Frequency-Adaptive Gaussian Grids for Efficient Image Representation

Anatolii Evdokimov[1] Stavros Diolatzis[2] Bingxuan Li[1] Akshay Jindal[2]
Patsorn Sangkloy[1] Qi Sun[1]

[1]New York University [2]Intel Corporation

{tolya.e, bingxuan.li, ps5688, qisun}@nyu.edu {stavros.diolatzis, akshay.jindal}@intel.com

Abstract

Neural representation has emerged as a powerful tool to mitigate the demand for increasingly higher-resolution images. However, a fundamental trade-off remains between structured (e.g., feature grids, which offer spatial consistency) and unstructured (e.g., 2D Gaussians, which adaptively capture local variances) representations. This choice significantly affects how well an algorithm represents an image. Here, we present Gaussian-Grid, a hybrid neural representation that adaptively integrates structured and unstructured primitives. By dynamically partitioning computation between 2D Gaussian distributions and their attached multi-resolution feature grids based on the local frequency content, Gaussian-Grid automatically optimizes its architecture to match the underlying complexity of the target signal. We evaluate our approach using high-resolution natural images and demonstrate its applications as GPU-native dataset storage, showing 1.24–$2.14\times$ *decoding speed improvement and PCIe data traffic reduction of* $23\times$–$176\times$ *over JPEG + ZIP data archiving. The results show that Gaussian-Grid maintains small memory footprints while achieving high reconstruction fidelity across a variety of datasets.*

1. Introduction

Representing images with high fidelity and low memory footprint is foundational in GPU systems and computer vision applications, especially with the recent surging demand for ultra-high-quality image data. Traditional grid-based representations scale quadratically in memory, creating a bottleneck where discrete sampling rates fail to maintain visual quality due to storage and bandwidth constraints.

To tackle the scaling limitations, neural representations have emerged over the past decade as an efficient alternative by learning image signals using neural networks. Among these approaches, images and texture stacks may be represented through a combination of structured feature grids [15, 16, 30] and neural networks to represent localized spatial signals. Alternatively, unstructured explicit primitives such as 2D Gaussians [38, 40] enable spatially adaptive representations that optimize resource allocation to where it is warranted. However, both directions have pros and cons: structured grids prioritize memory efficiency by implicitly defining feature distributions but suffer from limited spatial flexibility, whereas primitive-based methods offer better adaptivity at the cost of higher overhead for complex local details.

Here, we introduce Gaussian-Grid, a hybrid representation that integrates both structured and unstructured neural representations. By unifying the spatial flexibility of primitives with the efficient representation power of feature grids, Gaussian-Grid achieves a balance between fidelity and memory efficiency for high-resolution imagery across a wide range of content types. Our main insight is to represent images as collections of 2D Gaussians that each carry an embedded feature grid. In this architecture, the Gaussians act as a substrate, optimizing for global geometric structures and automatically anchoring high-capacity feature grids only where local signal complexity demands them.

To this aim, we design a two-stage algorithm composed of a frequency-guided framework to decide the Gaussian distribution and a final grid optimization step. The transition from the frequency-guided Gaussians to the grid optimization happens by attaching feature grids of appropriate resolution based on the local complexity represented by each Gaussian. Finally, these features are decoded through a lightweight multilayer perceptron (MLP).

We evaluate Gaussian-Grid across a variety of realistic and high-resolution images adapted from DIV2K [1] and CLIC2020 [29]. The results demonstrate that Gaussian-Grid robustly handles images of varying frequency content due to its ability to switch from a structured to an unstructured representation. As such, our method outperforms structured neural representations for concentrated frequency content and primitive based representations for

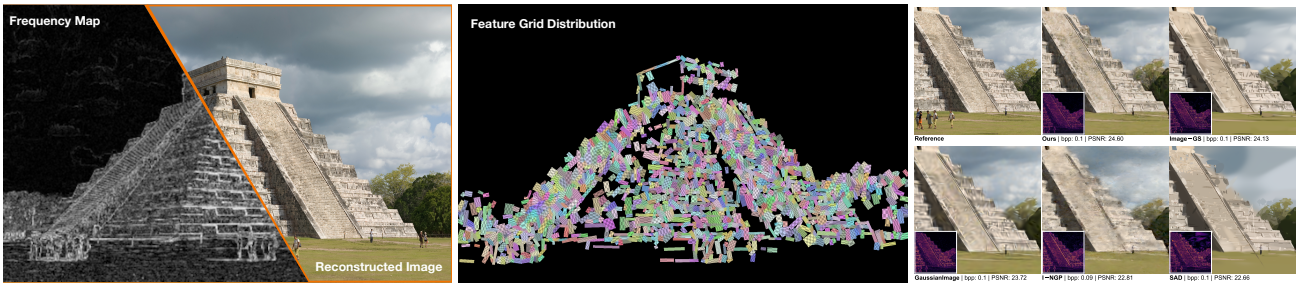


Figure 1. Gaussian-Grid leverages Gaussian-defined feature grid layout that is guided by image frequency map to generate high quality reconstruction.

uniform frequency content. We further demonstrate that our method serves as a GPU-native image dataset archiver that bypasses the CPU-dependent decompression overhead of popular ZIP and JPEG solutions, achieving 1.24–$2.14\times$ decoding speed improvement and PCIe data traffic reduction of $23\times$–$176\times$ over JPEG + ZIP data archiving. Our source code is released at https://github.com/NYU-ICL/gaussian-grid.

2. Related Work

Prior work on image representation can be broadly categorized into traditional block-based compression, structured neural representations based on regular grids, and unstructured neural representations that use continuous functions or learned primitives. These approaches exhibit different trade-offs between spatial adaptivity, memory efficiency, and decoding cost. Our work draws inspiration from all three lines while addressing their respective limitations in representing spatially heterogeneous high-frequency content.

2.1. Traditional Representations

Traditional techniques are designed to satisfy the constraints of real-time graphics hardware, including fast decoding, low memory bandwidth, and constant-time random access [2]. As a result, most formats adopt fixed-rate, block-based representations that compress small spatial regions independently. Early methods such as block truncation coding [8] introduced local approximation using a small number of representative values per block, an idea later extended to color textures [4] and standardized in widely deployed GPU compression formats. Subsequent formats added support for alpha channels, normal maps, and high dynamic range content [17, 20], while more recent standards such as ASTC allow variable block sizes and bitrates [11, 18, 25, 26]. These designs improve compression quality and flexibility but remain constrained by fixed block layouts and minimum per-block bitrates, which cap achievable compression ratios. Moreover, capacity is distributed uniformly across blocks, preventing adaptation to spatially varying signal complexity. High-frequency details and sharp transitions are forced to share the same bit budget as smooth regions, leading to inefficient allocation and visible artifacts at low bitrates. These limitations motivate neural representations that move beyond block-based compression and enable more flexible trade-offs between memory usage, reconstruction quality, and spatial capacity allocation.

2.2. Structured Neural Representation

Structured neural representations encode signals using spatially aligned feature grids coupled with lightweight neural decoders, enabling efficient storage and fast evaluation. Early work demonstrated that dense and multi-resolution grids can significantly reduce memory usage while maintaining high reconstruction quality [6, 15]. Subsequent methods introduced hash-based and learned indexing schemes and support for discontinuities to further improve scalability and inference performance [3, 16, 27, 28].

These approaches have recently been applied to texture compression and appearance representation [28]. Neural Texture Compression (NTC) [30], for example, encodes multi-channel textures and mipmap chains using uniform feature grids and a lightweight decoder, achieving strong compression ratios and fast random access. Extensions include block-compressed features [35] and differentiable indirection structures [7] to improve scalability for large textures.

Despite their efficiency, structured grid-based representations allocate capacity uniformly across space. Localized high-frequency details are therefore supported everywhere in the grid, resulting in over-allocation in smooth regions and limited adaptation to spatially heterogeneous frequency distributions. As a consequence, pure grid-based methods are less effective for high-resolution textures with concentrated or non-uniform detail.

2.3. Unstructured Neural Representation

Unstructured neural representations depart from spatially regular parameterizations and instead model signals as con-

tinuous functions. Coordinate-based implicit neural representations (INRs) map spatial coordinates directly to signal values using multilayer perceptrons with positional or sinusoidal encodings [9, 10, 23, 24, 32]. These methods are highly expressive and resolution-independent but often suffer from slow inference due to dense per-query network evaluation and limited scalability for large images or multi-channel textures.

An alternative class of unstructured representations uses explicit learnable primitives, most notably Gaussian functions. Gaussian-based methods represent images as collections of localized primitives parameterized by position, shape, and appearance, and reconstruct signals by aggregating their contributions [36, 38, 40]. An alternative method using soft anisotropic diagrams as a primitive has also been recently proposed [13]. By adjusting primitive density and size, these approaches can adaptively allocate capacity to complex regions while keeping smooth areas compact. While Gaussian primitives offer strong spatial adaptivity, each primitive introduces non-trivial parameter overhead. For dense textures or high-capacity representations, the total number of primitives required to model fine details can grow rapidly, reducing efficiency. This trade-off motivates hybrid approaches that combine the adaptive placement of unstructured primitives with the compactness and regularity of structured feature grids.

2.4. Hybrid Neural Representations

Hybrid neural representations combine structured parameterizations with unstructured or continuous components to balance efficiency and expressiveness. ACORN [15] introduces content-adaptive grid density through a hierarchical block decomposition coupled with a lightweight decoder, demonstrating that allocating finer grids to complex regions improves large-scale image representation. However, ACORN relies on axis-aligned grids, limiting its efficiency for anisotropic or oriented structures such as edges and textured patterns.

In 3D reconstruction, recent hybrid approaches augment explicit primitives with spatially varying detail. Textured Gaussian methods [5, 33] extend 3D Gaussian Splatting by attaching local texture maps to individual Gaussians, enabling high-frequency appearance modeling within each primitive's support. Related frequency-adaptive NeRF [21, 39] and Gaussian-based methods [19] similarly allocate capacity based on local scene complexity. While these ideas align with our goal of distributing capacity according to signal frequency, they have largely been explored in 3D settings. Their application to purely 2D image representation and compression remains underexplored. To address this gap, Gaussian-Grid combines frequency-adaptive Gaussian placement with localized feature grids, enabling fine-grained capacity allocation while maintaining efficient storage and decoding for high-resolution images.

3. Method

The core contribution of our method is a hybrid representation that integrates structured feature grids with unstructured Gaussian primitives. We achieve this by adaptively adjusting representation capacity based on the local frequency content of an image, allocating more parameters to high-frequency areas and fewer to low-frequency ones.

Specifically, we optimize learnable 2D Gaussians to match a local frequency map derived from the target image. These Gaussians provide the spatial layout for attaching high-capacity feature grids only where they are needed. For areas with lower frequency, a canvas grid reconstructs the smooth, low-frequency content. Finally, an MLP decoder (MLP_{dec}) decodes the latent grid features per image. Figure 2 shows an overview of our proposed method.

3.1. Gaussian Frequency Distribution Map

First, we establish a ground truth frequency distribution to guide the optimization of the Gaussians. For each spatial window of size 8×8 centered at pixel (i, j), we compute the local frequency value $F(i, j)$ as

$$F(i,j) = \max_{\mathbf{u} \neq 0} \left| \mathcal{F}\left[I_{(i,j)}\right](\mathbf{u}) \right|, \tag{1}$$

where $I_{(i,j)}$ is the windowed image patch , $\mathcal{F}$ denotes the 2D discrete Fourier transform (DFT), $\mathbf{u}$ indexes the frequency components. The DC term is excluded from the maximum calculation to ensure invariance to local brightness variations.

For Step 1 of our optimization, we represent this frequency map as a set of N learnable 2D anisotropic Gaussians. We parametrize each Gaussian $\mathcal{G}_n$ by its center μ_n, a diagonal covariance matrix $\Sigma_n = diag(\sigma_{(i,n)}, \sigma_{(j,n)})$, a scalar frequency assignment f_n, and a rotation angle θ_n. The Gaussian distribution of the n-th Gaussian at coordinate $\mathbf{x} = [i, j]$ is defined by the equation:

$$\mathcal{G}_n(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_n)^T R_n \Sigma_n^{-1} R_n^T (\mathbf{x}-\mu_n)\right), \tag{2}$$

where R_n is the 2D rotation matrix for angle θ_n.

The final pixel value f_r is rendered by accumulating each Gaussian's frequency assignment contributions for each coordinate $\mathbf{x}$. Specifically, for each non-overlapping tile T_l, we determine which Gaussians overlap with the tile in a 3-standard-deviation enclosing circle range (denoted as $\mathcal{S}_l$) and render only the contributions of the overlapping Gaussians for that T_l. This allows us to efficiently consider all Gaussians overlapping a tile.

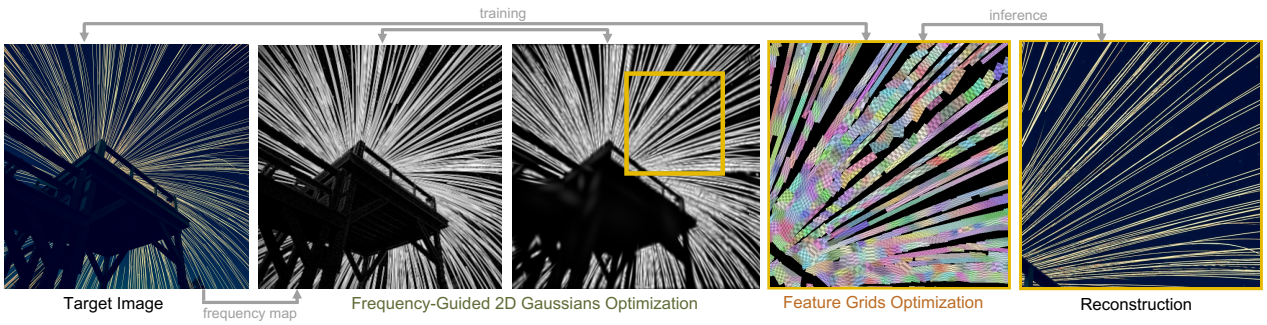


Figure 2. Overview of our method. Gaussian-Grid optimizes 2D Gaussian primitives to a spatial frequency map derived from the target image. Gaussians are then used to place feature grids of variable resolutions, aligning with the underlying frequency of the content. In the final stage, the feature grids are optimized to reconstruct the final image.

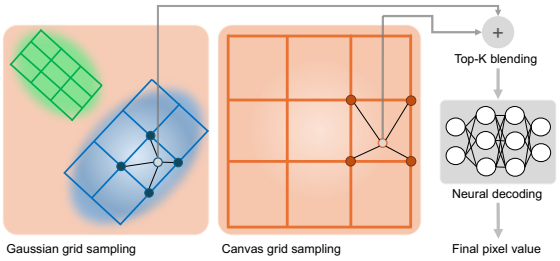


Figure 3. Pixel Rendering. For a given pixel location, Gaussian-Grid bilinearly samples top-K overlapping Gaussian grids as well as the canvas grid. The interpolated features are decoded by a small MLP_{dec} to render the final pixel value.

The final formula for Gaussian frequency map generation is:

$$f_r = \sum_{n \in \mathcal{S}_l(\mathbf{x})} \mathcal{G}_n(\mathbf{x}) \cdot f_n. \tag{3}$$

Figure 2 includes an example visualization of the frequency and Gaussian maps.

3.2. Grid Resolution and Placement

Once the Gaussians are optimized to match the frequency distribution of the given image, we switch to the grid optimization (Step 2). We place grids of appropriate resolution to match the position and shape of each Gaussian. Specifically, the grid budget is determined using the learned f_n value and the pixel coverage of each Gaussian. We compute the weight w_n for each Gaussian grid that determines the proportion of the overall bit budget assigned to that grid:

$$w_n = \frac{|f_n|}{\max_j |f_j|} \cdot 4\sigma_{x,n} \cdot 4\sigma_{y,n} \tag{4}$$

where $\sigma_{x,n}$ and $\sigma_{y,n}$ are the Gaussian scales (the factor of 4 gives a bounding box of $\pm 2\sigma$). This way, larger Gaussians with higher frequency magnitude f_n receive more features, while lower-complexity regions receive fewer. We then refine this allocation using directional analysis of the local frequency field.

For each Gaussian, we gather the per-pixel frequency vectors $\mathbf{v}_p = (F_x(i,j), F_y(i,j))$ assigned to it and compute their magnitude-weighted directional variance:

$$\text{Var}_n = \frac{\sum_p \|\mathbf{v}_p\| \left\|\hat{\mathbf{v}}_p - \bar{\mathbf{d}}_n\right\|^2}{\sum_p \|\mathbf{v}_p\|}, \tag{5}$$

where $\hat{\mathbf{v}}_p = \mathbf{v_p}/||\mathbf{v_p}||$ is the direction and $\bar{\mathbf{d}}_n$ is the normalized mean direction for Gaussian $\mathcal{G}_n$. We increase the budget for Gaussians with mixed directional content and reduce it for regions with simple, consistent directionality. For each Gaussian, the assigned grid cell count is finally split into height and width according to Gaussian anisotropy, with an additional directional aspect-ratio correction.

In addition to local Gaussian grids, we include a global canvas grid spanning the full image to provide stable low-frequency background coverage. Its budget proportion is 30% by default but adapted per image via a score $s \in [0,1]$ derived from the coefficient of variation of F and a global edge term (see supplemental for the full formula). Images with flat, homogeneous content score $s \approx 1$ and receive a larger canvas budget and heavier Gaussian pruning; textured images score $s \approx 0$ and retain dense Gaussian coverage with a smaller canvas budget.

During rendering, each pixel queries its top-K Gaussian grids, sampling features from each local grid in the Gaussian's own coordinate frame. Samples outside a grid are marked invalid, and the final feature is blended from valid contributors only.

3.3. Grid Sampling and Interpolation

For every Gaussian selected by top-K, we bilinearly sample the corresponding local feature grid. The per-Gaussian sampled features are then fused by normalized masked averaging across the valid top-K contributors. The global canvas grid is also sampled for every pixel and added as a

low-frequency background component during the blending stage. The blended feature vector is finally passed through a small 2-layer MLP_{dec} to predict the output signal. Figure 3 summarizes our rendering pipeline.

3.4. Quantization

To achieve high representation compactness while maintaining signal fidelity, our model employs a quantization-aware training strategy (QAT) inspired by NTC [30]. During training, both grid features and MLP_{dec} computations are passed through a simulated quantization pipeline.

For a given feature grid tensor and a target bit-depth B, we first compute a per-tensor scaling factor $s = \frac{\max(|X|)}{2^B-1}$. The floating-point tensor X is then scaled yielding the quantized tensor X_q. We also add uniform random noise n in the range $n \sim U(-0.5, 0.5)$:

$$X_q = \text{round}(\frac{X}{s} + n) \tag{6}$$

The resulting values are clamped to the integer range $[-2^{B-1}, 2^{B-1} - 1]$.

For the MLP_{dec}, we apply a per-channel quantization scheme to its weights, where a separate scale is calculated for each output channel of the linear layers. The QAT procedure is applied during the first 95% of the training time, while for the last 5%, the grid features are hard-quantized and frozen, allowing the MLP_{dec} to adapt to fully quantized low-bit grids.

3.5. Implementation Details

Gaussian-Grid is trained in a two-stage process: first, the 2D Gaussian optimization and, second, the grid optimization.

In the first stage, we set learning rates for (μ, σ, θ, f) to (1e-2, 1e-3, 1e-3, 1e-2). The number of initial Gaussians is set per budget: 800, 1500, 2000, and 2300 for 0.1, 0.4, 0.7, and 1.0 bpp on DIV2K, and 400, 900, 1200, and 1500 on CLIC2020. The loss function for the first stage is:

$$\mathcal{L}_1 = \mathcal{L}_{L1} + \lambda_{\text{ov}} \cdot \mathcal{L}_{\text{ov}}, \tag{7}$$

where $\mathcal{L}_{L1}$ is the L1 loss between the predicted and the target frequency maps and the overlap penalty $\mathcal{L}_{\text{ov}}$ discourages large spatial overlap between Gaussians. For each pair of Gaussians i and j, we compute a Gaussian overlap using the pairwise Mahalanobis distance under the summed covariances:

$$\mathcal{L}_{\text{ov}} = \sum_{i<j} \frac{1}{2\pi} \exp\left(-\frac{1}{2}(\mu_i - \mu_j)^\top (\Sigma_i + \Sigma_j)^{-1} (\mu_i - \mu_j) \right), \tag{8}$$

where Σ is the full covariance matrix of a Gaussian. This penalty is summed over all unique pairs and encourages Gaussians to spread apart and tile the image without excessive redundancy. We set the loss weight $\lambda_{\text{ov}} = 5 \times 10^{-5}$.

After the optimization of 2D Gaussians, we begin training the feature grids against the RGB image using an L2 loss between the predicted color and the ground truth. Each pixel queries its top-K=5 grids, sampling features from each local grid in the Gaussian's coordinate frame. The decoder MLP_{dec} has 2 hidden layers of width 64 and each grid feature has dimension 6, with learning rates of 1e-3 and 1e-2 respectively. We apply 8-bit quantization to MLP_{dec} and 4-bit quantization to the grids, and hard-quantize the Gaussian parameters to 16-bit floats at the start of the second stage. For both stages we use the Adam optimizer [14].

To ensure efficient training we implement performance-critical components using GPU kernels. The first-stage Gaussian optimization uses Taichi [12], while the second-stage reconstruction uses a custom CUDA kernel. Gaussians are optimized for 3K steps and the feature grids for 75K steps, yielding an average training time of 24.3 minutes per image on a single A100 GPU.

4. Evaluation

4.1. Experimental setup

Datasets We evaluate Gaussian-Grid's performance on the set of 100 images acquired from a validation subset of the DIV2K natural images dataset [1]. We provide an additional evaluation on the professional validation subset of the CLIC2020 benchmark [29], 41 images in total.

Metrics and baselines Following the methodology of prior studies [40], we evaluate reconstruction quality across various memory budgets, quantified in bits-per-pixel (bpp). We assess quality using standard error-based and perceptual metrics: PSNR, SSIM [34], MS-SSIM, and LPIPS [37]. We compare Gaussian-Grid to three recent primitive-based methods, [40] (**IMG-GS**), [38] (**GI**), and [13] (**SAD**). Additionally, we include a popular structured grid-based baseline, [16] (**I-NGP**).

4.2. Reconstruction Performance

As shown in Table 1, Gaussian-Grid achieves competitive results across all budget points on both DIV2K and CLIC2020. At the lowest budget (0.1 bpp), our method leads all baselines on PSNR by a clear margin, reaching 25.58 dB on DIV2K (+0.86 dB over the next-best SAD) and 27.70 dB on CLIC2020 (+0.75 dB), with LPIPS remaining competitive with IMG-GS at the lowest budget. At 0.4 bpp we match SAD on PSNR (29.36 dB) while outperforming all methods on SSIM, MS-SSIM, and LPIPS, demonstrating that the quality gain is not limited to pixel fidelity. At higher budgets (0.7–1.0 bpp) our method remains best or near-best across all metrics, achieving 32.55 dB PSNR and an LPIPS

of 0.1167 at 1.0 bpp on DIV2K — a 0.12 dB gain over SAD and an LPIPS reduction of 0.069 over the next-best GaussianImage. Results on CLIC2020 follow the same trend, with Gaussian-Grid ranking first on almost all four metrics at every budget point.

The quantitative improvements are reflected qualitatively in Figure 4 at 0.1 bpp. Perceptually, our method combines the adaptivity of unstructured methods such as IMG-GS without the "oil painting" artifacts that appear in SAD, IMG-GS, and GI in regions of missing detail. This is due to our method's use of local feature grids within each Gaussian footprint, which faithfully reproduces spatially coherent structure rather than relying solely on primitive color and opacity. I-NGP, by contrast, produces noise-like patterns at low budgets owing to hash-grid collisions. We provide additional qualitative examples in the supplemental material.

4.3. Ablation Study

We ablate our method through two complementary studies summarized in Table 2. The architecture ablation evaluates each component incrementally, starting from a structured uniform baseline constructed by skipping the 2D Gaussian optimization and placing a single uniform canvas grid over the entire image with 100% of the budget. All hyperparameters like feature grid dimension, MLP_{dec} parameters, and quantization are held fixed throughout; bit budget formerly split between Gaussians and canvas is consolidated into one uniform grid. Adding frequency-guided placement alone *reduces* performance (-1.22 dB PSNR), since each Gaussian receives equal grid budget regardless of its local frequency content, leaving low-frequency regions underserved. Budget weighting, which allocates grid resolution proportionally to each Gaussian's frequency energy, recovers this loss and surpasses the previous condition by +1.39 dB. Directional grid stretching along the dominant local frequency orientation contributes a further modest gain (+0.04 dB). The full method additionally applies adaptive canvas budget modulation and Gaussian pruning on a per-image basis, yielding +0.47 dB PSNR over the uniform baseline.

Notably, we observe that some numerical metrics (e.g., LPIPS) are marginally worse with our method than with the uniform placement condition. In fact, frequency-guided placement is designed to reallocate representation capacity from flat, low-frequency regions toward high-frequency textured regions, *thus preserving fine-grained details*. This trade-off is the source of the perceptual metric gap in Table 2. To further demonstrate the observation, Figure 5 shows a qualitative comparison. In the textured crop, our method improves over the uniform baseline on every metric including LPIPS, confirming that the reallocation succeeds where it is intended to. In the flat crop, canvas-only coverage introduces a slightly blurrier output, penalized by the perceptual metrics, which are sensitive to low-level texture statistics in smooth regions. We further discuss the limitation and future work for perceptually-aware budget allocation in Section 6.

The loss ablation isolates the contribution of the Step 1 overlap penalty. Training with $\mathcal{L}_{\text{L1}}$ alone produces placements with redundant, overlapping Gaussians that waste budget; adding $\mathcal{L}_{\text{ov}}$ penalizes this behavior and improves PSNR by +0.24 dB while simultaneously improving all perceptual metrics, confirming that better spatial coverage in Step 1 translates directly to reconstruction quality in Step 2.

5. Application: GPU-Native Dataset Archiver

Modern computer vision pipelines are fundamentally GPU-bound: training and inferring neural network models all require images loaded to GPU memory with local access for each GPU core. However, before images reach GPUs, our common practice is to unzip a large image dataset to the hard drive, decode the encoded image format (e.g., JPEG) into a decompressed pixel grid, and have them traverse the PCIe bus. Our neural image representation sidesteps this time-consuming bottleneck. Rather than the CPU-dependent data decoding to pixel values, our model-represented images can be loaded directly on the GPU. To demonstrate the application, we test the decoding efficiency of our method against JPEG [31] + ZIP combination on the DIV2K validation set across four budget targets. This setup closely resembles image dataset storage pipelines. All decode timings cover the full pipeline from a compressed file in the OS page cache: our path performs loading of the quantized file, GPU dequantization, and CUDA kernel execution; the JPEG path amortizes a one-time ZIP extraction (87–130 ms total, 0.87–1.30 ms per image) and then performs per-image JPEG decode and transfer to GPU. Both paths are benchmarked as the mean over three synchronized iterations after two warmup passes, with `torch.cuda.synchronize()` enforcing completion. We conduct all comparisons on NVIDIA RTX 5080 GPU. We summarize the results in Table 3. At 0.1 bpp, our method achieves $+2.33$ dB higher PSNR than size-matched JPEG ($\approx$ 47 KB per image), decodes in 5.44 ms versus 11.65 ms ($2.14\times$ faster), and transfers 47 KB over PCIe versus $8{,}306$ KB of decoded pixels, a $176\times$ reduction in bus traffic. At 0.4 bpp the quality gap narrows to $+0.31$ dB in our favour; above 0.4 bpp the advantage reverses (-0.53 to -1.00 dB at 0.7–1.0 bpp) as JPEG's entropy coder becomes more competitive at higher budgets. The decode speedup persists across all operating points (1.24–$2.14\times$), and the PCIe reduction remains substantial ($23\times$–$176\times$) at all budgets. This is not a replacement for JPEG in general-purpose imaging; it targets pipelines where images are consumed exclusively on GPU, PCIe is a shared resource across concurrent tasks, and the 0.1 bpp regime is appropriate, such



(a) I-NGP (b) GI (c) SAD (d) IMG-GS (e) Ours (f) Ref. Crop (g) Reference

Figure 4. Qualitative comparison with other recent image representation baselines. Examples are from the DIV2K dataset. Columns (a)-(e) show model reconstructions with error map insets, (f) and (g) show reference crops and the full reference images respectively. The budgets of I-NGP, GI, SAD, IMG-GS, and Gaussian-Grid are 0.1 bits per pixel respectively.

Table 1. Comparison of Gaussian-Grid with recent primitive-based and neural baselines on DIV2K (100 images) and CLIC2020 validation (41 images) natural image datasets across four budgets (0.1 bits per pixel to 1.0 bits per pixel). The best result is in dark blue bold and the second best is in light blue.

Method	Metric	DIV2K				CLIC2020			
		0.1 bpp	0.4 bpp	0.7 bpp	1.0 bpp	0.1 bpp	0.4 bpp	0.7 bpp	1.0 bpp
SAD [13]	PSNR ↑	24.72	**29.36**	**31.24**	32.43	26.95	31.29	33.04	34.10
	SSIM ↑	0.6573	0.7816	0.8309	0.8597	0.7210	0.8096	0.8465	0.8698
	MS-SSIM ↑	0.7954	0.9196	0.9495	0.9632	0.8353	0.9271	0.9518	0.9643
	LPIPS ↓	0.4512	0.2980	0.2399	0.2040	0.4238	0.2996	0.2524	0.2197
GI [38]	PSNR ↑	24.21	27.27	28.71	29.69	26.36	29.15	30.40	31.33
	SSIM ↑	0.6512	0.7640	0.8121	0.8402	0.7147	0.7980	0.8308	0.8528
	MS-SSIM ↑	0.8398	0.9271	0.9488	0.9588	0.8707	0.9364	0.9527	0.9613
	LPIPS ↓	0.4408	0.2812	0.2247	0.1856	0.3718	0.2546	0.2103	0.1778
I-NGP [16]	PSNR ↑	22.16	25.30	26.27	26.98	24.85	27.55	28.38	28.96
	SSIM ↑	0.5341	0.6656	0.7152	0.7501	0.6396	0.7281	0.7637	0.7894
	MS-SSIM ↑	0.7163	0.8602	0.8963	0.9184	0.7951	0.8888	0.9154	0.9308
	LPIPS ↓	0.5435	0.3749	0.3109	0.2599	0.4755	0.3344	0.2807	0.2447
IMG-GS [40]	PSNR ↑	24.46	27.97	29.49	30.51	26.66	30.15	31.66	32.62
	SSIM ↑	0.6711	0.7754	0.8180	0.8438	0.7329	0.8112	0.8423	0.8607
	MS-SSIM ↑	0.8243	0.9218	0.9467	0.9588	0.8586	0.9346	0.9536	0.9628
	LPIPS ↓	**0.3891**	0.2682	0.2217	0.1927	**0.3370**	0.2395	0.2048	0.1834
Ours	PSNR ↑	**25.58**	**29.36**	31.22	**32.55**	**27.70**	**31.32**	**33.06**	**34.26**
	SSIM ↑	**0.6997**	**0.8253**	**0.8730**	**0.8999**	**0.7523**	**0.8448**	**0.8821**	**0.9039**
	MS-SSIM ↑	**0.8869**	**0.9554**	**0.9713**	**0.9787**	**0.9074**	**0.9604**	**0.9735**	**0.9798**
	LPIPS ↓	0.3902	**0.2178**	**0.1545**	**0.1167**	0.3601	**0.2210**	**0.1665**	**0.1324**

as large-scale retrieval, streaming previews, and thumbnail caching.

Table 2. Step 1 loss and Step 2 architecture ablations at 1.0 bpp on 100 DIV2K validation images. For Step 1 loss ablation, the architecture is fixed to the full method. Each row adds one component to the previous. The best result is in dark blue bold and the second best is in light blue. We further clarify the perceptual metric result in Section 4.3 and in Figure 5.

Step 1 Conditions	PSNR ↑	SSIM ↑	MS-SSIM ↑	LPIPS ↓
$\mathcal{L}_{\mathrm{L1}}$	32.31	0.8852	0.9752	0.1434
$\mathcal{L}_{\mathrm{L1}} + \mathcal{L}_{\mathrm{ov}}$ (full)	**32.55**	**0.8999**	**0.9787**	**0.1167**
Step 2 Conditions				
Uniform placement	32.08	**0.9069**	**0.9809**	**0.0993**
+ Freq.-guided placement	30.86	0.8775	0.9738	0.1399
+ Budget weighting	32.25	0.8844	0.9750	0.1437
+ Directional grids	32.29	0.8866	0.9756	0.1413
Ours (full)	**32.55**	0.8999	0.9787	0.1167

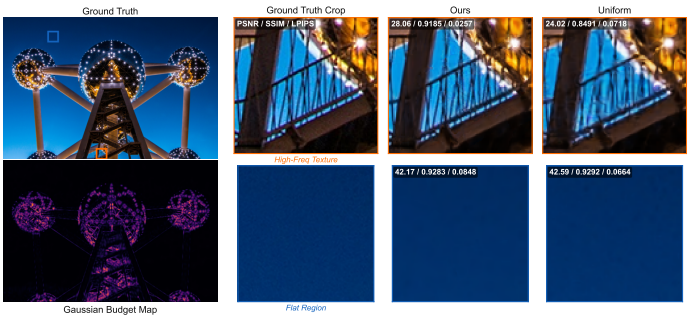


Figure 5. Qualitative comparison between our full method and the uniform placement baseline on DIV2K image 0851 at 1.0 bpp. **Top row (textured region):** Frequency-guided Gaussian grids preserve complex details, yielding improvements across all three metrics (e.g., sharper details along the fence). **Bottom row (flat region):** Capacity reallocation leaves smooth areas with low-frequency, canvas-only coverage, resulting in marginally lower metrics yet visually negligible to observers. This demonstrates how our method prioritizes critical details at a minor cost to flat regions that is imperceptible.

Table 3. End-to-end decode comparison against JPEG + ZIP on 100 DIV2K images at four budgets. At 0.1 bpp our representation yields +2.33 dB over a size-matched JPEG while decoding 2.14× faster; The decode speedup (1.24–2.14×, Speedup column) persists across all budgets. The rightmost column (PCIe×) shows the most distinctive property: transferring quantized model weights (47–366 KB) instead of decoded pixel maps (8,306 KB per image) reduces CPU-to-GPU traffic by 23–176×. **Pipeline columns.** Ours: weights load (*Load*), GPU transfer (*PCIe*), GPU dequantize (*Dq*), GPU kernel forward pass (*Krnl*). JPEG: one-time ZIP extraction on CPU amortized over 100 images (*Unzip*), JPEG CPU decode (*Dec*), and pixel transfer to GPU (*PCIe*).

	Dataset size (MB)		PSNR (dB)			JPEG decode (ms / image)				Our decode (ms / image)						
bpp	JPEG	Ours	JPEG	Ours	Δ	Unzip	Dec	PCIe	Total	Load	PCIe	Dq	Krnl	Total	Speedup	PCIe×
0.1	4.85	4.62	23.25	**25.58**	**+2.33**	0.87	10.05	0.73	11.65	0.17	0.14	0.64	4.48	**5.44**	**2.14×**	176×
0.4	14.71	14.85	29.05	**29.36**	**+0.31**	1.20	9.46	0.67	11.33	0.16	0.15	0.63	6.78	**7.73**	**1.47×**	55×
0.7	24.97	25.28	**31.75**	31.22	−0.53	1.26	9.88	0.64	11.78	0.16	0.15	0.57	8.25	**9.13**	**1.29×**	32×
1.0	35.48	35.71	**33.55**	32.55	−1.00	1.30	10.61	0.64	12.55	0.16	0.16	0.59	9.19	**10.11**	**1.24×**	23×
All	80.0	80.5	—		**+0.27**	1.16	10.00	0.67	11.83	0.16	0.15	0.61	7.18	**8.10**	**1.46×**	40×

6. Limitations and Future Work

The gains of Gaussian-Grid depend on how spatial heterogeneity of an image frequency content. When detail is concentrated, frequency-guided allocation directs capacity to complex regions and yields clear gains; as frequency energy becomes uniform, the allocation approaches the uniform baseline (see supplement). Perceptual metrics do not always follow the PSNR gains, since capacity is drawn away from smooth regions. In the future, we plan to investigate human visual contrast sensitivity function based on eye-display distances [22] for our resource allocation weighing. Integrating the visual sensitivity to extend the currently high-frequency preservation would yield increased perceptual quality for human observers. Additionally, Gaussian-Grid encodes each image by per-image optimization, which takes on average 24.3 min per DIV2K image on a single NVIDIA A100, almost all of it in Step 2. This is far slower than traditional codecs such as JPEG, and we do not claim competitive encoding times against the primitive-based baselines. Our target setting is write-once, read-many storage (Section 5). Shorter Step 2 schedules with early stopping, or learned initialization of the grid features, could reduce this cost and are left for future work.

7. Conclusion

In this work, we proposed Gaussian-Grid, a hybrid content-adaptive image representation utilizing frequency-guided feature grids via 2D Gaussian primitives. Gaussian-Grid achieves a favorable quality at low budgets, while preserving a significant perceptual win and fast rendering speeds, making it suitable for GPU-native data storage. We hope this work inspires future advances in content-adaptive image representations.

References

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017. 1, 5

[2] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-time rendering*. AK Peters/crc Press, 2019. 2

[3] Yash Belhe, Michael Gharbi, Matthew Fisher, Iliyan Georgiev, Ravi Ramamoorthi, and Tzu-Mao Li. Discontinuity-aware 2d neural fields. *ACM Transactions on Graphics (TOG)*, 42(6):1–11, 2023. 2

[4] Graham Campbell, Thomas A DeFanti, Jeff Frederiksen, Stephen A Joyce, Lawrence A Leske, John A Lindberg, and Daniel J Sandin. Two bit/pixel full color encoding. *ACM SIGGRAPH Computer Graphics*, 20(4):215–223, 1986. 2

[5] Brian Chao, Hung-Yu Tseng, Lorenzo Porzi, Chen Gao, Tuotuo Li, Qinbo Li, Ayush Saraf, Jia-Bin Huang, Johannes Kopf, Gordon Wetzstein, et al. Textured gaussians for enhanced 3d scene appearance modeling. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 8964–8974, 2025. 3

[6] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8628–8638, 2021. 2

[7] Sayantan Datta, Carl Marshall, Zhao Dong, Zhengqin Li, and Derek Nowrouzezahrai. Efficient graphics representation with differentiable indirection. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–10, 2023. 2

[8] E. Delp and O. Mitchell. Image compression using block truncation coding. *IEEE Transactions on Communications*, 27(9):1335–1342, 1979. 2

[9] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations, 2021. 3

[10] Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Goliński, Yee Whye Teh, and Arnaud Doucet. Coin++: Neural compression across modalities, 2022. 3

[11] Simon Fenney. Texture compression using low-frequency signal modulation. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 84–91, 2003. 2

[12] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Trans. Graph.*, 38(6), 2019. 5

[13] Laki Iinbor, Zhiyang Dou, and Wojciech Matusik. Soft anisotropic diagrams for differentiable image representation. *ACM Transactions on Graphics*, 2026. SIGGRAPH 2026. 3, 5, 7

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 5

[15] Julien NP Martel, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein. Acorn: adaptive coordinate networks for neural scene representation. *ACM Transactions on Graphics (TOG)*, 40(4):1–13, 2021. 1, 2, 3

[16] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022. 1, 2, 5, 7

[17] Jacob Munkberg, Petrik Clarberg, Jon Hasselgren, and Tomas Akenine-Möller. High dynamic range texture compression for graphics hardware. *ACM Transactions on Graphics (TOG)*, 25(3):698–706, 2006. 2

[18] Jörn Nystad, Anders Lassen, Andy Pomianowski, Sean Ellis, and Tom Olson. Adaptive scalable texture compression. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics Conference on High-Performance Graphics*, pages 105–114, 2012. 2

[19] Panagiotis Papantonakis, Georgios Kopanas, Fredo Durand, and George Drettakis. Content-aware texturing for gaussian splatting. *arXiv preprint arXiv:2512.02621*, 2025. 3

[20] Kimmo Roimela, Tomi Aarnio, and Joonas Itäranta. High dynamic range texture compression. *ACM Transactions on Graphics (TOG)*, 25(3):707–712, 2006. 2

[21] Victor Rong, Jan Held, Victor Chu, Daniel Rebain, Marc Van Droogenbroeck, Kiriakos N Kutulakos, Andrea Tagliasacchi, and David B Lindell. Nexels: Neurally-textured surfels for real-time novel view synthesis with sparse geometries. *arXiv preprint arXiv:2512.13796*, 2025. 3

[22] Jyrki Rovamo, Juvi Mustonen, and Risto Näsänen. Modelling contrast sensitivity as a function of retinal illuminance and grating area. *Vision Research*, 34(10):1301–1314, 1994. 8

[23] Joseph Shenouda, Yamin Zhou, and Robert D Nowak. Relus are sufficient for learning implicit neural representations. *arXiv preprint arXiv:2406.02529*, 2024. 3

[24] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020. 3

[25] Jacob Ström and Tomas Akenine-Möller. i packman: High-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 63–70, 2005. 2

[26] Jacob Ström and Martin Pettersson. Etc 2: texture compression using invalid combinations. In *Graphics Hardware*, pages 49–54, 2007. 2

[27] Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022. 2

[28] Towaki Takikawa, Thomas Müller, Merlin Nimier-David, Alex Evans, Sanja Fidler, Alec Jacobson, and Alexander Keller. Compact neural graphics primitives with learned hash probing. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–10, 2023. 2

[29] George Toderici, Wenzhe Shi, Radu Timofte, Johannes Balle Lucas Theis, Eirikur Agustsson, Nick Johnston, and Fabian

Mentzer. Workshop and challenge on learned image compression (clic2020), 2020. 1, 5

[30] Karthik Vaidyanathan, Marco Salvi, Bartlomiej Wronski, Tomas Akenine-Moller, Pontus Ebelin, and Aaron Lefohn. Random-access neural compression of material textures. *ACM Transactions on Graphics (TOG)*, 42(4):1–25, 2023. 1, 2, 5

[31] Gregory K. Wallace. The jpeg still picture compression standard. *Commun. ACM*, 34(4):30–44, 1991. 6

[32] Chao Wang, Ana Serrano, Xingang Pan, Krzysztof Wolski, Bin Chen, Karol Myszkowski, Hans-Peter Seidel, Christian Theobalt, and Thomas Leimkühler. An implicit neural representation for the image stack: Depth, all in focus, and high dynamic range. *ACM Transactions on Graphics (TOG)*, 42 (6):1–11, 2023. 3

[33] Yiming Wang, Shaofei Wang, Marko Mihajlovic, and Siyu Tang. Neural texture splatting: Expressive 3d gaussian splatting for view synthesis, geometry, and dynamic reconstruction. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, pages 1–12, 2025. 3

[34] Zhou 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*, 13(4): 600–612, 2004. 5

[35] Clément Weinreich, Louis De Oliveira, Antoine Houdard, and Georges Nader. Real-time neural materials using block-compressed features. In *Computer Graphics Forum*, page e15013. Wiley Online Library, 2024. 2

[36] Zhaojie Zeng, Yuesong Wang, Tao Guan, Chao Yang, and Lili Ju. Instant gaussianimage: A generalizable and self-adaptive image representation via 2d gaussian splatting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 27896–27905, 2025. 3

[37] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 5

[38] Xinjie Zhang, Xingtong Ge, Tongda Xu, Dailan He, Yan Wang, Hongwei Qin, Guo Lu, Jing Geng, and Jun Zhang. Gaussianimage: 1000 fps image representation and compression by 2d gaussian splatting. *arXiv preprint arXiv:2403.08551*, 2024. 1, 3, 5, 7

[39] Xiaoyu Zhang, Weihong Pan, Chong Bao, Xiyu Zhang, Xiaojun Xiang, Hanqing Jiang, and Hujun Bao. Lookcloser: Frequency-aware radiance field for tiny-detail scene. In *Proceedings of the computer vision and pattern recognition conference*, pages 16122–16132, 2025. 3

[40] Yunxiang Zhang, Bingxuan Li, Alexandr Kuznetsov, Akshay Jindal, Stavros Diolatzis, Kenneth Chen, Anton Sochenov, Anton Kaplanyan, and Qi Sun. Image-gs: Content-adaptive image representation via 2d gaussians. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pages 1–11, 2025. 1, 3, 5, 7