?* This leads to the context modeling as a conditional probability:

$$p(\mathcal{A}, \boldsymbol{x}^a, \mathcal{H}) = p(\mathcal{A}|\boldsymbol{x}^a, \mathcal{H}) \times p(\boldsymbol{x}^a, \mathcal{H}) \sim p(\mathcal{A}|\boldsymbol{f}^h) \times p(\mathcal{H}) \tag{3}$$

where $\boldsymbol{x}^a$ is omitted in the last term as we assume the independence of $\boldsymbol{x}^a$ and $\mathcal{H}$ (it can be anywhere), making $p(\mathcal{H}|\boldsymbol{x}^a) \sim p(\mathcal{H})$, and do not employ entropy constraints to $\boldsymbol{x}^a$. According to information theory [45], a higher probability corresponds to lower uncertainty (entropy) and fewer bits consumption. Thus, the large mutual information between $\mathcal{A}$ and $\boldsymbol{f}^h$ ensures a large $p(\mathcal{A}|\boldsymbol{f}^h)$. Our goal is to devise a solution to effectively leverage this relationship. Furthermore, $p(\mathcal{H})$ signifies that the size of the hash grid itself should also be compressed, which can be done by adopting the existing solution for Instant-NGP compression [31].

We underscore the significance of this conditional probability based approach since it ensures both rendering speed and fidelity upper-bound unaffected as it only utilizes hash features to estimate the entropy of anchor attributes for entropy coding but does not modify the original Scaffold-GS structure. In the following subsections, we delve into the technical details of our HAC++ approach, which consists of Hach-grid Assisted Context (HAC) and an intra-anchor context that captures inter- and intra- relation of anchors, respectively.

TABLE I
EXPERIMENTAL RESULTS OF DIRECTLY SUBSTITUTING ANCHOR FEATURE $\boldsymbol{f}^a$ WITH HASH FEATURE $\boldsymbol{f}^h$ ON THE SYNTHETIC-NERF DATASET [1].

Synthetic-NeRF [1]	PSNR↑	SSIM↑	LPIPS↓
3DGS [5]	33.80	0.970	0.031
Scaffold-GS [19]	33.41	0.966	0.035
Substituting $\boldsymbol{f}^a$ with $\boldsymbol{f}^h$	32.85	0.963	0.041

C. HAC: Hash-Grid Assisted Context Framework

The pipeline of HAC is shown in the right of Fig. 2, which aim at minimizing the entropy of anchor attributes $\mathcal{A}$ by eliminating redundancies among anchors with the assistance of hash feature $\boldsymbol{f}^h$ (*i.e.*, maximize $p(\mathcal{A}|\boldsymbol{f}^h)$), thereby facilitating bit reduction when encoding anchor attributes using entropy coding like AE [20]. As shown in Fig. 2, anchor locations $\boldsymbol{x}^a$ are firstly inputted into the hash grid for interpolation, the obtained $\boldsymbol{f}^h$ are then employed as context for $\mathcal{A}$.

Adaptive Quantization Module (AQM). To facilitate entropy coding, values of $\mathcal{A}$ must be quantized to a finite set. Our experimental studies reveal that binarization, as that in BiRF [26], is unsuitable for $\mathcal{A}$ as it fails to preserve sufficient information, with a PSNR of only 31.27 dB in the Synthetic-NeRF dataset [1] if we simply binarize all $\boldsymbol{f}^a$. Thus, we opt for rounding them to maintain their comprehensive features. To ensure backpropagation, we utilize the "adding noise" operation during training and "rounding" during testing, as described in [46].

Nevertheless, the conventional rounding is essentially a quantization with a step size of "1", which is inappropriate for the scaling $\boldsymbol{l}$ and the offsets $\boldsymbol{o}$, since they are usually decimal values. To address this, we further introduce an Adaptive Quantization Module (AQM), which adaptively determines quantization steps. In particular, for the ith anchor $\boldsymbol{x}_i^a$, we denote $\boldsymbol{f}_i$ as any of its $\mathcal{A}_i$'s components: $\boldsymbol{f}_i \in \{\boldsymbol{f}_i^a, \boldsymbol{l}_i, \boldsymbol{o}_i\} \in \mathbb{R}^{D^f}$, where $D^f \in \{D^a, 6, 3K\}$ is its respective dimension. The quantization can be written as,

$$\begin{aligned} \hat{\boldsymbol{f}}_i &= \boldsymbol{f}_i + \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right) \times \boldsymbol{q}_i, && \text{for training} \\ &= \text{Round}(\boldsymbol{f}_i / \boldsymbol{q}_i) \times \boldsymbol{q}_i, && \text{for testing} \end{aligned} \tag{4}$$

where

$$\begin{aligned} \boldsymbol{q}_i &= Q_0 \times (1 + \text{Tanh}(\boldsymbol{r}_i)) \\ \boldsymbol{r}_i &= \text{MLP}_\text{q}\left(\boldsymbol{f}_i^h\right). \end{aligned} \tag{5}$$

We use a simple MLP-based context model MLP_q to predict from hash feature $\boldsymbol{f}_i^h$ a refinement $\boldsymbol{r}_i \in \mathbb{R}$, which is used to adjust the predefined quantization step size Q_0. Note that Q_0 varies for $\boldsymbol{f}^a$, $\boldsymbol{l}$, and $\boldsymbol{o}$. Eq. 5 essentially restricts the quantization step size $\boldsymbol{q}_i \in \mathbb{R}$ to be chosen within $(0, 2Q_0)$, enabling $\hat{\boldsymbol{f}}_i$ to closely resemble the original characteristics of $\boldsymbol{f}_i$, while maintaining a high fidelity.

HAC for Gaussian Distribution Modeling. To measure the bit consumption of $\hat{\boldsymbol{f}}_i$ during training, its probability needs to be estimated in a differentiable manner. As shown in Fig. 3, all three components of anchor attributes $\mathcal{A}$ exhibit statistical tendencies of Gaussian distributions, where $\boldsymbol{l}$ displays a

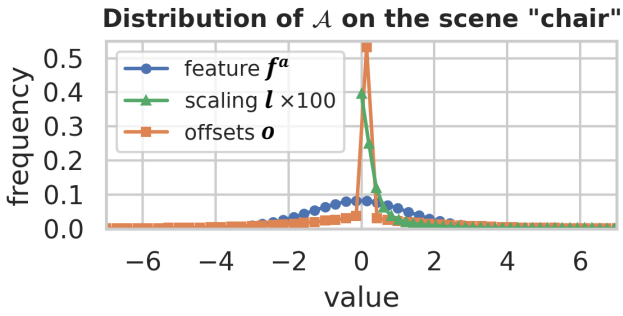

Fig. 3. Statistical analysis of the value distributions of $\mathcal{A}$ on the scene "chair" of the Synthetic-NeRF dataset [1]. All three components $\{\boldsymbol{f}^a, \boldsymbol{l}, \boldsymbol{o}\}$ exhibit statistical Gaussian distributions. Note that the values of $\boldsymbol{l}$ are scaled by a factor of 100 for better visualization.

single-sided pattern due to Sigmoid activation[1], while $\boldsymbol{o}$ exhibits an impulse at zero, suggesting the occurrence of substantial redundant Gaussians. This observation establishes a lower bound for probability prediction when all $\hat{\boldsymbol{f}}_i$s in $\mathcal{A}$ are estimated using the respective μ and σ of the statistical Gaussian Distribution of $\boldsymbol{f}^a$, $\boldsymbol{l}$ and $\boldsymbol{o}$. Nevertheless, employing a single set of μ and σ for all attributes may lack accuracy. Therefore, we assume anchor attributes $\mathcal{A}$'s values independent, and construct their respective Gaussian distributions, where their individual $\boldsymbol{\mu}^{\text{s}}$ and $\boldsymbol{\sigma}^{\text{s}}$ are estimated by a simple MLP-based context model MLP_{c} from $\boldsymbol{f}^h$. Specifically, for the ith anchor and its quantized anchor attribute vector $\hat{\boldsymbol{f}}_i$, with the estimated $\boldsymbol{\mu}^{\text{s}}_i \in \mathbb{R}^D$ and $\boldsymbol{\sigma}^{\text{s}}_i \in \mathbb{R}^D$, we compute the probability of $\hat{\boldsymbol{f}}_i$ as,

$$
\begin{aligned}
p(\hat{\boldsymbol{f}}_i) &= \int_{\hat{\boldsymbol{f}}_i - \frac{\boldsymbol{q}_i}{2}}^{\hat{\boldsymbol{f}}_i + \frac{\boldsymbol{q}_i}{2}} \phi\left(x \mid \boldsymbol{\mu}^{\text{s}}_i, \boldsymbol{\sigma}^{\text{s}}_i\right) dx \\
&= \Phi(\hat{\boldsymbol{f}}_i + \frac{\boldsymbol{q}_i}{2} \mid \boldsymbol{\mu}^{\text{s}}_i, \boldsymbol{\sigma}^{\text{s}}_i) - \Phi(\hat{\boldsymbol{f}}_i - \frac{\boldsymbol{q}_i}{2} \mid \boldsymbol{\mu}^{\text{s}}_i, \boldsymbol{\sigma}^{\text{s}}_i), \\
\boldsymbol{\mu}^{\text{s}}_i, \boldsymbol{\sigma}^{\text{s}}_i, \boldsymbol{\pi}^{\text{s}}_i &= \text{MLP}_{\text{c}}\left(\boldsymbol{f}^h_i\right)
\end{aligned}
\tag{6}
$$

where ϕ and Φ represent the probability density function and the cumulative distribution function of Gaussian distribution, respectively. Note that $\boldsymbol{\mu}^{\text{s}}$ and $\boldsymbol{\sigma}^{\text{s}}$ are Gaussian distribution parameters, and $\boldsymbol{\pi}^{\text{s}}$ represents the weight parameter for distribution combination with the intra-anchor context model that will be introduced in Subsec III-D. This approach effectively captures the relations among anchors, eliminating potential spatial redundancies without compromising the fidelity upper bound of the reconstruction branch.

D. Improving Context Accuracy with Intra Information

Intra-Anchor Context Model as Auxiliary Context. While the proposed HAC context model effectively eliminates spatial redundancies, parameter redundancies still persist within individual anchors. Integrating the intra design from our work FCGS [47], we employ an intra-anchor context model to leverage auxiliary information, which further enhances context accuracy. To construct the intra-anchor context model, we divide each $\boldsymbol{f}^a$ into N^{c} chunks and employ MLP_{a} to infer the distribution parameters for each chunk based on the previously decoded ones.

$$
\begin{aligned}
\boldsymbol{\mu}^{\text{c}}_i, \boldsymbol{\sigma}^{\text{c}}_i, \boldsymbol{\pi}^{\text{c}}_i &= \bigoplus_{n^{\text{c}}=1}^{N^{\text{c}}} \{\boldsymbol{\mu}^{\text{c}}_{i,n^{\text{c}}}, \boldsymbol{\sigma}^{\text{c}}_{i,n^{\text{c}}}, \boldsymbol{\pi}^{\text{c}}_{i,n^{\text{c}}}\}, \\
\boldsymbol{\mu}^{\text{c}}_{i,n^{\text{c}}}, \boldsymbol{\sigma}^{\text{c}}_{i,n^{\text{c}}}, \boldsymbol{\pi}^{\text{c}}_{i,n^{\text{c}}} &= \text{MLP}_{\text{a}}([\hat{\boldsymbol{f}}^a_{i,[0,n^{\text{c}}c-c)}; \boldsymbol{\mu}^{\text{s}}_i; \boldsymbol{\sigma}^{\text{s}}_i; \boldsymbol{\pi}^{\text{s}}_i])
\end{aligned}
\tag{7}
$$

where n^{c} and c denote the chunk index and the number of channels per chunk, respectively. $\boldsymbol{\mu}^{\text{c}}_{i,n^{\text{c}}}, \boldsymbol{\sigma}^{\text{c}}_{i,n^{\text{c}}}, \boldsymbol{\pi}^{\text{c}}_{i,n^{\text{c}}}$ represent the probability parameters for chunk n^{c}, each with a dimensional size of c. $\oplus$ is the channel-wise concatenate operation. $\boldsymbol{\pi}^{\text{c}}$ is the weight parameter for distribution combination with HAC in GMM. It is important to note that this approach is applied only to the anchor feature $\boldsymbol{f}^a$, as the internal redundancies of other attributes are negligible. Specifically, for the scaling $\boldsymbol{l}$, it only has a dimensionality of 6, making the potential storage savings from reducing its internal redundancies minimal. Moreover, achieving such reduction requires additional MLPs, introducing extra storage and coding complexity. As for the offset $\boldsymbol{o}$, the adaptive offset masking strategy introduced in Subsec. III-E has already removed redundant Gaussians. Further attempts to reduce redundancies among the "xyz" coordinates using the intra-anchor context are challenging. Detailed results can be found in the ablation studies in Subsec. IV-C.

Gaussian Mixture Model (GMM). To achieve joint probability estimation for the anchor feature $\boldsymbol{f}^a$ using both context models (*i.e.*, HAC and the intra-anchor context model), we employ GMM. This approach combines the Gaussian distribution parameter sets from each context model to represent the final probability. The probability estimation is formulated as follows:

$$
\begin{aligned}
p(\hat{\boldsymbol{f}}^a_i) &= \sum_{l \in \{\text{s,c}\}} \boldsymbol{\theta}^l_i \left(\Phi(\hat{\boldsymbol{f}}^a_i + \frac{\boldsymbol{q}_i}{2} \mid \boldsymbol{\mu}^l_i, \boldsymbol{\sigma}^l_i) - \Phi(\hat{\boldsymbol{f}}^a_i - \frac{\boldsymbol{q}_i}{2} \mid \boldsymbol{\mu}^l_i, \boldsymbol{\sigma}^l_i)\right), \\
\boldsymbol{\theta}^l_i &= \frac{\exp(\boldsymbol{\pi}^l_i)}{\sum_{v \in \{\text{s,c}\}} \exp(\boldsymbol{\pi}^v_i)}
\end{aligned}
\tag{8}
$$

This formulation ensures that the GMM adaptively weights the contributions of the two context models for probability estimation. By dynamically balancing their respective distribution parameter sets, it yields a more accurate and robust probability estimation for the anchor feature $\boldsymbol{f}^a$.

E. Adaptive Offset Masking

From Fig. 3, we can observe that the offset $\boldsymbol{o}$ exhibits an impulse at zero, suggesting the occurrence of substantial unnecessary Gaussians and subsequently, anchors.

Pruning trivial Gaussians or anchors directly reduces the number of parameters, thereby slimming the model. To this end, we draw inspiration from [13], which employs learnable binary masks updated via the straight-through [48] strategy to eliminate invalid Gaussians. Notably, [13] utilizes an additional loss term (*i.e.*, L_m) to regularize the mask rate, which balances the degree of compression. However, directly applying this approach to our HAC++ framework by incorporating an extra mask loss term, as in [13], poses challenges. Specifically, the weight of L_m must be manually

[1]We define $\boldsymbol{l}$ as the one *after* Sigmoid activation, which is slightly different from [19].

adjusted across different RD trade-off points in Eq. 13 as λ varies to achieve optimal mask ratios. This process is tedious, and difficult to optimize effectively.

To address this problem, we incorporate the mask into the rate calculation, allowing the mask ratio to be adaptively adjusted according to the rate via backpropagation. Following [13], for each anchor, the Gaussian-level mask $\boldsymbol{m} \in \mathbb{R}^K$ is firstly obtained as:

$$\boldsymbol{m}_i = \text{sg}\left(\mathbb{1}[\texttt{Sig}(\boldsymbol{f}_i^m) > \epsilon_m] - \texttt{Sig}(\boldsymbol{f}_i^m)\right) + \texttt{Sig}(\boldsymbol{f}_i^m) \quad (9)$$

where $\boldsymbol{f}^m \in \mathbb{R}^K$ is a learnable feature to deduce the mask, $\texttt{Sig}$ represents the sigmoid function, and sg is the stop-gradient operator. A value of 1 in $\boldsymbol{m}$ indicates the corresponding Gaussian offset is valid, while 0 denotes invalid. Invalid Gaussian offsets can be removed to save parameters. Furthermore, if all offsets associated with an anchor are pruned, then the anchor becomes irrelevant for rendering and can be entirely removed, including its $\boldsymbol{x}^a$, $\mathcal{A}$, and $\boldsymbol{m}$. To make the model aware of the rate change by anchor pruning, we introduce an anchor-level mask $\boldsymbol{m}^a \in \mathbb{R}$, which is derived from the Gaussian-level mask $\boldsymbol{m}$. This design encourages anchor pruning, thereby enhancing parameter savings:

$$m_i^a = \text{sg}\left(\mathbb{1}[\overline{m}_i > 0] - \overline{m}_i\right) + \overline{m}_i, \ \overline{m}_i = \frac{1}{K}\sum_{k=1}^{K} m_{i,k} \quad (10)$$

where $\overline{m}$ represents the average offset mask ratio of an anchor. If all the offsets are pruned on an anchor (*i.e.*, $\overline{m} = 0$), then this anchor no longer contributes to rendering and should be pruned entirely (including its $\boldsymbol{x}^a$, $\mathcal{A}$ and $\boldsymbol{m}$).

To enable adaptive mask updates, the mask information should be involved into both the rendering process and entropy estimation in a differentiable manner. For the rendering process, following [13], the Gaussian-level mask is applied to the opacity and scale of each Gaussian as $m_{i,k}\alpha_{i,k}$ and $m_{i,k}s_{i,k}$, ensuring invalid Gaussians do not contribute to rendering while valid ones remain unaffected. For entropy estimation, both Gaussian-level and anchor-level masks are included in the bit consumption calculation, making the model explicitly aware of the pruning scheme of both Gaussians and anchors. The overall entropy loss is then defined as the bit consumption b across all anchors:

$$\begin{aligned} L_{\text{entropy}} &= \sum_{i}^{N} b_i, \ \text{where} \\ b_i &= m_i^a \sum_{\boldsymbol{f}\in\{\boldsymbol{f}^a,\boldsymbol{l}\}} \sum_{j=1}^{D^f} \left(-\log_2 p(\hat{f}_{i,j})\right) + \\ &\quad m_i^a \sum_{\boldsymbol{f}\in\{\boldsymbol{o}\}} \sum_{k=1}^{K} m_{i,k} \sum_{j=1}^{3} \left(-\log_2 p(\hat{f}_{i,3k+j})\right) \end{aligned} \quad (11)$$

where N is the total number of anchors, and b_i indicates the bit consumption of the i-th anchor. Minimizing L_{entropy} promotes accurate probability estimation by $p(\hat{\boldsymbol{f}}_i)$, which in turn guides the context models' learning. By incorporating mask information into both paths' gradient chain, this approach ensures adaptive updates across different λ constraints in Eq. 13, eliminating the need for additional loss terms. Consequently, the mask update process dynamically identifies the optimal pruning ratio.

F. Hash Grid Compression

As shown in 3, the size of the hash grid $\mathcal{H}$ also significantly influences the final storage size. To this end, we binarize the hash table to $\{-1, +1\}$ using straight-through estimation (STE) [26] and calculate the occurrence frequency h_f [31] of the symbol "+1" to estimate its bit consumption:

$$L_{\text{hash}} = M_+ \times (-\log_2(h_f)) + M_- \times (-\log_2(1 - h_f)) \quad (12)$$

where M_+ and M_- are total numbers of "+1" and "−1" in the hash grid.

G. Training and Coding Process

During training, we incorporate both the rendering fidelity loss and the entropy loss to ensure the model improves rendering quality while controlling total bitrate consumption in a differentiable manner. Our overall loss is

$$Loss = L_{\text{Scaffold}} + \lambda \frac{1}{N(D^a + 6 + 3K)}(L_{\text{entropy}} + L_{\text{hash}}). \quad (13)$$

Here, L_{Scaffold} represents the rendering loss as defined in [19], which includes two fidelity penalty loss terms and one regularization term for the scaling $\boldsymbol{l}$. The second part in Eq. 13 is the estimated controllable bit consumption, including the estimated bits L_{entropy} for anchor attributes and L_{hash} for the hash grid. λ is the trade-off hyperparameters used to balance the rate and fidelity.

For the encoding/decoding process, the anchor location $\boldsymbol{x}^a$ and the binary hash grid $\mathcal{H}$ are initially encoded/decoded separately using Geometric Point Cloud Compression (GPCC) [49] and AE with h_f, respectively. Then, hash feature $\boldsymbol{f}^h$ is obtained through interpolation based on $\mathcal{H}$ and $\boldsymbol{x}^a$. Once $\boldsymbol{f}^h$ is acquired, the context models MLP_q and MLP_c are then employed to estimate quantization refinement term $\boldsymbol{r}$ and parameters of the Gaussian Distribution (*i.e.*, $\boldsymbol{\mu}^\text{s}$ and $\boldsymbol{\sigma}^\text{s}$) to derive the probability $p(\hat{\boldsymbol{f}})$ for entropy encoding/decoding with AE. For the anchor feature $\boldsymbol{f}^a$, the intra-anchor context model is also integrated to enhance the overall context accuracy via the GMM. Consequently, it is encoded/decoded sequentially chunk by chunk.

IV. Experiments

In this section, we first outline the implementation details of our HAC++ framework (Subsec. IV-A) and evaluate its performance compared to existing 3DGS compression methods (Subsec. IV-B). Additionally, we conduct ablation studies to demonstrate the effectiveness of each technical component (Subsec. IV-C). Moreover, Subsec. IV-D presents the variation in mask ratios across different RD trade-off points. To further illustrate the role of context models, we visualize the bit allocation map (Subsec. IV-H). Finally, we present in-depth statistical analyses of HAC++ from

various perspectives, including storage size (Subsec. IV-E), coding time (Subsec. IV-F), training and inference efficiency (Subsec. IV-G), and performance variations across training iterations (Subsec. IV-I).

A. Implementation Details

Basic Settings. We implement our HAC++ method based on the Scaffold-GS repository [19] using the PyTorch framework [50] and train the model on a single NVIDIA L40s GPU with 48 GB memory. We increase the dimension of the Scaffold-GS anchor feature $\boldsymbol{f}^a$ (*i.e.*, D^a) to 50, and disable its feature bank as we found it may lead to unstable training. Other hyperparameters remain unchanged (*e.g.*, the number of offsets per anchor $K = 10$). For the hash grid $\mathcal{H}$, we utilize a mixed 3D-2D structured binary hash grid, with 12 levels of 3D embeddings ranging from 16 to 512 resolutions, and 4 levels of 2D embeddings ranging from 128 to 1024 resolutions. The maximum hash table sizes are 2^{13} and 2^{15} for the 3D and 2D grids, respectively, both with a feature dimension of $D^h = 4$. We change λ from $0.5e-3$ to $4e-3$ to achieve variable bitrates. We set Q_0 as 1, 0.001 and 0.2 for $\boldsymbol{f}^a$, $\boldsymbol{l}$ and $\boldsymbol{o}$, respectively. The number of intra steps is $N^{\mathrm{c}} = 5$. In implementation, $\mathrm{MLP}_{\mathrm{q}}$ and $\mathrm{MLP}_{\mathrm{c}}$ are combined into a single 3-layer MLP with ReLU activation, while $\mathrm{MLP}_{\mathrm{a}}$ consists of multiple MLPs with varying input channels to accommodate different number of input chunks across different intra steps. For the Synthetic-NeRF [1] dataset, a lightweight version of $\mathrm{MLP}_{\mathrm{a}}$ is employed to minimize overhead. Note that the total training iteration of HAC++ is $30k$, which is consistent with Scaffold-GS and 3DGS methods to ensure a fair comparison.
Sampling Strategy. During training, using all anchors for entropy training in each iteration results in prolonged training time and potential out-of-memory (OOM) issues. Therefore, we adopt a sampling strategy: in each iteration, we only randomly sample and entropy train 5% anchors from all.
Training Process. We enhance the training process to improve stability, as shown in Fig. 4. *During the initial* $3k$ *iterations*, we train the original Scaffold-GS [19] model to ensure a stable start of the anchor attribute training and anchor spawning process. *From iteration* $3k$ *to* $10k$, we add noise to anchor attributes $\mathcal{A}$, which allows the model to adapt to quantization. Note that, in this stage, we only apply Q_0 for quantization without using $\boldsymbol{r}$. Therefore, we do not need the hash grid. Specifically, we pause the anchor spawning process between iterations $3k$ and $4k$ for a transitional period, as the sudden introduction of quantization may introduce instability to the spawning process. *After iteration* $10k$, assuming the 3D model is fitted to quantization, we fully integrate the HAC++ framework to jointly train the context models.

B. Experiment Evaluation

Baselines. We compare HAC++ against a wide range of existing 3DGS compression methods. Approaches such as [13], [14], [16], [35], [36], [39], [52] primarily rely on codebook-based or parameter pruning techniques. EAGLES [15] and SOG [38] apply entropy constraints and sorting strategies to minimize storage, respectively. For MesonGS [52], we utilize its finetuned variant to achieve improved RD performance. Additionally, Scaffold-GS [19] introduces anchor points for a compact representation. Building on Scaffold-GS, HAC [21], CompGS [44] and ContextGS [43] incorporate context models to further reduce model size.

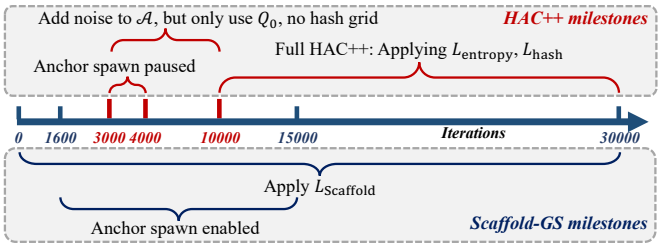

Fig. 4. Detailed training process of HAC++. We use **red lines** and **blue lines** to indicate the training process of our model and Scaffold-GS, respectively.

Datasets. We follow Scaffold-GS to perform evaluations on multiple datasets, including a small-scale Synthetic-NeRF [1] and four large-scale real-scene datasets: BungeeNeRF [8], DeepBlending [53], Mip-NeRF360 [9], and Tanks&Temples [51]. Note that we evaluate the entire 9 scenes from Mip-NeRF360 dataset [9]. Covering diverse scenarios, these datasets allow us to comprehensively demonstrate the effectiveness of all methods.
Metrics. To comprehensively evaluate compression RD performance, we calculate relative rate (size) change of our approach over others under a similar fidelity. We further provide BD-rate [54] in our ablation studies to better reflect performance changes.
Results. Quantitative results are shown in II and 5, the qualitative outputs are illustrated in 6. Full per-scene results of HAC++ across multiple fidelity metrics (*i.e.*, PSNR, SSIM [55], LPIPS [56], and size) can be found in Sec. V. HAC++ achieves significant size reductions, surpassing $100\times$ compared to the vanilla 3DGS [5] on average across all datasets, while also delivering improved fidelity. Moreover, it achieves over $20\times$ size reduction compared to the base model, Scaffold-GS [19]. Notably, HAC++ surpasses Scaffold-GS in fidelity, primarily due to two factors: 1) the entropy loss effectively regularizes the model to prevent overfitting, and 2) we increase the dimension of the anchor feature (*i.e.*, D^a) to 50, resulting in a larger model volume. For methods primarily relying on codebooks and pruning techniques (mid-chunk), their designs struggle to achieve significant storage reductions due to the limited utilization of contextual information among parameters. While SOG [38] achieves a small size, it significantly sacrifices fidelity. While ContextGS [43] and CompGS [44] each introduce their context models, HAC++ demonstrates superior performance owing to its more effective and well-optimized context model designs and mask strategies.
Bitstream. The bitstream of HAC++ comprises five components: anchor attributes $\mathcal{A}$ (*i.e.*, $\boldsymbol{f}^a$, $\boldsymbol{l}$, and $\boldsymbol{o}$), binary hash grid $\mathcal{H}$, offset masks $\boldsymbol{m}$, anchor locations $\boldsymbol{x}^a$, and MLP parameters. Among these, $\mathcal{A}$ is entropy-encoded using AE [20], leveraging probabilities estimated by the context models, and constitutes the dominant portion of storage. The

TABLE II
QUANTITATIVE RESULTS. 3DGS [5] AND SCAFFOLD-GS [19] ARE TWO BASELINES. FOR OUR APPROACH, WE PROVIDE TWO RESULTS WITH DIFFERENT SIZE AND FIDELITY TRADE-OFFS BY ADJUSTING λ. A SMALLER λ RESULTS IN A LARGER SIZE BUT IMPROVED FIDELITY, AND VICE VERSA. THE BEST AND SECOND-BEST RESULTS ARE HIGHLIGHTED IN RED AND YELLOW CELLS. THE SIZES ARE MEASURED IN MB.

Datasets	Synthetic-NeRF [1]				Mip-NeRF360 [9]				Tank&Temples [51]			
methods	psnr↑	ssim↑	lpips↓	size↓	psnr↑	ssim↑	lpips↓	size↓	psnr↑	ssim↑	lpips↓	size↓
3DGS [5]	33.80	0.970	0.031	68.46	27.46	0.812	0.222	750.9	23.69	0.844	0.178	431.0
Scaffold-GS [19]	33.41	0.966	0.035	19.36	27.50	0.806	0.252	253.9	23.96	0.853	0.177	86.50
Lee *et al.* [13]	33.33	0.968	0.034	5.54	27.08	0.798	0.247	48.80	23.32	0.831	0.201	39.43
Compressed3D [14]	32.94	0.967	0.033	3.68	26.98	0.801	0.238	28.80	23.32	0.832	0.194	17.28
EAGLES [15]	32.51	0.964	0.039	4.26	27.14	0.809	0.231	58.91	23.28	0.835	0.203	28.99
LightGaussian [16]	32.73	0.965	0.037	7.84	27.00	0.799	0.249	44.54	22.83	0.822	0.242	22.43
SOG [38]	31.37	0.959	0.043	2.00	26.56	0.791	0.241	16.70	23.15	0.828	0.198	9.30
Navaneet *et al.* [35]	32.99	0.966	0.037	3.10	27.12	0.806	0.240	19.33	23.44	0.838	0.198	12.50
Reduced3DGS [36]	33.02	0.967	0.035	2.11	27.19	0.807	0.230	29.54	23.57	0.840	0.188	14.00
RDOGaussian [39]	33.12	0.967	0.035	2.31	27.05	0.802	0.239	23.46	23.34	0.835	0.195	12.03
MesonGS-FT [52]	32.92	0.968	0.033	3.66	26.99	0.796	0.247	27.16	23.32	0.837	0.193	16.99
HAC (lowrate) [21]	33.24	0.967	0.037	1.18	27.53	0.807	0.238	15.26	24.04	0.846	0.187	8.10
HAC (highrate) [21]	33.71	0.968	0.034	1.86	27.77	0.811	0.230	21.87	24.40	0.853	0.177	11.24
ContextGS (lowrate) [43]	32.79	0.965	0.040	1.01	27.62	0.808	0.237	12.68	24.20	0.852	0.184	7.05
ContextGS (highrate) [43]	33.51	0.968	0.035	1.56	27.75	0.811	0.231	18.41	24.29	0.855	0.176	11.80
CompGS (lowrate) [44]	/	/	/	/	26.37	0.778	0.276	8.83	23.11	0.815	0.236	5.89
CompGS (highrate) [44]	/	/	/	/	27.26	0.803	0.239	16.50	23.70	0.837	0.208	9.60
Ours HAC++ (lowrate)	33.03	0.966	0.039	0.88	27.60	0.803	0.253	8.34	24.22	0.849	0.190	5.18
Ours HAC++ (highrate)	33.76	0.969	0.033	1.84	27.82	0.811	0.231	18.48	24.32	0.854	0.178	8.63

Datasets	DeepBlending [53]				BungeeNeRF [8]			
methods	psnr↑	ssim↑	lpips↓	size↓	psnr↑	ssim↑	lpips↓	size↓
3DGS [5]	29.42	0.899	0.247	663.9	24.87	0.841	0.205	1616
Scaffold-GS [19]	30.21	0.906	0.254	66.00	26.62	0.865	0.241	183.0
Lee *et al.* [13]	29.79	0.901	0.258	43.21	23.36	0.788	0.251	82.60
Compressed3D [14]	29.38	0.898	0.253	25.30	24.13	0.802	0.245	55.79
EAGLES [15]	29.72	0.906	0.249	52.34	25.89	0.865	0.197	115.2
LightGaussian [16]	27.01	0.872	0.308	33.94	24.52	0.825	0.255	87.28
SOG [38]	29.12	0.892	0.270	5.70	22.43	0.708	0.339	48.25
Navaneet *et al.* [35]	29.90	0.907	0.251	13.50	24.70	0.815	0.266	33.39
Reduced3DGS [36]	29.63	0.902	0.249	18.00	24.57	0.812	0.228	65.39
RDOGaussian [39]	29.63	0.902	0.252	18.00	23.37	0.762	0.286	39.06
MesonGS-FT [52]	29.51	0.901	0.251	24.76	23.06	0.771	0.235	63.11
HAC (lowrate) [21]	29.98	0.902	0.269	4.35	26.48	0.845	0.250	18.49
HAC (highrate) [21]	30.34	0.906	0.258	6.35	27.08	0.872	0.209	29.72
ContextGS (lowrate) [43]	30.11	0.907	0.265	3.45	26.90	0.866	0.222	14.00
ContextGS (highrate) [43]	30.39	0.909	0.258	6.60	27.15	0.875	0.205	21.80
CompGS (lowrate) [44]	29.30	0.895	0.293	6.03	/	/	/	/
CompGS (highrate) [44]	29.69	0.901	0.279	8.77	/	/	/	/
Ours HAC++ (lowrate)	30.16	0.907	0.266	2.91	26.78	0.858	0.235	11.75
Ours HAC++ (highrate)	30.34	0.911	0.254	5.28	27.17	0.879	0.196	20.82

hash grid $\mathcal{H}$ and masks $\boldsymbol{m}$ are binary data encoded by AE based on their occurrence frequencies. Anchor locations $\boldsymbol{x}^a$ are 16-bit quantized and losslessly compressed using GPCC [49]. The MLP parameters are stored directly using 32-bit precision. For detailed analysis of storage size and coding time, please refer to Subsec. IV-E and IV-F, respectively.

C. Ablation Study

In this subsection, we conduct ablation studies to evaluate the effectiveness of individual technical components in HAC++. Our experiments are performed on the Mip-NeRF360 dataset [9], which features large-scale and diverse scenes, offering reliable results. Quantitative comparisons are presented as curves in Fig. 7 and 8. BD-rate [54] is calculated to measure the relative size change compared to the full HAC++ method at the same fidelity in Tab. III. Note that positive BD-rate values indicate increased sizes at the same fidelity compared to the anchor method (*i.e.*, HAC++), which is undesirable. The ablation studies encompass three aspects:

- **Fidelity Preservation**. (Fig. 7 and Tab. III) We investigate the impact of disabling the adaptive quantization module (AQM). Specifically, the adaptive term $\boldsymbol{r}$ is removed, retaining only Q_0 to ensure a necessary decimal quantization step. This modification leads to a significant drop in fidelity, especially at higher rates, as the anchor attributes $\mathcal{A}$ fail to retain sufficient information for rendering after quantization. The drastic reduction in fidelity results in an incalculable BD-rate due to the absence of overlap with HAC++ in fidelity, which is denoted as N/A in Tab. III.
- **Probability Estimation Accuracy**. (Fig. 7 and Tab. III) To assess context models, we perform three experiments: **(1)** We set the hash grid to all zeros to eliminate mutual information from HAC. This downgrades the conditional probability from $p(\mathcal{A}|\boldsymbol{f}^h)$ to $p(\mathcal{A})$, leading to inaccurate probability estimation, consequently, a significantly larger model size. **(2)** We remove the auxiliary intra-anchor context model, which also causes a noticeable increase

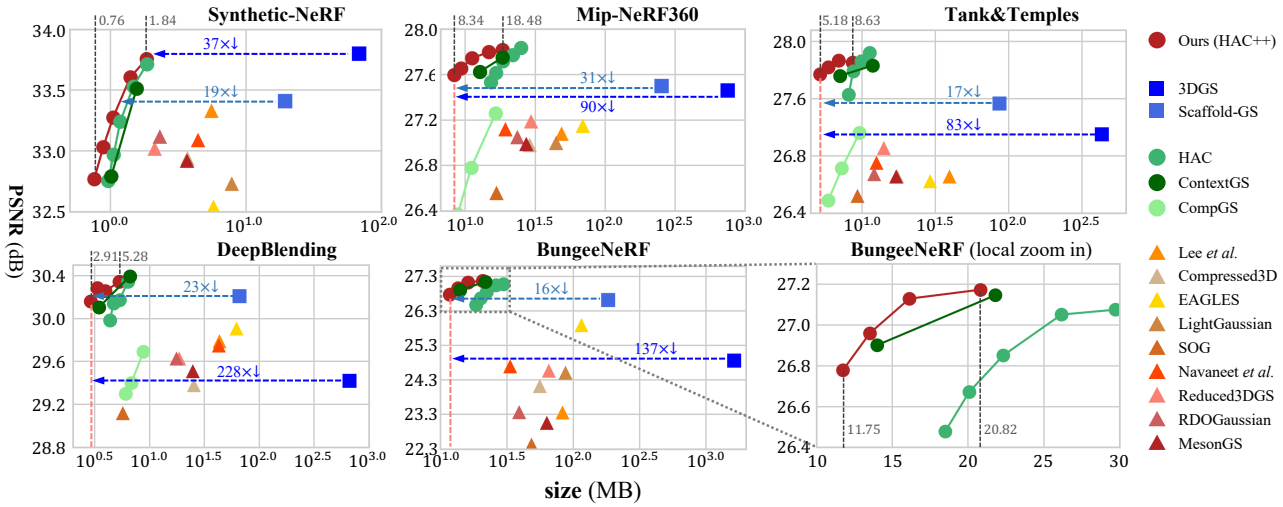

Fig. 5. RD curves for quantitative comparisons. We vary λ to achieve variable bitrates. Note that $\log_{10}$ scale is used for x-axis for better visualization.

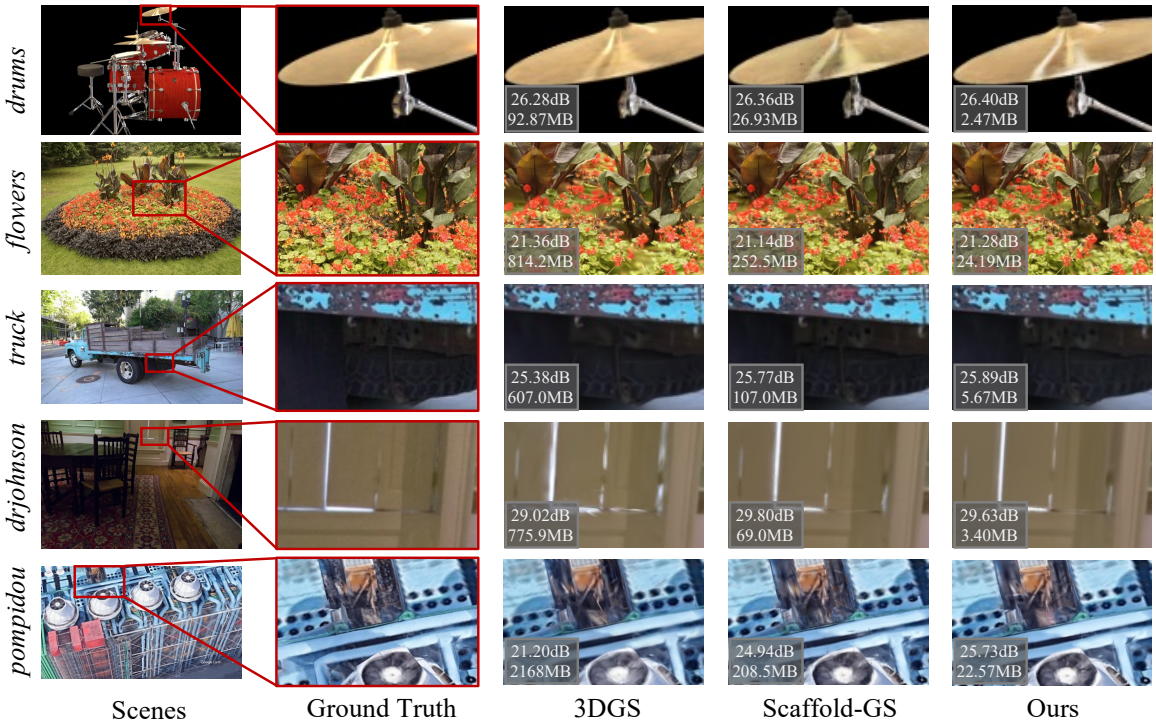

Fig. 6. Qualitative comparisons of scenes across different datasets. PSNR and size results are given at lower-left.

in mode size. Here's a refined version for clarity and style: **(3)** To evaluate the approach of fusing the two context models, we examine the effectiveness of the GMM. Specifically, we replace the GMM with a simpler concatenation strategy for combining the two context models. This modification results in only a *single* set of Gaussian distribution parameters being deduced for probability estimation. As observed from the 5th line in Tab. III, this approach is suboptimal, since the GMM provides a more flexible distribution estimation to better approximate the true conditional distribution. We also evaluate the impact of applying the intra-anchor context model to the scaling $\boldsymbol{l}$ and the offset $\boldsymbol{o}$ individually. The results in Tab. III show negligible improvements. As their RD curves overlap significantly with that of HAC++, we omit them from Fig. 7 for clarity. As highlighted in Tab. III, enhancing entropy estimation accuracy contributes to an improved BD-rate while preserving comparable training and rendering efficiency.

- **Masking Strategies**. (Fig. 7 and Tab. III) We evaluate the effects of different mask strategies, including Gaussian-level and anchor-level masks. The

TABLE III
ABLATION STUDIES ON THE MIP-NERF360 DATASET [9]. POSITIVE BD-RATE VALUES INDICATE INCREASED SIZES COMPARED TO ANCHOR METHOD HAC++ AT THE SAME FIDELITY, WHICH ARE UNEXPECTED.

Ablation Item	**BD-rate** ↓	**Train Time** (s)	**FPS**
W/o AQM	N/A	2073	140
W/o HAC information	+63.3%	2303	144
W/o intra-anchor probability	+14.7%	2130	143
W/o using GMM for probability fusion	+5.7%	2289	140
W/ intra-anchor context on the scaling $\boldsymbol{l}$	−0.1%	2386	142
W/ intra-anchor context on the offset $\boldsymbol{o}$	+0.9%	2456	141
W/o adaptive offset masking $\boldsymbol{m}$	+31.4%	2239	125
W/o anchor-level mask $\boldsymbol{m}^a$ in entropy loss	+9.3%	2277	137
W/ extra mask loss term, instead of mask-aware rate	+9.6%	2208	147
W/o using GPCC for location coding	+14.0%	2292	141
HAC++ (anchor method)	0.0%	2292	141

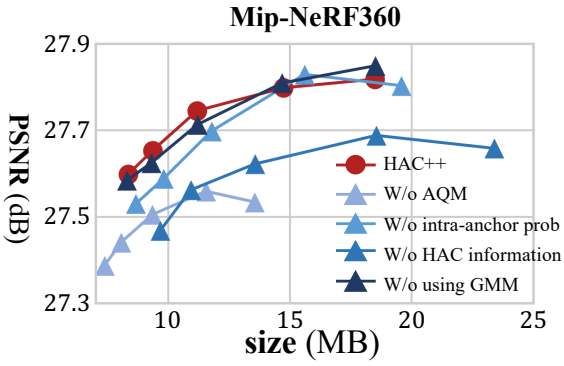

Fig. 7. RD curve of ablation study on **context models**. Experiments are conducted on the Mip-NeRF360 dataset [9] dataset. We vary λ for variable rates.

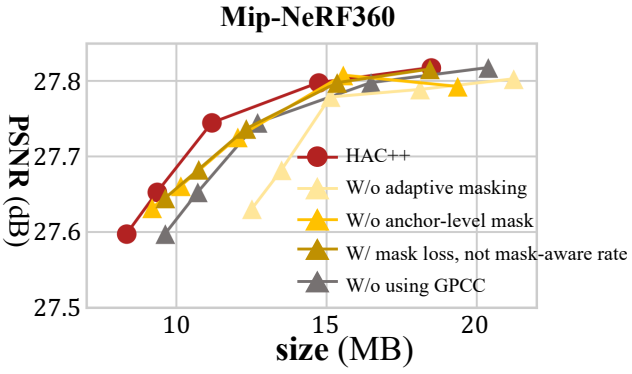

Fig. 8. RD curve of ablation study on **masking strategies** and **GPCC**. Experiments are conducted on the Mip-NeRF360 dataset [9]. We vary λ for variable rates.

Gaussian-level mask $\boldsymbol{m}$, which is applied consistently to both rendering and entropy paths, demonstrates its effectiveness in reducing parameter count and enhancing compression performance. For the anchor-level mask $\boldsymbol{m}^a$, its removal makes the anchor masking scheme passive, reducing its effectiveness in eliminating redundant anchors. We also present the result of using an extra mask loss term for regularization as an alternative to the proposed mask-aware rate calculation, which proves less effective. Finally, employing GPCC for anchor location coding successfully reduces the storage size further. As shown in Tab. III, pruning invalid Gaussians and anchors not only enhances compression performance but also improves rendering efficiency.

Overall, each technical component in HAC++ contributes to improved rate-distortion performance. Collectively, these components form a robust framework for effective 3DGS compression.

D. Mask Ratio Analysis

We present statistical data on the mask ratio of the adaptive offset masking in Tab. VIII. The valid ratio $r(\cdot)$ denotes the ratio of value 1 in the mask, which indicates the corresponding anchor/Gaussian is valid. The 2nd and 3rd columns indicate that while the total number of anchors exceeds $560k$ due to the large scale of the dataset, only a subset of these anchors are valid. As λ increases, stricter rate constraints lead to a decrease in the mask ratio. A similar trend is observed for Gaussians (offsets) in the 4th column. Since each anchor contains $K = 10$ Gaussians, an anchor is considered valid if even just one of its Gaussians is valid. This results in the valid ratio of Gaussians ($r(\text{Gaussian})$) is significantly smaller than that of anchors ($r(\text{anchor})$). Moreover, the value $\frac{r(\text{Gaussian})}{r(\text{anchor})}$ decreases as λ increases, which indicates a reduced proportion of valid Gaussians within valid anchors, meaning growing positional redundancies. Overall, leveraging this mask information, we effectively eliminate *invalid anchors* and *invalid offsets in valid anchors*, achieving a compact representation.

E. Decomposition of Storage Size of Difference Components

This subsection provides a detailed analysis of the storage size associated with various attribute components, from both a macroscopic and microscopic perspective. It is important to note that only valid anchors require encoding, while invalid anchors are directly discarded. Specifically, for offsets $\boldsymbol{o}$, only the valid offsets are encoded for calculation, which is different from other attributes.

TABLE IV
ENCODING TIME FOR DIFFERENT COMPONENTS ON THE MIP-NERF360 DATASET [9], WHICH CONTAINS OVER $400K$ ANCHORS ON AVERAGE. ALL TIMES ARE MEASURED IN SECONDS, WITH EACH COMPONENT'S PERCENTAGE CONTRIBUTION INDICATED IN PARENTHESES.

λ	# Valid Anchors	Total Time (s)	$\boldsymbol{x}^a$	$\boldsymbol{f}^a$	$\boldsymbol{l}$	$\boldsymbol{o}$	$\mathcal{H}$	$\boldsymbol{m}$	Others
$0.5e-3$	491852	18.10	3.27 (18%)	8.31 (46%)	2.82 (16%)	3.13 (17%)	0.01 (0%)	0.01 (0%)	0.55 (3%)
$1e-3$	449658	15.19	2.93 (19%)	6.95 (46%)	2.34 (15%)	2.84 (19%)	0.01 (0%)	0.01 (0%)	0.50 (3%)
$2e-3$	396485	12.17	2.57 (21%)	5.59 (46%)	1.86 (15%)	1.97 (16%)	0.01 (0%)	0.01 (0%)	0.33 (3%)
$3e-3$	359629	10.59	2.29 (22%)	4.82 (46%)	1.61 (15%)	1.56 (15%)	0.01 (0%)	0.01 (0%)	0.31 (3%)
$4e-3$	342049	9.80	2.20 (22%)	4.38 (45%)	1.46 (15%)	1.37 (14%)	0.01 (0%)	0.01 (0%)	0.26 (3%)

TABLE V
DECODING TIME FOR DIFFERENT COMPONENTS ON THE MIP-NERF360 DATASET [9], WHICH CONTAINS OVER $400K$ ANCHORS ON AVERAGE. ALL TIMES ARE MEASURED IN SECONDS, WITH EACH COMPONENT'S PERCENTAGE CONTRIBUTION INDICATED IN PARENTHESES.

λ	# Valid Anchors	Total Time (s)	$\boldsymbol{x}^a$	$\boldsymbol{f}^a$	$\boldsymbol{l}$	$\boldsymbol{o}$	$\mathcal{H}$	$\boldsymbol{m}$	Others
$0.5e-3$	491852	30.86	1.18 (4%)	14.97 (49%)	7.46 (24%)	6.55 (21%)	0.01 (0%)	0.02 (0%)	0.67 (2%)
$1e-3$	449658	25.62	1.09 (4%)	12.59 (49%)	6.35 (25%)	5.07 (20%)	0.01 (0%)	0.02 (0%)	0.51 (2%)
$2e-3$	396485	20.05	0.96 (5%)	10.06 (50%)	5.06 (25%)	3.55 (18%)	0.01 (0%)	0.02 (0%)	0.39 (2%)
$3e-3$	359629	17.22	0.88 (5%)	8.73 (51%)	4.32 (25%)	2.91 (17%)	0.01 (0%)	0.01 (0%)	0.35 (2%)
$4e-3$	342049	15.77	0.83 (5%)	8.03 (51%)	3.98 (25%)	2.61 (17%)	0.01 (0%)	0.02 (0%)	0.29 (2%)

TABLE VI
STORAGE SIZE OF DIFFERENT COMPONENTS ON THE MIP-NERF360 DATASET [9]. ALL SIZES ARE MEASURED IN **MB**.

λ	Total Size	$\boldsymbol{x}^a$	$\boldsymbol{f}^a$	$\boldsymbol{l}$	$\boldsymbol{o}$	$\mathcal{H}$	$\boldsymbol{m}$	MLP
$0.5e-3$	18.48	0.91	9.54	2.52	4.49	0.12	0.56	0.33
$1e-3$	14.73	0.83	7.52	2.18	3.27	0.12	0.48	0.33
$2e-3$	11.18	0.75	5.61	1.81	2.19	0.11	0.38	0.33
$3e-3$	9.35	0.70	4.61	1.60	1.69	0.10	0.32	0.33
$4e-3$	8.34	0.67	4.00	1.48	1.47	0.09	0.30	0.33

TABLE VII
PER-PARAMETER BITS OF DIFFERENT COMPONENTS ON THE MIP-NERF360 DATASET [9]. ALL SIZES ARE MEASURED IN **BITS**.

λ	$\boldsymbol{x}^a$	$\boldsymbol{f}^a$	$\boldsymbol{l}$	$\boldsymbol{o}$	$\boldsymbol{m}$
$0.5e-3$	5.19	3.23	7.15	6.72	0.94
$1e-3$	5.21	2.77	6.75	6.45	0.88
$2e-3$	5.34	2.32	6.35	6.20	0.80
$3e-3$	5.48	2.09	6.19	6.08	0.74
$4e-3$	5.53	1.90	6.00	5.93	0.72

TABLE VIII
RATIO OF VALID ANCHORS AND GAUSSIANS (*i.e.*, OFFSETS) ON THE MIP-NERF360 DATASET [9]. EACH ANCHOR HAS $K = 10$ GAUSSIANS.

λ	# Total Anchors	r(anchor)	r(Gaussian)
$0.5e-3$	560425	0.865	0.314
$1e-3$	561429	0.790	0.240
$2e-3$	564852	0.695	0.168
$3e-3$	570567	0.628	0.132
$4e-3$	571923	0.596	0.118

TABLE IX
ENCODING AND DECODING TIME ACROSS ALL DATASETS. "V. A." DENOTES "AVERAGED NUMBER OF VALID ANCHORS".

Datasets	V. A.	Enc. Time (s)		Dec. Time (s)	
		lowrate	highrate	lowrate	highrate
Synthetic-NeRF [1]	38467	0.80	1.57	1.18	2.36
Mip-NeRF360 [9]	407934	9.80	18.10	15.77	30.86
Tank&Temples [51]	240915	6.01	9.01	9.58	14.20
DeepBlending [53]	145773	3.35	5.17	4.92	7.86
BungeeNeRF [8]	460713	12.44	18.62	18.67	28.58

- From a macroscopic perspective, the total storage size for each component is summarized in Tab. VI. As λ increases, the rate constraint is stricter, leading to an overall reduction in storage size. Notably, the storage size of the offset $\boldsymbol{o}$ decreases most significantly ($3\times$ reduction), due to two factors: a reduction in per-parameter bits (see Tab. VII) and the decreased ratio of valid offsets within valid anchors (indicated by $\frac{r(\text{Gaussian})}{r(\text{anchor})}$ in Tab. VIII), leading to fewer parameters.
- From a microscopic perspective, the per-parameter bits for each component are in Tab. VII. While most attributes show a decreasing trend as λ increases, the anchor location $\boldsymbol{x}^a$ behaves differently. Since $\boldsymbol{x}^a$ is compressed using GPCC, more anchor locations in high-rate segments lead to greater positional redundancies, lowering the per-parameter bits. For the mask $\boldsymbol{m}$, as λ increases, the ratio of valid offsets within valid anchors decreases, making zeros more dominant in the offset mask and consequently reducing entropy for each parameter.

F. Decomposition of Coding Time of Difference Components

In this subsection, we analyze the coding time of different attributes in HAC++. Tab. IV and V present the times

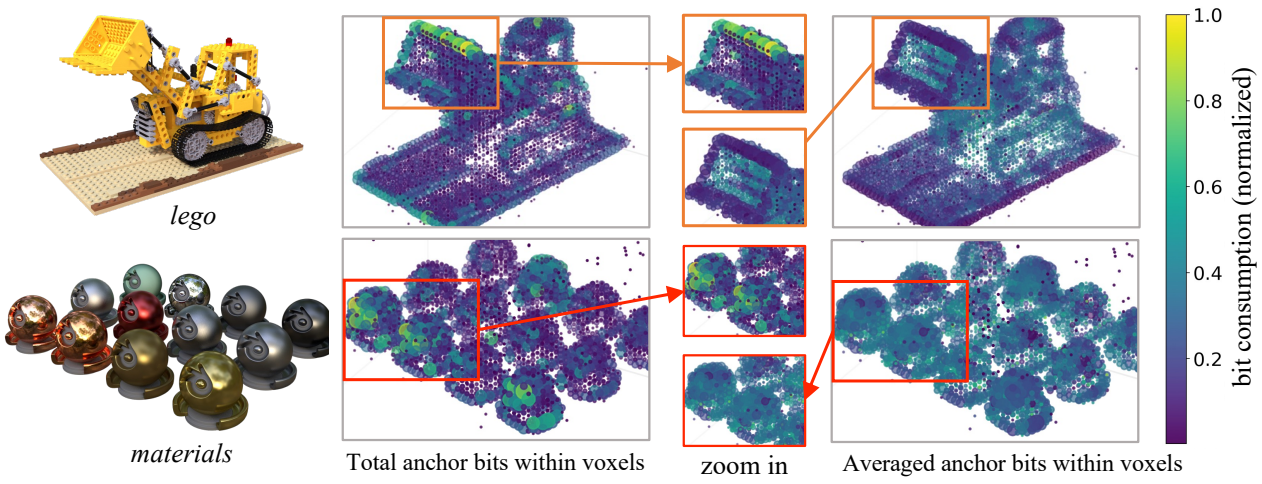

Fig. 9. Visualization of bit allocation across anchors for the scenes "lego" and "materials" on the Synthetic-NeRF dataset [1]. The 3D space is voxelized, with each voxel represented by a ball and the radius of a ball indicating the number of anchors in the voxel. For the 2nd column, the color of a ball indicates the *total* bit consumption of all anchors in the voxel, while for the 4th column, the color represents the *averaged* bit consumption per anchor within a voxel. The 3rd column gives zoom-in views. It shows more anchors are allocated to important regions while the bit consumption for each anchor is smooth.

TABLE X
TRAINING TIME AND PEAK GPU MEMORY USAGE OF OUR APPROACH COMPARED TO PREVIOUS METHODS ON THE MIP-NERF360 DATASET [9].

Methods	Training Time (s)		Peak GPU Mem (GB)	
	lowrate	highrate	lowrate	highrate
3DGS [5]	1590		12.00	
Scaffold-GS [19]	1286		9.69	
HAC++	2384	2278	10.87	11.66

TABLE XI
GAUSSIAN COUNT AND FPS OF OUR APPROACH COMPARED TO PREVIOUS METHODS ON THE MIP-NERF360 DATASET [9].

Methods	# Valid Gaussians		FPS	
	lowrate	highrate	lowrate	highrate
3DGS [5]	3175k		99	
Scaffold-GS [19]	5674k		135	
HAC++	682k	1853k	151	130

for encoding and decoding, respectively. Despite the large scale of the Mip-NeRF360 dataset, with over $400K$ valid anchors on average, both encoding and decoding processes remain efficient within 30 seconds. The anchor locations $\boldsymbol{x}^a$ are compressed using GPCC, leading to a relatively long encoding time due to the complex RD search process, but a significantly shorter decoding time. Conversely, the other attributes are encoded and decoded via AE, where the decoding phase involves index searching, increasing its complexity. The "Others" category accounts for the time taken for model-related operations, such as MLP processing. Additionally, Table IX reports the coding times across all datasets. For simpler datasets, the coding time is much shorter due to the smaller number of anchors. Overall, the coding process is efficient and could be further optimized through advanced codec techniques, which we consider an engineering task for future work.

G. Training and Rendering Efficiency

In this subsection, we evaluate the training and rendering efficiency of our HAC++ method, as shown in Tab. X and XI. For the two base methods, 3DGS [5] and Scaffold-GS [19], Scaffold-GS demonstrates lower training times and faster rendering FPS, despite having a higher Gaussian count. This performance advantage is attributed to its pre-filtering scheme, which skips computations for out-of-view anchors (and Gaussians) and precomputes colors to avoid the complex SH calculations. These features enable Scaffold-GS to achieve faster rendering speeds compared to 3DGS. Since HAC++ is built upon Scaffold-GS, it inherits these efficient features. **During training**, as shown in Tab. X, the inclusion of context models in HAC++ results in an 81% increase in training time compared to Scaffold-GS. However, HAC++ still maintains a fast training speed. Notably, the training time remains consistent across different rates because, during training, masks are applied to all Gaussians/anchors to preserve gradients, regardless of their validity. Consequently, forward and backward passes for both valid and invalid Gaussians/anchors are executed, making the time consumption consistent. Additionally, due to the sampling strategy employed, the peak GPU memory usage remains reasonable, with only a 16% average increase over Scaffold-GS. **During inference**, as shown in Tab. XI, HAC++ benefits from its context modeling design, enabling the removal of the hash grid after decoding $\mathcal{A}$. This design eliminates the need for additional operations during rendering. Furthermore, HAC++ exhibits FPS improvements over Scaffold-GS, particularly at low rates. This improvement arises from the adaptive masking design, which prunes invalid Gaussians/anchors, thereby accelerating the rendering process.

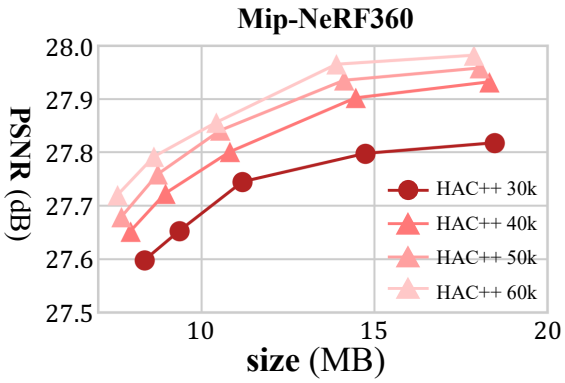

Fig. 10. Training with different number of iterations on the Mip-NeRF360 dataset [9].

TABLE XII
TRAINING TIME AND BD-RATE CHANGES ACROSS DIFFERENT ITERATIONS. EXPERIMENTS ARE CONDUCTED ON THE MIP-NERF360 DATASET [9]. NEGATIVE BD-RATE VALUES INDICATE DECREASES IN RELATIVE SIZE COMPARED TO THE ANCHOR METHOD (HAC++) AT THE SAME FIDELITY, REPRESENTING PERFORMANCE IMPROVEMENTS.

Training Iterations	**Training Time** (s)	**BD-rate** ↓
HAC++ 60k	5294	−32.9%
HAC++ 50k	4224	−25.6%
HAC++ 40k	3307	−17.5%
HAC++ 30k (anchor method)	2292	0.0%

H. Visualization of Bit Allocation

While HAC++ measures the parameters' bit consumption, we are interested in the bit allocation across different local areas in the space. In Fig. 9, we utilize scenes in Synthetic-NeRF dataset [1] for visualization, and represent bit allocation conditions by voxelized colored balls. As observed from the 2nd column of visualized sub-figures, the model tends to allocate more total bits to areas with complex appearances or sharp edges. For instance, edge areas in "lego" and specular objects in "materials" exhibit higher total bit consumption due to the complex textures. The analysis of the 4th column from an averaging viewpoint reveals varied trends in bit consumption per anchor. In high bit-consumption voxels, creating more anchors for precise modeling averages the bit per anchor, smoothing or reducing bit consumption for each. This aligns with our assumption that anchors demonstrate inherent consistency in the 3D space where nearby anchors exhibit similar values of attributes, making it easier for the hash grid to accurately estimate their value probabilities.

I. Training with Different Number of Iterations

In this subsection, we analyze the impact of varying number of training iterations on performance, as illustrated in Fig. 10 and Tab. XII. HAC++, leveraging a per-scene optimization scheme, exhibits enhanced performance with increased training iterations, at the expense of extended training time. Notably, a substantial BD-rate improvement of -32.9% is observed when comparing 60k iterations to the default 30k iterations. When comparing performance under the same λ value across different training curves, the primary gains are attributed to fidelity improvements, with minimal reductions in storage size. However, as the total number of iterations increases, the incremental benefit of additional iterations becomes diminished.

Overall, selecting appropriate training iterations is crucial for achieving a desirable trade-off between training time and compression performance, which can be tailored to specific applications or computational resources.

V. SUPPLEMENTARY: PER-SCENE RESULT OF HAC++

We present per-scene results of our HAC++ method across multiple fidelity metrics (*i.e.*, PSNR, SSIM [55] and LPIPS [56]) over all datasets. Results are presented in the Appendix.

VI. CONCLUSION AND LIMITATION

We have explored the relationship between unorganized, sparse Gaussians (or anchors) and well-structured hash grids, leveraging their mutual information to achieve compressed 3DGS representations. Through the integration of a Hash-grid Assisted Context (HAC) module and an intra-anchor context model, our HAC++ method achieves the SoTA compression performance. Extensive experiments validate the effectiveness of HAC++ and its technical components through comprehensive analyses. By addressing the significant challenge of large storage requirements in 3DGS representations, our work paves the way for their deployment in large-scale scenes.

Limitation. The main limitation of HAC++ lies in its increased training time compared to the base method, Scaffold-GS, due to the additional loss term and the incorporation of context models. Future work could explore lightweight context model designs to alleviate this issue. Furthermore, HAC++ establishes relationships among anchors indirectly through an intermediate hash grid. Investigating approaches that directly model relationships among anchors could provide an alternative strategy for redundancy elimination.

ACKNOWLEDGEMENT

The paper is supported in part by The National Natural Science Foundation of China (No. 62325109, U21B2013). MH is supported by funding from The Australian Research Council Discovery Program DP230101176.

REFERENCES

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[2] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics (ToG)*, vol. 41, no. 4, pp. 1–15, 2022.

[3] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "Tensorf: Tensorial radiance fields," in *European Conference on Computer Vision*. Springer, 2022, pp. 333–350.

[4] S. Fridovich-Keil, G. Meanti, F. R. Warburg, B. Recht, and A. Kanazawa, "K-planes: Explicit radiance fields in space, time, and appearance," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 12 479–12 488.

[5] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, 2023.

[6] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[7] C. Lassner and M. Zollhofer, "Pulsar: Efficient sphere-based neural rendering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1440–1449.

[8] Y. Xiangli, L. Xu, X. Pan, N. Zhao, A. Rao, C. Theobalt, B. Dai, and D. Lin, "Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering," in *European conference on computer vision*. Springer, 2022, pp. 106–122.

[9] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5470–5479.

[10] G. Chen and W. Wang, "A survey on 3d gaussian splatting," *arXiv preprint arXiv:2401.03890*, 2024.

[11] B. Fei, J. Xu, R. Zhang, Q. Zhou, W. Yang, and Y. He, "3d gaussian as a new vision era: A survey," *arXiv preprint arXiv:2402.07181*, 2024.

[12] M. T. Bagdasarian, P. Knoll, F. Barthel, A. Hilsmann, P. Eisert, and W. Morgenstern, "3dgs.zip: A survey on 3d gaussian splatting compression methods," 2024. [Online]. Available: https://arxiv.org/abs/2407.09510

[13] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park, "Compact 3d gaussian representation for radiance field," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[14] S. Niedermayr, J. Stumpfegger, and R. Westermann, "Compressed 3d gaussian splatting for accelerated novel view synthesis," *arXiv preprint arXiv:2401.02436*, 2023.

[15] S. Girish, K. Gupta, and A. Shrivastava, "Eagles: Efficient accelerated 3d gaussians with lightweight encodings," in *European Conference on Computer Vision*, 2024.

[16] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps," *Advances in neural information processing systems*, 2024.

[17] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Learned image compression with discretized gaussian mixture likelihoods and attention modules," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 7939–7948.

[18] D. He, Y. Zheng, B. Sun, Y. Wang, and H. Qin, "Checkerboard context model for efficient learned image compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 771–14 780.

[19] T. Lu, M. Yu, L. Xu, Y. Xiangli, L. Wang, D. Lin, and B. Dai, "Scaffold-gs: Structured 3d gaussians for view-adaptive rendering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[20] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[21] Y. Chen, Q. Wu, W. Lin, M. Harandi, and J. Cai, "Hac: Hash-grid assisted context for 3d gaussian splatting compression," in *European Conference on Computer Vision*, 2024.

[22] C. Sun, M. Sun, and H.-T. Chen, "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5459–5469.

[23] L. Li, Z. Shen, Z. Wang, L. Shen, and L. Bo, "Compressing volumetric radiance fields to 1 mb," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4222–4231.

[24] C. L. Deng and E. Tartaglione, "Compressing explicit voxel grid representations: fast nerfs become also small," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 1236–1245.

[25] L. Li, Z. Wang, Z. Shen, L. Shen, and P. Tan, "Compact real-time radiance fields with neural codebook," in *ICME*, 2023.

[26] S. Shin and J. Park, "Binary radiance fields," *Advances in neural information processing systems*, 2023.

[27] S. Girish, A. Shrivastava, and K. Gupta, "Shacira: Scalable hash-grid compression for implicit neural representations," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 513–17 524.

[28] D. Rho, B. Lee, S. Nam, J. C. Lee, J. H. Ko, and E. Park, "Masked wavelet representation for compact neural radiance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 20 680–20 690.

[29] J. Tang, X. Chen, J. Wang, and G. Zeng, "Compressible-composable nerf via rank-residual decomposition," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 798–14 809, 2022.

[30] Z. Song, W. Duan, Y. Zhang, S. Wang, S. Ma, and W. Gao, "Spc-nerf: Spatial predictive compression for voxel based radiance field," *arXiv preprint arXiv:2402.16366*, 2024.

[31] Y. Chen, Q. Wu, M. Harandi, and J. Cai, "How far can we compress instant-ngp-based nerf?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[32] S. Li, H. Li, Y. Liao, and L. Yu, "Nerfcodec: Neural feature compression meets neural radiance fields for memory-efficient scene representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 21 274–21 283.

[33] G. Kang, Y. Lee, and E. Park, "Codecnerf: Toward fast encoding and decoding, compact, and high-quality novel-view synthesis," *arXiv preprint arXiv:2404.04913*, 2024.

[34] T. Wu, Y.-J. Yuan, L.-X. Zhang, J. Yang, Y.-P. Cao, L.-Q. Yan, and L. Gao, "Recent advances in 3d gaussian splatting," *Computational Visual Media*, vol. 10, no. 4, pp. 613–642, 2024.

[35] K. Navaneet, K. P. Meibodi, S. A. Koohpayegani, and H. Pirsiavash, "Compact3d: Compressing gaussian splat radiance field models with vector quantization," in *European Conference on Computer Vision*, 2024.

[36] P. Papantonakis, G. Kopanas, B. Kerbl, A. Lanvin, and G. Drettakis, "Reducing the memory footprint of 3d gaussian splatting," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 7, no. 1, pp. 1–17, 2024.

[37] M. S. Ali, M. Qamar, S.-H. Bae, and E. Tartaglione, "Trimming the fat: Efficient compression of 3d gaussian splats through pruning," *arXiv preprint arXiv:2406.18214*, 2024.

[38] W. Morgenstern, F. Barthel, A. Hilsmann, and P. Eisert, "Compact 3d scene representation via self-organizing gaussian grids," *arXiv preprint arXiv:2312.13299*, 2023.

[39] H. Wang, H. Zhu, T. He, R. Feng, J. Deng, J. Bian, and Z. Chen, "End-to-end rate-distortion optimized 3d gaussian representation," in *European Conference on Computer Vision*. Springer, 2025, pp. 76–92.

[40] R. Yang, Z. Zhu, Z. Jiang, B. Ye, X. Chen, Y. Zhang, Y. Chen, J. Zhao, and H. Zhao, "Spectrally pruned gaussian fields with neural compensation," *arXiv preprint arXiv:2405.00676*, 2024.

[41] G. Fang and B. Wang, "Mini-splatting: Representing scenes with a constrained number of gaussians," in *European Conference on Computer Vision*, 2024.

[42] M. Wu and T. Tuytelaars, "Implicit gaussian splatting with efficient multi-level tri-plane representation," *arXiv preprint arXiv:2408.10041*, 2024.

[43] Y. Wang, Z. Li, L. Guo, W. Yang, A. C. Kot, and B. Wen, "Contextgs: Compact 3d gaussian splatting with anchor level context model," *Advances in neural information processing systems*, 2024.

[44] X. Liu, X. Wu, P. Zhang, S. Wang, Z. Li, and S. Kwong, "Compgs: Efficient 3d scene representation via compressed gaussian splatting," in *Proceedings of the 32nd ACM International Conference on Multimedia*, 2024, pp. 2936–2944.

[45] T. M. Cover, *Elements of information theory*. John Wiley & Sons, 1999.

[46] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," in *International Conference on Learning Representations*, 2018.

[47] Y. Chen, Q. Wu, M. Li, W. Lin, M. Harandi, and J. Cai, "Fast feedforward 3d gaussian splatting compression," *arXiv preprint arXiv:2410.08017*, 2024.

[48] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[49] A. Chen, S. Mao, Z. Li, M. Xu, H. Zhang, D. Niyato, and Z. Han, "An introduction to point cloud compression standards," *GetMobile: Mobile Computing and Communications*, vol. 27, no. 1, pp. 11–17, 2023.

[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[51] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, pp. 1–13, 2017.

[52] S. Xie, W. Zhang, C. Tang, Y. Bai, R. Lu, S. Ge, and Z. Wang, "Mesongs: Post-training compression of 3d gaussians via efficient attribute transformation," in *European Conference on Computer Vision*. Springer, 2025, pp. 434–452.
[53] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Transactions on Graphics (ToG)*, vol. 37, no. 6, pp. 1–15, 2018.
[54] G. Bjontegaard, "Calculation of average psnr differences between rd-curves," *ITU SG16 Doc. VCEG-M33*, 2001.
[55] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
[56] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 586–595.

Yihang Chen received the BS degree from Electronic Information Engineering, Dalian University of Technology, in 2021. He is currently pursuing the Ph.D. degree with the joint program of Shanghai Jiao Tong University and Monash University. His current research interests include 3D vision and data compression techniques.

Qianyi Wu received the BSc and MSc degrees from the Special Class for the Gifted Youth and School of Mathematical Sciences, University of Science and Technology of China in 2016 and 2019, respectively. He is currently pursuing the Ph.D. degree with the Faculty of Information Technology, the Monash University. His current research interest includes computer vision and computer graphics.

Weiyao Lin (Senior Member, IEEE) received the B.E. and M.E. degrees from Shanghai Jiao Tong University, Shanghai, China, in 2003 and 2005, respectively, and the Ph.D. degree from the University of Washington, Seattle, WA, USA, in 2010, all in electrical engineering. He is currently a Professor with the Department of Electronic Engineering, Shanghai Jiao Tong University. He has authored or coauthored more than 100 technical articles on top journals/conferences including the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, International Journal of Computer Vision, the IEEE TRANSACTIONS ON IMAGE PROCESSING, CVPR, and ICCV. He holds more than 20 patents. His research interests include video/image analysis, computer vision, and video/image processing applications .

Mehrtash Harandi is an associate professor with the Department of Electrical and Computer Systems Engineering at Monash University. He is also a contributing research scientist in the Machine Learning Research Group (MLRG) at Data61/CSIRO. His current research interests include theoretical and computational methods in machine learning, computer vision, and Riemannian geometry.

Jianfei Cai (S'98-M'02-SM'07-F'21) received his PhD degree from the University of Missouri-Columbia. He is currently a Professor at Faculty of IT, Monash University, where he had served as the inaugural Head for the Data Science & AI Department. Before that, he was Head of Visual and Interactive Computing Division and Head of Computer Communications Division in Nanyang Technological University (NTU). His major research interests include computer vision, deep learning and multimedia. He is a co-recipient of paper awards in ACCV, ICCM, IEEE ICIP and MMSP, and the winner of Monash FIT's Dean's Researcher of the Year Award. He is currently on the editorial board of TPAMI and IJCV. He has served as an Associate Editor for IEEE T-IP, T-MM, and T-CSVT as well as serving as Area Chair for CVPR, ICCV, ECCV, ACM Multimedia, IJCAI, ICME, ICIP, and ISCAS. He was the Chair of IEEE CAS VSPC-TC during 2016-2018. He had served as the leading TPC Chair for IEEE ICME 2012 and the best paper award committee chair & co-chair for IEEE T-MM 2020 & 2019. He is the leading General Chair for ACM Multimedia 2024, and a Fellow of IEEE.

–Appendix–

In this Appendix, we present per-scene results of our HAC++ method across multiple fidelity metrics (*i.e.*, PSNR, SSIM [55] and LPIPS [56]) over all datasets, as shown from Tab. A to Tab. E.

TABLE A
HAC++'S RESULTS ON THE SYNTHETIC-NERF DATASET [1] FOR DIFFERENT λ VALUES.

λ	Scenes	PSNR ↑	SSIM ↑	LPIPS ↓	Size (MB) ↓
$4e-3$	chair	33.90	0.980	0.021	0.65
	drums	26.17	0.950	0.045	0.90
	ficus	34.53	0.984	0.016	0.64
	hotdog	36.53	0.979	0.035	0.45
	lego	34.33	0.976	0.027	0.80
	materials	30.22	0.959	0.046	0.87
	mic	35.21	0.989	0.012	0.44
	ship	31.26	0.903	0.125	1.30
$3e-3$	chair	34.40	0.982	0.018	0.76
	drums	26.27	0.951	0.043	1.20
	ficus	34.74	0.984	0.015	0.70
	hotdog	36.87	0.980	0.032	0.54
	lego	34.64	0.977	0.025	0.97
	materials	30.35	0.960	0.044	0.97
	mic	35.64	0.990	0.010	0.50
	ship	31.34	0.903	0.124	1.42
$2e-3$	chair	34.79	0.984	0.016	0.94
	drums	26.25	0.951	0.043	1.25
	ficus	34.80	0.985	0.014	0.84
	hotdog	37.27	0.982	0.028	0.60
	lego	35.10	0.979	0.022	1.08
	materials	30.49	0.961	0.041	1.15
	mic	36.05	0.991	0.009	0.61
	ship	31.43	0.903	0.119	1.88
$1e-3$	chair	35.34	0.986	0.014	1.21
	drums	26.41	0.952	0.041	1.79
	ficus	35.21	0.986	0.013	1.11
	hotdog	37.69	0.983	0.025	0.78
	lego	35.52	0.981	0.019	1.49
	materials	30.64	0.962	0.040	1.48
	mic	36.51	0.991	0.008	0.79
	ship	31.52	0.905	0.114	2.55
$0.5e-3$	chair	35.60	0.986	0.012	1.64
	drums	26.48	0.952	0.041	2.47
	ficus	35.25	0.986	0.013	1.42
	hotdog	37.89	0.984	0.023	0.98
	lego	35.77	0.982	0.018	1.83
	materials	30.71	0.962	0.038	1.90
	mic	36.79	0.992	0.008	1.09
	ship	31.54	0.904	0.111	3.43
$4e-3$	**AVG**	32.77	0.965	0.041	0.76
$3e-3$	**AVG**	33.03	0.966	0.039	0.88
$2e-3$	**AVG**	33.27	0.967	0.037	1.04
$1e-3$	**AVG**	33.60	0.968	0.034	1.40
$0.5e-3$	**AVG**	33.76	0.969	0.033	1.84

TABLE B
HAC++'S RESULTS ON THE DEEPBLENDING DATASET [53] FOR DIFFERENT λ VALUES.

λ	Scenes	PSNR ↑	SSIM ↑	LPIPS ↓	Size (MB) ↓
$4e-3$	playroom	30.69	0.909	0.268	2.42
	drjohnson	29.63	0.905	0.264	3.40
$3e-3$	playroom	30.82	0.911	0.263	2.78
	drjohnson	29.75	0.906	0.260	3.91
$2e-3$	playroom	30.79	0.911	0.262	3.29
	drjohnson	29.73	0.907	0.257	4.61
$1e-3$	playroom	30.93	0.913	0.255	4.35
	drjohnson	29.76	0.908	0.253	6.21
$4e-3$	**AVG**	30.16	0.907	0.266	2.91
$3e-3$	**AVG**	30.28	0.909	0.262	3.34
$2e-3$	**AVG**	30.26	0.909	0.260	3.95
$1e-3$	**AVG**	30.34	0.911	0.254	5.28

TABLE C
HAC++'S RESULTS ON THE TANK&TEMPLES DATASET [51] FOR DIFFERENT λ VALUES.

λ	Scenes	PSNR ↑	SSIM ↑	LPIPS ↓	Size (MB) ↓
$4e-3$	train	25.89	0.883	0.156	5.67
	truck	22.54	0.815	0.225	4.69
$3e-3$	train	25.98	0.884	0.153	6.53
	truck	22.57	0.818	0.219	5.25
$2e-3$	train	26.06	0.886	0.148	7.64
	truck	22.60	0.821	0.213	6.21
$1e-3$	train	26.05	0.887	0.147	9.71
	truck	22.58	0.821	0.210	7.55
$4e-3$	**AVG**	24.22	0.849	0.190	5.18
$3e-3$	**AVG**	24.28	0.851	0.186	5.89
$2e-3$	**AVG**	24.33	0.853	0.181	6.92
$1e-3$	**AVG**	24.32	0.854	0.178	8.63

TABLE D
HAC++'S RESULTS ON THE MIP-NERF360 DATASET [9] FOR DIFFERENT λ VALUES.

λ	Scenes	PSNR ↑	SSIM ↑	LPIPS ↓	Size (MB) ↓
$4e-3$	bicycle	25.08	0.732	0.289	12.84
	garden	27.19	0.832	0.179	12.83
	stump	26.56	0.757	0.289	9.40
	room	31.75	0.920	0.217	3.41
	counter	29.37	0.910	0.203	4.69
	kitchen	31.20	0.923	0.137	5.26
	bonsai	32.66	0.944	0.190	5.35
	flower	21.26	0.569	0.393	10.68
	treehill	23.31	0.638	0.379	10.57
$3e-3$	bicycle	25.09	0.734	0.285	14.36
	garden	27.21	0.835	0.172	14.37
	stump	26.61	0.758	0.283	10.47
	room	31.83	0.921	0.213	3.84
	counter	29.48	0.912	0.199	5.28
	kitchen	31.34	0.924	0.134	6.01
	bonsai	32.73	0.946	0.188	5.86
	flower	21.28	0.570	0.390	11.99
	treehill	23.32	0.640	0.375	11.96
$2e-3$	bicycle	25.22	0.738	0.276	17.24
	garden	27.36	0.842	0.159	17.22
	stump	26.62	0.761	0.277	12.29
	room	32.00	0.924	0.206	4.54
	counter	29.64	0.915	0.193	6.26
	kitchen	31.51	0.927	0.129	7.14
	bonsai	32.88	0.947	0.185	6.83
	flower	21.22	0.571	0.385	14.42
	treehill	23.26	0.642	0.366	14.65
$1e-3$	bicycle	25.04	0.740	0.268	23.09
	garden	27.44	0.847	0.147	22.14
	stump	26.60	0.762	0.269	16.47
	room	32.04	0.927	0.199	5.99
	counter	29.71	0.917	0.188	8.06
	kitchen	31.70	0.930	0.125	9.23
	bonsai	33.06	0.949	0.181	8.84
	flower	21.31	0.575	0.378	19.03
	treehill	23.30	0.646	0.352	19.72
$0.5e-3$	bicycle	25.05	0.738	0.268	29.25
	garden	27.49	0.850	0.140	27.22
	stump	26.52	0.761	0.265	21.02
	room	32.05	0.928	0.196	7.55
	counter	29.81	0.919	0.183	9.73
	kitchen	31.81	0.931	0.122	11.37
	bonsai	33.17	0.950	0.179	11.15
	flower	21.28	0.576	0.375	24.19
	treehill	23.20	0.645	0.349	24.85
$4e-3$	**AVG**	27.60	0.803	0.253	8.34
$3e-3$	**AVG**	27.65	0.805	0.249	9.35
$2e-3$	**AVG**	27.74	0.808	0.242	11.18
$1e-3$	**AVG**	27.80	0.810	0.234	14.73
$0.5e-3$	**AVG**	27.82	0.811	0.231	18.48

TABLE E
HAC++'S RESULTS ON THE BUNGEENERF DATASET [8] FOR DIFFERENT λ VALUES.

λ	Scenes	PSNR ↑	SSIM ↑	LPIPS ↓	Size (MB) ↓
$4e-3$	amsterdam	26.85	0.871	0.220	14.05
	bilbao	27.92	0.881	0.206	11.02
	hollywood	24.30	0.753	0.348	11.22
	pompidou	25.35	0.846	0.244	12.49
	quebec	30.00	0.930	0.172	9.84
	rome	26.25	0.865	0.222	11.91
$3e-3$	amsterdam	27.03	0.879	0.206	16.01
	bilbao	28.09	0.887	0.191	12.59
	hollywood	24.51	0.766	0.334	12.76
	pompidou	25.55	0.852	0.235	14.81
	quebec	30.19	0.934	0.164	11.39
	rome	26.39	0.872	0.210	13.61
$2e-3$	amsterdam	27.22	0.888	0.189	19.35
	bilbao	28.12	0.893	0.179	15.14
	hollywood	24.71	0.784	0.312	15.37
	pompidou	25.70	0.858	0.227	17.39
	quebec	30.35	0.937	0.155	13.47
	rome	26.67	0.881	0.199	16.07
$1e-3$	amsterdam	27.29	0.896	0.169	25.20
	bilbao	27.89	0.894	0.169	19.58
	hollywood	24.84	0.797	0.290	19.48
	pompidou	25.73	0.860	0.219	22.57
	quebec	30.51	0.941	0.146	17.49
	rome	26.78	0.887	0.184	20.58
$4e-3$	**AVG**	26.78	0.858	0.235	11.75
$3e-3$	**AVG**	26.96	0.865	0.223	13.53
$2e-3$	**AVG**	27.13	0.873	0.210	16.13
$1e-3$	**AVG**	27.17	0.879	0.196	20.82