# SPLATFOUNDRY: VOXEL-ADAPTIVE GAUSSIAN DECODING WITH LEARNED ADAM REFINEMENT

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Feed-forward 3D Gaussian reconstruction replaces costly per-scene fitting with a single network evaluation. However, existing primitive allocations impose an unfavorable choice: pixel-aligned predictors replicate Gaussians across views, while a fixed latent codebook cannot grow with scene extent and complexity. We introduce SPLATFOUNDRY, a two-stage framework that combines adaptive 3D allocation with learned Adam refinement. First, we aggregate points and features from a frozen geometric foundation model into depth-adaptive voxel tokens. A Transformer decoder applies self-attention to the concatenation of image-patch and voxel tokens, then decodes a constant number of Gaussians from each occupied voxel token. The resulting primitive count adapts to the occupied 3D space rather than scaling with the number of patch tokens or being capped by a global budget. Second, a state estimator predicts per-Gaussian, per-attribute Adam moments and learning rates from voxel tokens. The differentiable post-optimization fits the Gaussians to the context views, while the refined Gaussians are supervised end-to-end by the standard novel-view objective. Unlike cold-start Adam, this learned initialization produces substantial improvement after only a few refinement epochs. To reduce training memory, we introduce the Accumulated Adam Jacobian (AAJ), whose backward-pass memory is independent of the number of Adam refinement iterations. Across DL3DV and three out-of-distribution datasets, our method improves PSNR by $2.5$ dB and uses $6.2\times$ fewer Gaussians on average than the strongest prior method in each setting.

## 1 INTRODUCTION

3D Gaussian Splatting (3DGS) combines an explicit scene representation with fast differentiable rendering (Kerbl et al., 2023). Its quality, however, traditionally comes from fitting hundreds of thousands of primitive parameters independently for every captured scene. Generalizable predictors instead reconstruct a scene in a single feed-forward pass, substantially reducing per-scene processing time. Their learned reconstruction priors also make them particularly robust when only a few input views are available. Extending these benefits to larger inputs depends on the Gaussian decoding strategy, which controls how reconstruction cost and capacity scale as additional views reveal more of the scene.

Most current Gaussian predictors allocate primitives in one of two ways. Patch-aligned approaches (Charatan et al., 2024; Chen et al., 2024; Zhang et al., 2024b) decode one or more Gaussians from each image patch. They inherit useful 2D detail, but their Gaussian count grows with both the number and resolution of input images, and multi-view observations of the same surface generate repeated primitives. Increasing the number of views under fixed scene coverage therefore increases redundancy. At the other extreme, latent-query or codebook approaches compress the observations and decode the scene through a fixed number of learnable tokens (Ren et al., 2026; Itkin et al., 2026). They produce compact reconstructions, but their maximum Gaussian budget is fixed before the scene is observed. Sliding-window decoding can extend coverage but reintroduces duplicate predictions. This creates the first scaling tension: representation capacity should grow with occupied 3D structure—remaining compact when views overlap, yet expanding as new scene regions are observed.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

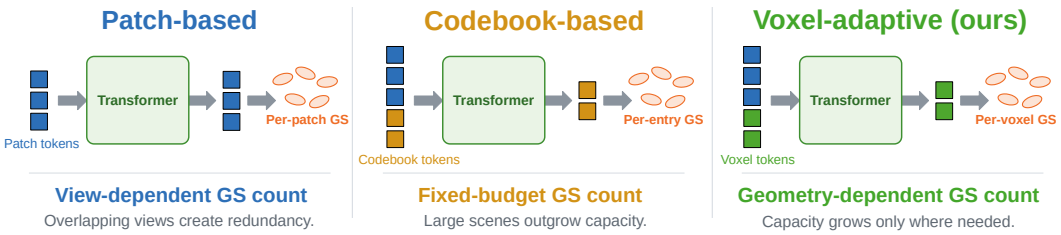


Figure 1: Comparison of Gaussian-allocation strategies. Patch-based decoding grows with the number of context views and duplicates primitives across overlapping views; codebook-based decoding uses a fixed budget; SPLATFOUNDRY allocates according to occupied 3D structure.

We address this problem with *adaptive voxel tokens*, using proxy geometry from a frozen multi-view geometry backbone to guide efficient voxel allocation. We lift all views into a shared 3D space and assign each point to the best-fit level in a discrete voxel-grid hierarchy based on its projected pixel footprint. Near surfaces use finer levels, while distant surfaces use coarser ones. The resulting voxel tokens exchange information with patch tokens through joint Transformer self-attention, after which each output voxel token decodes a fixed number of Gaussians. The output count therefore scales with the number of occupied voxels rather than the number of input views and is not capped by a fixed budget. Fig. 1 contrasts different allocation strategies.

Adaptive allocation controls the size and spatial distribution of the representation, but does not by itself close the quality gap between feed-forward prediction and scene-specific fitting. A single forward pass predicts all Gaussian attributes without leveraging the rendering residuals of the scene, which motivates subsequent scene-specific refinement (Li et al., 2026; Liu et al., 2026). However, standard Adam (Kingma & Ba, 2015) initializes its first- and second-moment estimates to zero, leaving the optimizer to accumulate scene-specific gradient statistics during refinement. We instead train each voxel token to predict, alongside its Gaussian attributes, initial Adam moments and per-attribute learning rates. A few differentiable Adam steps then minimize context-view rendering losses, while a novel-view rendering objective supervises the predicted optimizer initialization through the refined Gaussians. Naively differentiating through these steps requires retaining an unrolled computation graph whose memory cost grows with the refinement horizon. To address this, we introduce the Accumulated Adam Jacobian (AAJ), which makes backward-pass memory complexity independent of the number of refinement iterations. Our learned Adam converges substantially faster than cold-start Adam.

A second scaling tension concerns image resolution. Existing feed-forward models commonly pursue finer detail by directly processing high-resolution images, but doing so creates long token sequences throughout the reconstruction network. We use a different approach. SPLATFOUNDRY runs its feed-forward stage at a capped resolution to predict a compact set of Gaussians that preserves the essential scene geometry. Learned Adam then refines these Gaussians against the high-resolution context images. High-resolution evidence is therefore incorporated through differentiable rendering rather than additional tokens, keeping the feed-forward sequence short while the compact representation and predicted Adam state enable rapid refinement.

The resulting method has four principal contributions:

- We introduce an adaptive voxel-based Gaussian Transformer whose token count follows observed 3D structure, reducing cross-view redundancy without a fixed codebook capacity. It uses $6.2\times$ fewer Gaussians on average than the highest-quality baseline in each setting.
- We introduce learned Adam refinement with predicted per-Gaussian, per-attribute moments and learning rates. In our controlled experiments, it surpasses cold-start Adam by $0.64$ dB after one refinement epoch and $0.96$ dB after eight.
- We introduce the Accumulated Adam Jacobian algorithm, which reduces the outer-backward space complexity from $O(TG)$ to $O(G)$ when differentiating through $T$ Adam iterations over $G$ Gaussian parameters and saves over $20$ GiB of GPU memory during training.
- We propose a resolution-decoupled reconstruction strategy: the Transformer predicts a compact Gaussian representation at a capped resolution, while learned Adam recovers detail from high-resolution context images without lengthening the token sequence.

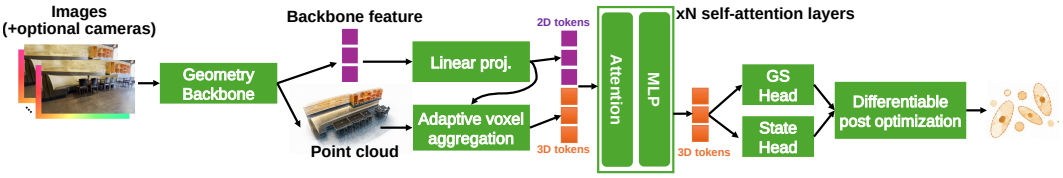


Figure 2: Overview of SPLATFOUNDRY: adaptive voxel allocation (Sec. 3.2), joint patch–voxel decoding (Sec. 3.3), and learned Adam refinement (Sec. 3.4).

## 2 RELATED WORK

**Per-scene Gaussian optimization.** 3D Gaussian Splatting (3DGS) fits explicit primitives to each scene through differentiable rasterization and adaptive density control (Kerbl et al., 2023). Related variants structure the Gaussians with voxel- or octree-based anchors for adaptive allocation and rendering (Lu et al., 2024; Ren et al., 2025; Lin et al., 2025). They demonstrate the value of voxel structure for per-scene optimization, whereas we learn a feed-forward voxel representation.

**Pixel-aligned Gaussian prediction.** Most feed-forward methods attach Gaussian decoding to a 2D image grid. PixelSplat, MVSplat, and DepthSplat predict view-aligned Gaussians from per-pixel or ray features, using depth distributions, plane-sweep volumes, or pretrained depth features, respectively (Charatan et al., 2024; Chen et al., 2024; Xu et al., 2025). GS-LRM directly decodes Gaussians from patch tokens (Zhang et al., 2024b). Long-LRM and tttLRM scale this paradigm to longer contexts through linear-complexity sequence modeling (Chen et al., 2025; Wang et al., 2026a). AnySplat and Splatt3RFusion consolidate dense per-view predictions through post-prediction voxelization or fusion (Jiang et al., 2025; Smart et al., 2026). In contrast, our voxel tokens are allocated in 3D before Transformer processing, and we avoid dense per-view Gaussian prediction.

**Latent-token Gaussian prediction.** TokenGS, GlobalSplat, and C3G aggregate image evidence into fixed sets of learned scene tokens or queries that directly regress Gaussians (Ren et al., 2026; Itkin et al., 2026; An et al., 2026). These designs remove view-driven output growth, but choose their scene capacity before observing the scene. Our representation has no fixed global budget: tokens are instantiated only for occupied voxels and therefore grow with observed 3D structure.

**Voxel-aligned Gaussian prediction.** Voxel-aligned Gaussian decoding has also been explored for object generation: GVGEN and GaussianCube represent Gaussian attributes on dense voxel grids, while TRELLIS decodes sparse voxel latents into Gaussians (He et al., 2024; Zhang et al., 2024a; Xiang et al., 2025). In reconstruction, concurrent VolSplat unprojects image features into sparse voxels and predicts one Gaussian per occupied cell (Wang et al., 2026b). These methods use a uniform voxel size for Gaussian decoding, whereas our footprint-aware hierarchy adapts the voxel size to each observation's projected image footprint.

**Optimization-aware feed-forward reconstruction.** QuickSplat learns a 2DGS initializer together with neural update and densification modules to accelerate per-scene surface reconstruction (Liu et al., 2025). ForeSplat trains feed-forward Gaussian initializers for rapid downstream optimization using a first-order meta-gradient objective (Li et al., 2026). Diff3R employs a preconditioned conjugate-gradient solver with uncertainty-aware constraints (Liu et al., 2026), whereas concurrent ReSplat applies attention conditioned on rendering residuals to recurrently update the Gaussians (Xu et al., 2026). Our learned Adam instead initializes Adam with predicted moments and learning rates—a simple yet effective refinement strategy.

## 3 METHOD

### 3.1 OVERVIEW AND PROBLEM FORMULATION

Given $V$ context images $\{I_v\}_{v=1}^{V}$, we reconstruct the captured scene as a 3D Gaussian Splat representation $\Theta = \{(\boldsymbol{\mu}_i, \mathbf{s}_i, \mathbf{q}_i, o_i, \mathbf{c}_i)\}_{i=1}^{N_G}$. Here $\boldsymbol{\mu}_i \in \mathbb{R}^3$ is the mean, $\mathbf{s}_i \in \mathbb{R}^3_+$ is the scale, $\mathbf{q}_i$ is a unit quaternion encoding orientation, $o_i \in (0,1)$ is opacity, and $\mathbf{c}_i$ contains the spherical-harmonic color coefficients. A differentiable Gaussian rasterizer $\mathcal{R}$ produces $\hat{I}_C = \mathcal{R}(\Theta, C)$ from a camera $C$.

Our voxel-based decoder assumes proxy geometry from a frozen multi-view geometry backbone $\mathcal{B}$. From the context images, $\mathcal{B}$ predicts a dense depth map $D_v$, decoder features $F_v$, and camera

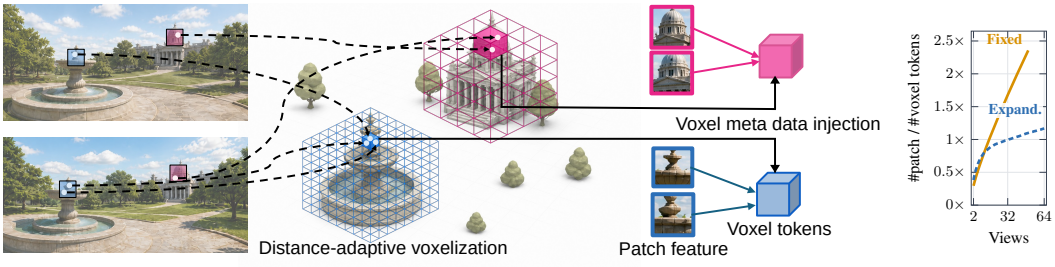


Figure 3: Distance-adaptive voxelization, voxel-token initialization, and allocation scaling. Left: lifted observations select a grid level from their distance to the source camera; nearby geometry uses fine cells (blue), while distant geometry uses coarse cells (pink). Observations assigned to the same occupied cell are reduced into one voxel token, whose 3D position and level-dependent scale are injected as metadata. Right: the patch-to-voxel token ratio as context views are added under fixed or expanding coverage on DL3DV.

parameters for every view. If externally calibrated cameras are provided, we transform their poses into the predicted coordinate system before use, as detailed in App. B.1. Each depth map and its camera define a dense point map $\mathbf{p}_v$. Our contribution focuses on allocating voxel tokens from this proxy and decoding Gaussians from them, rather than on estimating the proxy geometry itself.

As illustrated in Fig. 2, SPLATFOUNDRY has a feed-forward reconstruction stage followed by learned Adam refinement. The first stage uses the proxy geometry to aggregate image features into adaptive sparse voxels (Sec. 3.2), then concatenates patch tokens $P$ and voxel tokens $Z$ into a single Transformer sequence and decodes the initial Gaussian parameters $\Theta_0$ exclusively from the output voxel tokens (Sec. 3.3). The same voxel tokens also predict the initial Adam moments and learning rates. Starting from this predicted representation and state, learned Adam performs $T$ scene-specific refinement iterations to produce the final parameters $\Theta_T$ (Sec. 3.4).

### 3.2 ADAPTIVE SPARSE VOXEL ALLOCATION

We organize the shared 3D space as a sparse octree grid with power-of-two cell widths. Instead of using one resolution throughout the scene, each lifted point selects the level whose cell has a target projected footprint of $p_v \times p_v$ pixels in its source view. Points closer to their source cameras therefore select finer levels, whereas more distant points select coarser levels. This avoids under-resolving nearby surfaces with overly large voxels while preventing distant regions from consuming an excessive number of voxels. Each point is then identified by its integer 3D cell coordinate together with its selected grid level.

We select the level for each point as follows. Let $f_x$ and $f_y$ be the focal lengths in pixels. The angular width of a pixel is approximated by $\delta = 1/\sqrt{f_x f_y}$. At source-view depth $D$, one pixel therefore spans a world-space width of $D\delta$. We select the discrete octree level as

$$\widetilde{s} = D\delta p_v \ln 2, \qquad \ell = \lceil \log_2 \widetilde{s} \rceil . \tag{1}$$

Here $\widetilde{s}$ is the calibrated continuous cell width, and $\ell$ may be negative. The selected octree cell has world-space width $2^\ell$. Assuming a uniform logarithmic phase, Eq. 1 gives an expected projected voxel width of $p_v$ pixels. We provide the derivation in App. B.4.

A point at shared-frame position $\mathbf{p}$ is assigned the sparse key $\kappa = (\lfloor \mathbf{p}/2^\ell \rfloor, \ell)$. The unique keys across all lifted points define the candidate occupied voxel set. We retain only voxels containing more than $\tau$ lifted points and keep the remaining point-to-voxel assignments.

**Analysis.** The right panel of Fig. 3 compares patch- and voxel-token growth on DL3DV (Ling et al., 2024) under fixed and expanding scene coverage, using matched patch and voxel footprints. Patch tokens grow linearly with the number of views. Under fixed coverage, the occupied-voxel count quickly saturates as overlapping observations merge into existing cells. Under expanding coverage, both counts grow as new views reveal new geometry, but voxel allocation grows more slowly because overlap is still consolidated. The rising patch-to-voxel ratio in both regimes demonstrates that voxel allocation follows observed scene content rather than raw view count. App. B.3 provides the sampling protocol.

### 3.3 TRANSFORMER FOR VOXEL-BASED GAUSSIAN DECODING

Given the proxy geometry and features produced by the frozen geometry backbone, our Transformer aggregates image evidence into a set of voxel tokens and decodes each token into Gaussian primitives.

**Patch and voxel tokens.** For image patch $i$, let $I_i$ and $R_i$ denote its RGB and Plücker ray-map values, and let $F_i$ be its frozen-backbone feature. We construct the patch token

$$\widetilde{P}_i = W_I \,\mathrm{flatten}(I_i) + W_R \,\mathrm{flatten}(R_i) + W_F F_i, \qquad P_i^0 = \mathrm{LN}(\widetilde{P}_i), \tag{2}$$

where $\mathrm{flatten}$ vectorizes the values within a patch and LN denotes layer normalization. The point-to-voxel assignments from Sec. 3.2 then lift these patch features into voxel tokens. Let $\Omega_j$ be the multiset of patch indices induced by the lifted pixels assigned to voxel $j$; repeated indices count multiple assigned pixels. The initial voxel token is

$$\widetilde{Z}_j = W_Z \,\mathrm{VoxelPE}(\kappa_j) + |\Omega_j|^{-1} \sum\nolimits_{i \in \Omega_j} \widetilde{P}_i, \qquad Z_j^0 = \mathrm{LN}(\widetilde{Z}_j), \tag{3}$$

where $\kappa_j$ is the sparse voxel key from Sec. 3.2, and $\mathrm{VoxelPE}$ is its positional encoding detailed in App. B.2. Thus each patch token is weighted by the number of its lifted pixels assigned to the voxel.

**Joint self-attention.** We concatenate the patch and voxel tokens along the sequence dimension as $S^0 = [P^0; Z^0] \in \mathbb{R}^{(N_P+N_Z)\times d}$. A stack of $L$ standard Transformer blocks (Vaswani et al., 2017) applies joint self-attention and a SwiGLU MLP (Shazeer, 2020) to the full sequence. With pre-normalization (Xiong et al., 2020), block $t$ computes

$$\widetilde{S}^t = S^{t-1} + \mathrm{SelfAttn}\big(\mathrm{LN}(S^{t-1})\big), \qquad S^t = \widetilde{S}^t + \mathrm{SwiGLU\text{-}MLP}(\mathrm{LN}(\widetilde{S}^t)), \tag{4}$$

allowing patch–patch, patch–voxel, and voxel–voxel interactions within every self-attention layer.

**Gaussian prediction.** After the last block, we discard the output patch tokens and decode Gaussian parameters only from the voxel tokens:

$$[P^L; Z^L] = S^L, \qquad \Theta_0 = \mathrm{SwiGLU\text{-}MLP}(\mathrm{LN}(Z^L)). \tag{5}$$

We predict $K$ Gaussians per voxel token, resulting in $N_Z K$ Gaussians in total. Their attributes are parameterized relative to the corresponding voxel. Full output parameterization and initialization details are provided in App. B.

### 3.4 LEARNED ADAM REFINEMENT

Feed-forward features describe which Gaussian attributes are geometrically certain, weakly observed, or ambiguous. Cold-start Adam (Kingma & Ba, 2015) discards this information and begins every scene from zero moments and hand-designed learning rates. We instead predict a scene-conditioned initial Adam state and learning rates for every Gaussian attribute. Fig. 4 illustrates our pipeline.

**State estimator.** A separate head with the same architecture as Eq. 5 predicts a first-moment logit $\widehat{m}_0$, log second moment $\widehat{v}_0$, and log learning rate $\widehat{\eta}$ for each Gaussian attribute, which we map to

$$v_0 = \exp(\widehat{v}_0), \qquad m_0 = \tanh(\widehat{m}_0)\sqrt{v_0}, \qquad \eta = \exp(\widehat{\eta}). \tag{6}$$

This parameterization enforces $m_0^2 \le v_0$, consistent with the Cauchy–Schwarz relation between first and second moments. Clipping and initialization are detailed in App. B.5.

**Differentiable context refinement.** Starting from the learned Adam state, each post-optimization iteration samples one context view $\nu_t$ and computes a detached gradient of its rendering MSE:

$$g_t = \mathrm{StopGradient}(\nabla_{\Theta_t} \,\mathrm{MSE}(\mathcal{R}(\Theta_t, C_{\nu_t}), I_{\nu_t})\,). \tag{7}$$

The standard Adam rule uses $g_t$ to advance the moments and Gaussian attributes from $(m_t, v_t, \Theta_t)$ to $(m_{t+1}, v_{t+1}, \Theta_{t+1})$. Detaching the context gradient avoids expensive second derivatives through the Gaussian rasterizer, while the Adam recurrence remains differentiable with respect to $\Theta_0$, $m_0$, $v_0$, and $\eta$. A held-out novel-view objective can therefore train the entire predicted initialization through $\Theta_T$. Rather than retaining and backpropagating through the unrolled trajectory, our *Accumulated Adam Jacobian* (AAJ) accumulates the exact endpoint-derivative factors during forward refinement and maps the novel-view gradient directly back to $(\Theta_0, m_0, v_0, \eta)$. AAJ reduces the outer backward's space and time complexity from $O(TG)$ to $O(G)$, where $G$ is the number of Gaussian parameters. The Adam updates and AAJ derivation are given in Apps. B.5 and B.6, respectively.




Figure 4: Differentiable learned Adam refinement. The Transformer predicts initial Gaussian parameters $\Theta_0$, Adam moments $(m_0, v_0)$, and learning rates $\eta$. Each of $T$ refinement iterations renders one context view, computes and detaches its rendering-loss gradient, and advances the Gaussian attributes and Adam state. During training, the Accumulated Adam Jacobian (AAJ) directly propagates the held-out novel-view gradient from $\Theta_T$ to the predicted initialization $(\Theta_0, m_0, v_0, \eta)$ without retaining the unrolled trajectory (red arrow). Inference uses only context views.

**Long-horizon state handoff.** We define one refinement epoch as a complete pass over all context views; with $V$ views, an epoch therefore contains $V$ Adam iterations. Although trained for one epoch, learned Adam generalizes well to a few epochs at test time. With dense context sets, however, we find that learned-state refinement saturates after a few epochs. We therefore introduce a configurable handoff epoch $E_{\text{swap}}$. Alongside the learned moments, we accumulate a zero-initialized empirical moment stream from the same detached gradients. At the end of epoch $E_{\text{swap}}$, we replace the learned state and learning rates with the empirical state and standard attribute-wise 3DGS learning rates. Subsequent iterations use conventional Adam, retaining the rapid early refinement of the learned state while enabling continued long-horizon improvement.

### 3.5 High-resolution feedback

Most feed-forward reconstruction methods use higher-resolution images to recover finer details. However, more input pixels produce more tokens, making self-attention increasingly expensive. By contrast, rendering-based post-optimization scales more gently with image resolution. We also observe diminishing returns from feed-forward prediction as the decoder allocates more tokens and Gaussians. These observations suggest that post-optimization is a more efficient way to extract high-resolution evidence than processing it with additional tokens. We therefore cap the input to the feed-forward pipeline at a pixel budget $N_{\mathcal{B}}$. Learned Adam then rapidly refines the predicted Gaussians using high-resolution source-view feedback, as in Eq. 7.

### 3.6 Training recipe

Given a held-out novel view with image $I^*$ and camera $C^*$, let $\ell(\hat{I}, I) = \text{MSE}(\hat{I}, I) + \lambda_{\text{perc}} \mathcal{L}_{\text{perc}}(\hat{I}, I)$ denote the rendering loss, where $\mathcal{L}_{\text{perc}}$ is a VGG perceptual loss (Johnson et al., 2016). We jointly supervise the feed-forward Gaussians $\Theta_0$ and their post-optimized counterpart $\Theta_T$ with

$$\mathcal{L} = \lambda_{\text{pre}} \ell(\mathcal{R}(\Theta_0, C^*), I^*) + \lambda_{\text{post}} \ell(\mathcal{R}(\Theta_T, C^*), I^*) \,. \tag{8}$$

Our curriculum consists of a sequence of checkpoints trained at progressively higher feed-forward resolutions. After reaching the pixel budget $N_{\mathcal{B}}$, we introduce learned Adam and jointly train its state estimator with our Transformer. More details are provided in App. B.7.

## 4 Experiments

### 4.1 Implementation details

**Evaluation protocol.** Our main evaluation follows the known-camera setting and uses camera parameters obtained through offline calibration. We view joint camera refinement as a complementary direction for future work; App. C.2 reports feed-forward performance under an uncalibrated-camera setting. We train on the DL3DV (Ling et al., 2024) training split and evaluate on its test split under two complementary regimes. The low-resolution regime follows the sparse-view protocol of DepthSplat (Xu et al., 2025), using 2, 4, or 6 nearby context views at $256 \times 448$. The high-resolution regime uses 16, 32, or 64 context views sampled across the full sequence at $540 \times 960$ to test wide-coverage reconstruction and scaling with additional observations. To assess out-of-distribution (OOD) generalization, we additionally evaluate on Mip-NeRF 360 (Barron et al., 2022), the Free dataset from F$^2$-NeRF (Wang et al., 2023), and Tanks & Temples (Knapitsch et al., 2017). We use 16 and 64 context views, center-crop each image, and then resize it to $540 \times 960$.

Table 1: Quantitative comparison on DL3DV and out-of-distribution datasets. Context time is measured on a single NVIDIA H100 GPU, and $E$ denotes the number of learned-Adam refinement epochs. Concurrent methods are shown in gray and excluded from ranking. First-, second-, and third-ranked results within each setting are highlighted in red, orange, and yellow, respectively.

**(a) DL3DV dataset**

| Method | PSNR ↑ | SSIM ↑ | LPIPS ↓ | #GS (M) ↓ | Context (s) ↓ | Render FPS ↑ |
|---|---|---|---|---|---|---|
| **Short sequence, low resolution** | *2 / 4 / 6 views at* 256×448 | | | | | |
| MVSplat (Chen et al., 2024) | 17.54 / 21.63 / 22.93 | 0.529 / 0.721 / 0.775 | 0.402 / 0.233 / 0.193 | 0.23 / 0.46 / 0.69 | 0.10 / 0.15 / 0.23 | 352 / 305 / 253 |
| DepthSplat (Xu et al., 2025) | 19.20 / 22.16 / 23.00 | 0.610 / 0.734 / 0.764 | 0.307 / 0.188 / 0.161 | 0.23 / 0.46 / 0.69 | 0.10 / 0.11 / 0.13 | 349 / 278 / 230 |
| TokenGS (Ren et al., 2026) | 20.07 / 23.11 / 24.09 | 0.636 / 0.746 / 0.776 | 0.420 / 0.319 / 0.297 | 0.26 / 0.26 / 0.26 | 0.14 / 0.17 / 0.22 | 547 / 587 / 587 |
| Ours ($E$=0) | 22.38 / 25.84 / 27.10 | 0.725 / 0.835 / 0.864 | 0.258 / 0.165 / 0.140 | | 0.13 / 0.15 / 0.16 | 565 / 543 / 528 |
| Ours ($E$=1) | 22.69 / 26.50 / 28.04 | 0.736 / 0.851 / 0.884 | 0.251 / 0.155 / 0.127 | 0.12 / 0.17 / 0.20 | 0.15 / 0.17 / 0.25 | 562 / 520 / 530 |
| Ours ($E$=16) | 23.06 / 27.56 / 29.68 | 0.750 / 0.876 / 0.913 | 0.232 / 0.127 / 0.097 | | 0.34 / 0.34 / 0.48 | 568 / 547 / 534 |
| **Full sequence, high resolution** | *16 / 32 / 64 views at* 540×960 | | | | | |
| Long-LRM (Chen et al., 2025) | 21.30 / 23.67 / 23.65 | 0.713 / 0.770 / 0.787 | 0.332 / 0.279 / 0.267 | 4.2 / 8.4 / 16.7 | 0.8 / 1.3 / 2.8 | 210 / 135 / 84 |
| iLRM (Kang et al., 2026a) | 22.75 / 24.12 / 24.28 | 0.757 / 0.794 / 0.811 | 0.303 / 0.265 / 0.248 | 2.1 / 4.2 / 8.4 | 0.3 / 0.8 / 2.7 | 191 / 140 / 90 |
| tttLRM (Wang et al., 2026a) | 23.59 / 25.05 / 25.92 | 0.783 / 0.820 / 0.842 | 0.258 / 0.220 / 0.200 | 3.8 / 7.6 / 11.0 | 2.6 / 4.6 / 9.1 | 131 / 81 / 59 |
| MVP (Kang et al., 2026b) | 23.58 / 25.73 / 27.44 | 0.789 / 0.839 / 0.873 | 0.248 / 0.197 / 0.165 | 8.4 / 16.7 / 33.4 | 0.1 / 0.2 / 0.4 | 195 / 138 / 89 |
| MVP + post-opt. 16 epochs | 23.71 / 25.13 / 24.36 | 0.791 / 0.818 / 0.789 | 0.240 / 0.232 / 0.290 | | 6.0 / 20.2 / 72.3 | 153 / 94 / 54 |
| ReSplat (Xu et al., 2026) | 24.34 / 25.77 / 26.44 | 0.802 / 0.838 / 0.856 | 0.260 / 0.221 / 0.204 | 0.6 / 1.1 / 2.2 | 1.8 / 5.0 / 16.4 | 346 / 268 / 186 |
| AdaptSplat (Xing et al., 2026) | 23.88 / 26.05 / 27.69 | 0.804 / 0.852 / 0.883 | 0.237 / 0.186 / 0.154 | 8.4 / 16.7 / 33.4 | 0.3 / 0.6 / 1.4 | 115 / 70 / 40 |
| Ours ($E$=0) | 23.99 / 24.94 / 25.44 | 0.792 / 0.813 / 0.824 | 0.261 / 0.238 / 0.225 | | 1.0 / 2.4 / 6.4 | 273 / 248 / 225 |
| Ours ($E$=1) | 24.70 / 25.94 / 26.49 | 0.815 / 0.844 / 0.860 | 0.248 / 0.220 / 0.206 | 1.2 / 1.7 / 2.2 | 1.3 / 2.8 / 7.4 | 274 / 250 / 228 |
| Ours ($E$=16) | 25.73 / 27.43 / 28.37 | 0.846 / 0.879 / 0.899 | 0.219 / 0.190 / 0.175 | | 3.8 / 8.9 / 22.0 | 281 / 257 / 241 |

**(b) Out-of-distribution datasets**

| | Mip-NeRF 360 | | | Free | | | Tanks & Temples | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | PSNR ↑ | #GS (M) ↓ | Context (s) ↓ | PSNR ↑ | #GS (M) ↓ | Context (s) ↓ | PSNR ↑ | #GS (M) ↓ | Context (s) ↓ |
| **Full sequence, high resolution** | *16 / 64 views, center-cropped and resized to* 540×960 | | | | | | | | |
| Long-LRM (Chen et al., 2025) | 19.34 / 21.64 | 4.18 / 16.71 | 1.3 / 2.3 | 20.32 / 23.67 | 4.18 / 16.71 | 0.3 / 2.2 | 17.45 / 20.02 | 4.18 / 16.71 | 0.3 / 2.2 |
| iLRM (Kang et al., 2026a) | 20.35 / 22.01 | 2.09 / 8.36 | 0.3 / 2.6 | 21.17 / 23.84 | 2.09 / 8.36 | 0.3 / 2.6 | 18.23 / 20.25 | 2.09 / 8.36 | 0.3 / 2.6 |
| tttLRM (Wang et al., 2026a) | 21.25 / 23.65 | 3.79 / 11.03 | 3.0 / 8.2 | 22.38 / 25.76 | 3.79 / 11.03 | 1.9 / 8.1 | 18.90 / 21.55 | 3.79 / 11.03 | 1.9 / 8.1 |
| MVP (Kang et al., 2026b) | 20.71 / 24.27 | 8.36 / 33.42 | 0.1 / 0.4 | 22.12 / 26.50 | 8.36 / 33.42 | 0.1 / 0.4 | 18.79 / 22.02 | 8.36 / 33.42 | 0.1 / 0.4 |
| ReSplat (Xu et al., 2026) | 22.22 / 24.45 | 0.55 / 2.21 | 1.8 / 15.8 | 23.94 / 27.51 | 0.55 / 2.21 | 1.7 / 15.7 | 19.54 / 21.73 | 0.55 / 2.21 | 1.7 / 15.9 |
| AdaptSplat (Xing et al., 2026) | 20.95 / 24.54 | 8.36 / 33.42 | 0.3 / 1.2 | 21.21 / 24.91 | 8.36 / 33.42 | 0.3 / 1.2 | 18.64 / 21.92 | 8.36 / 33.42 | 0.3 / 1.1 |
| Ours ($E$=0) | 22.42 / 23.83 | | 1.3 / 7.3 | 23.35 / 25.00 | | 1.1 / 6.9 | 19.73 / 21.35 | | 1.2 / 8.5 |
| Ours ($E$=1) | 22.94 / 25.18 | 1.63 / 3.14 | 1.5 / 8.9 | 23.85 / 26.40 | 1.55 / 2.83 | 1.4 / 8.5 | 20.02 / 22.28 | 1.88 / 4.28 | 1.5 / 10.6 |
| Ours ($E$=16) | 23.61 / 27.07 | | 4.8 / 29.5 | 24.72 / 28.59 | | 4.8 / 29.0 | 20.18 / 23.26 | | 5.5 / 38.8 |

**Hyperparameter setup.** We employ Pi3X (Wang et al., 2026c) as our geometry backbone and condition it on the known input cameras. We impose a feed-forward pixel budget of $N_{\mathcal{B}}$=255K on the entire feed-forward pipeline. We use image patch size $p$=14, target voxel footprint $p_v$=14, voxel-retention threshold $\tau$=4, and $K$=32 Gaussians per retained voxel. Our Transformer decoder has nine layers, a hidden dimension of 768, and 12 attention heads. It uses pre-normalization (Xiong et al., 2020) and query–key normalization (Henry et al., 2020) for stable training. We represent view-dependent color using spherical harmonics of degree one. Learned Adam always uses source-resolution images as refinement feedback. We report the feed-forward prediction ($E$=0) and refined results after $E$>0 epochs, with the optimizer-state handoff at $E_{\text{swap}}$=8.

### 4.2 Comparison with state of the art

**Baselines.** In the sparse-view, low-resolution regime, we compare against the pixel-based MVSplat (Chen et al., 2024) and DepthSplat (Xu et al., 2025), as well as the codebook-based TokenGS (Ren et al., 2026). For wide-coverage and OOD evaluation, we compare against Long-LRM (Chen et al., 2025), iLRM (Kang et al., 2026a), tttLRM (Wang et al., 2026a), and MVP (Kang et al., 2026b). Under our allocation taxonomy, these wide-coverage methods are pixel-based: their Gaussian decoding sites remain tied to per-view spatial grids, even when the model compresses its tokens or prunes its outputs. We also report the concurrent pixel-based ReSplat (Xu et al., 2026) and AdaptSplat (Xing et al., 2026) in gray but exclude them from the ranking.

**DL3DV.** As shown in Tab. 1(a), our feed-forward prediction outperforms prior methods in the sparse-view regime and remains competitive over full sequences while using substantially fewer Gaussians. Learned Adam consistently improves quality, achieving the strongest results in most settings while preserving high rendering throughput. Even when concurrent ReSplat and AdaptSplat are included, our refined model achieves the highest PSNR in every reported full-sequence and OOD setting.

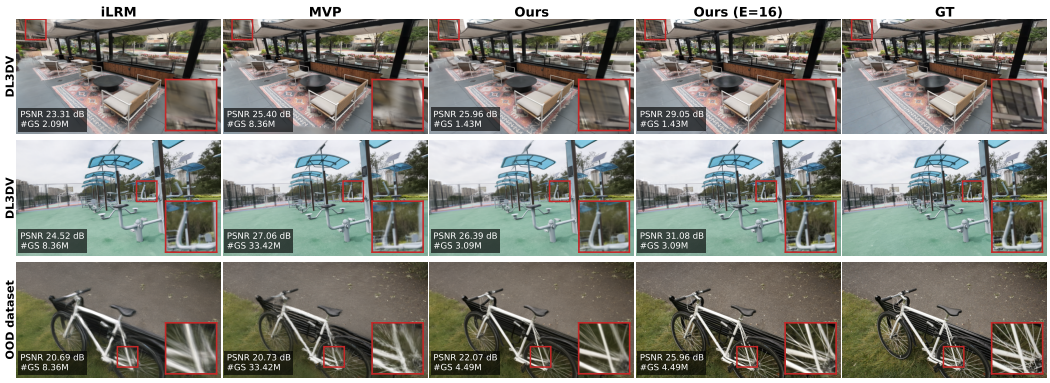


Figure 5: Qualitative comparison on DL3DV and out-of-distribution scenes.

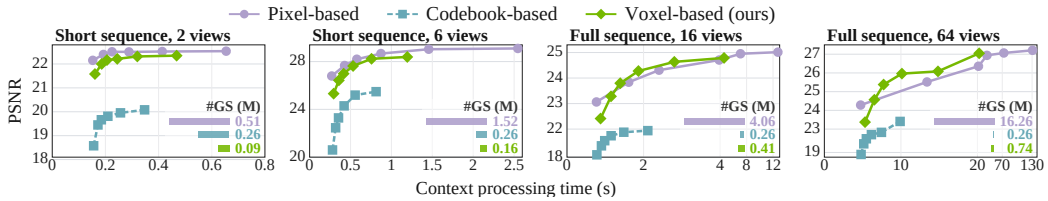


Figure 6: Quality versus total context processing time for pixel-, codebook-, and voxel-based Gaussian allocation. Each curve connects results after 0, 1, 2, 4, 8, and 16 epochs of learned Adam refinement.

We additionally apply cold-start Adam to MVP, the strongest feed-forward baseline. Its large Gaussian count makes refinement more than $3\times$ slower than ours at 64 views, while its novel-view quality degrades, suggesting that its overparameterized representation is prone to context-view overfitting. This result highlights the advantage of our efficient voxel-based allocation for post-optimization.

**Full-sequence processing time.** Our feed-forward context processing is slower than several baselines whose architectures are specifically designed for efficient long-context inference. Our current pipeline retains a standard self-attention Transformer. Exploring more efficient long-context architectures within our voxel-adaptive pipeline is a promising direction for future work.

**Out-of-distribution generalization.** The same trend holds across all three OOD datasets in Tab. 1(b): our feed-forward prediction leads at 16 views, and learned Adam closes the remaining 64-view gap after one epoch on nearly every dataset. Longer refinement achieves the strongest quality throughout while retaining a markedly smaller Gaussian representation. Fig. 5 shows that our compact feed-forward reconstruction provides a strong initialization and learned Adam further sharpens local details; more results are provided in App. C.3.

### 4.3 Ablation Studies

We conduct controlled experiments on the DL3DV dataset. Unless stated otherwise, we evaluate models with a 255k-pixel feed-forward budget. A comprehensive study of our hyperparameter configuration is provided in App. C.1; below, we focus on the ablations most relevant to our design.

**Allocation strategy.** Fig. 6 compares the quality–efficiency trade-off of the three allocation strategies. In our unified implementation, they differ only in their input token types and the output tokens used for Gaussian decoding, as illustrated in Fig. 1. The pixel-based decoder achieves the strongest feed-forward quality, but its Gaussian count grows with the number of context views, making refinement increasingly expensive. The codebook-based decoder remains efficient through a fixed Gaussian budget, but this fixed capacity limits reconstruction quality, particularly as scene coverage increases. In contrast, our voxel-based decoder allocates substantially fewer Gaussians according to occupied 3D space and scales more favorably as the context-view count grows.

Learned Adam improves all three decoders and substantially narrows the initial quality gap between pixel- and voxel-based allocation. For full-sequence inputs, our voxel-based decoder approaches the quality of the pixel-based decoder with substantially less processing time while using only about 5–10% of the Gaussians. For short sequences, voxel-based allocation outperforms codebook-based

**(a) Learned Adam state ablation**

| Pred. | Novel-view PSNR ↑ $E$=1 | $E$=2 | $E$=4 | $E$=8 |
|---|---|---|---|---|
| Cold-start | 23.54 (+0.00) | 23.98 (+0.00) | 24.44 (+0.00) | 24.93 (+0.00) |
| $m, v$ | 23.71 (+0.17) | 24.19 (+0.21) | 24.65 (+0.21) | 25.09 (+0.16) |
| $\eta$ | 23.84 (+0.30) | 24.29 (+0.31) | 24.72 (+0.28) | 25.02 (+0.09) |
| $v, \eta$ | **24.20** (+0.66) | 24.84 (+0.86) | 25.39 (+0.95) | 25.83 (+0.90) |
| $m, v, \eta$ | 24.18 (+0.64) | **24.87** (+0.89) | **25.46** (+1.02) | **25.89** (+0.96) |
| ⊢ Scene-level | 23.92 (+0.38) | 24.50 (+0.52) | 25.02 (+0.58) | 25.48 (+0.55) |
| ⌞ Global-level | 23.32 (-0.22) | 23.55 (-0.43) | 23.86 (-0.58) | 24.25 (-0.68) |

**(b) Long-horizon post-optimization evaluation**

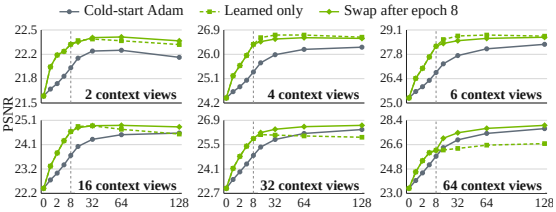


Figure 7: Learned Adam ablations. **(a)** State-prediction and state-granularity ablations with 32 context views, where $m$, $v$, and $\eta$ denote Adam's first moment, second moment, and learning rate, respectively, and $E$ denotes the number of post-optimization epochs. **(b)** Post-optimization improvement through 128 epochs across all six context-view settings. Learned Adam improves much faster in every setting but saturates with denser context.

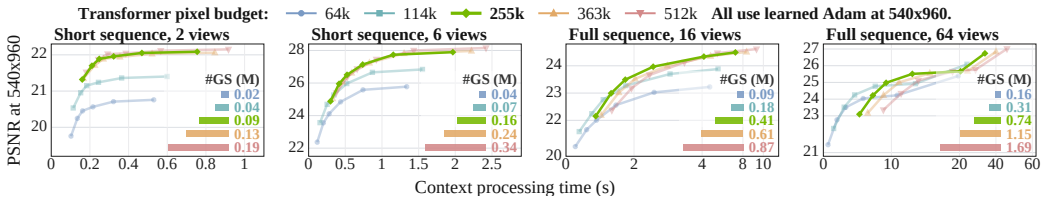


Figure 8: Novel-view quality versus total context processing time for different Transformer pixel budgets. Learned Adam and novel-view evaluation always operate at the source resolution of $540\times960$. Each curve connects results after 0, 1, 2, 4, 8, and 16 epochs of learned Adam refinement.

allocation while using fewer Gaussians, indicating that aligning decoding tokens with occupied 3D space provides a useful inductive bias. Overall, voxel-based allocation with learned Adam offers a better quality–efficiency trade-off than the pixel- and codebook-based strategies.

**Learned Adam design.** We ablate both the predicted Adam state and its granularity against cold-start Adam in Fig. 7(a). Predicting the full state yields the largest gain, with the second moment and learning rate contributing most of the improvement. Sharing a predicted state across the Gaussians within each scene reduces the gain, while a single learned state shared across scenes does not outperform cold-start Adam. Fig. 7(b) compares learned and cold-start Adam across the six context-view settings for up to 128 epochs. Learned Adam improves much faster in every setting, although learned-only refinement saturates when more context views are available. Handing off to the empirically accumulated Adam state at $E_{\text{swap}}$=8 restores long-horizon progress. We therefore use this handoff strategy in all other experiments.

**Resolution-decoupling study.** Fig. 8 reveals a clear quality–efficiency knee near a 255k-pixel feed-forward budget. Increasing the budget from 64k or 114k pixels to 255k yields substantial gains. Further increasing it to 363k or 512k pixels, however, provides only marginal improvements both before and after learned Adam. For full-sequence inputs, the 255k-pixel model with learned Adam matches the PSNR of the larger budgets at lower processing cost. Thus, increasing the Transformer pixel budget helps only up to a certain point—255k pixels in our study—beyond which learned Adam more effectively exploits high-resolution context information.

## 5 Conclusion

We presented SplatFoundry, a framework for scalable feed-forward Gaussian reconstruction. Its voxel-adaptive decoder allocates a compact representation according to occupied 3D space, while learned Adam rapidly refines it using context-view feedback. AAJ enables memory-efficient back-propagation through the Adam refinement trajectory. Our resolution-decoupled strategy caps the feed-forward resolution while learned Adam recovers detail from high-resolution images. Comparisons with state-of-the-art methods on DL3DV and three out-of-distribution datasets show that SplatFoundry improves PSNR by 2.5 dB and uses $6.2\times$ fewer Gaussians on average. Code will be released.

### AI USE STATEMENT

In this work, we used generative AI tools to implement and debug new features specified by the authors and to assist with interpreting and presenting experimental results. The original codebase, the proposed method, and all experiments were designed by the authors. We did not use generative AI to propose hypotheses, design the research methodology or experiments. Additionally, we used generative AI to draft and edit the manuscript, organize the LaTeX source, locate and summarize relevant literature, prepare tables and scientific figures, and write analysis and plotting scripts. The first author traced and manually verified all AI-assisted code changes and checked every AI-suggested citation against its original source. All co-authors proofread the complete manuscript. The authors take responsibility for the final content of this work, including text, claims, code, and artifacts produced with the aid of generative AI.

## REFERENCES

Honggyu An, Jaewoo Jung, Mungyeom Kim, Sunghwan Hong, Chaehyun Kim, Kazumi Fukuda, Minkyeong Jeon, Jisang Han, Takuya Narihira, Hyuna Ko, Junsu Kim, Yuki Mitsufuji, and Seungryong Kim. C3G: Learning compact 3d representations with 2k gaussians. In *CVPR*, 2026.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, pp. 5470–5479, 2022.

David Charatan, Sizhe Lester Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelSplat: 3d gaussian splats from image pairs for scalable generalizable 3d reconstruction. In *CVPR*, pp. 19457–19467, 2024.

Yuedong Chen, Haofei Xu, Chuanxia Zheng, Bohan Zhuang, Marc Pollefeys, Andreas Geiger, Tat-Jen Cham, and Jianfei Cai. MVSplat: Efficient 3d gaussian splatting from sparse multi-view images. In *ECCV*, pp. 370–386, 2024.

Ziwen Chen, Hao Tan, Kai Zhang, Sai Bi, Fujun Luan, Yicong Hong, Fuxin Li, and Zexiang Xu. Long-LRM: Long-sequence large reconstruction model for wide-coverage gaussian splats. In *ICCV*, pp. 4349–4359, 2025.

Xianglong He, Junyi Chen, Sida Peng, Di Huang, Yangguang Li, Xiaoshui Huang, Chun Yuan, Wanli Ouyang, and Tong He. GVGEN: Text-to-3d generation with volumetric representation. In *ECCV*, 2024.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of EMNLP*, pp. 4246–4253, 2020. doi: 10.18653/v1/2020.findings-emnlp.379.

Roni Itkin, Noam Issachar, Yehonatan Keypur, Xingyu Chen, Anpei Chen, and Sagie Benaim. GlobalSplat: Efficient feed-forward 3d gaussian splatting via global scene tokens. In *ECCV*, 2026.

Lihan Jiang, Yucheng Mao, Linning Xu, Tao Lu, Kerui Ren, Yichen Jin, Xudong Xu, Mulin Yu, Jiangmiao Pang, Feng Zhao, Dahua Lin, and Bo Dai. AnySplat: Feed-forward 3d gaussian splatting from unconstrained views. *ACM Transactions on Graphics*, 44(6):1–16, 2025. doi: 10.1145/3763326.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, pp. 694–711, 2016.

Gyeongjin Kang, Seungtae Nam, Seungkwon Yang, Xiangyu Sun, Sameh Khamis, Abdelrahman Mohamed, and Eunbyung Park. iLRM: An iterative large 3d reconstruction model. In *CVPR*, pp. 37332–37342, 2026a.

Gyeongjin Kang, Seungkwon Yang, Seungtae Nam, Younggeun Lee, Jungwoo Kim, and Eunbyung Park. Multi-view pyramid transformer: Look coarser to see broader. In *CVPR*, 2026b.

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

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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

Yuke Li, Weihang Liu, Cheng Zhang, Yuefeng Zhang, Jiadi Cui, Zixuan Wang, Junran Ding, Haoyu Wu, Yujiao Shi, Jingyi Yu, and Xin Lou. ForeSplat: Optimization-aware foresight for feed-forward 3d gaussian splatting. *arXiv preprint arXiv:2605.22020*, 2026.

Chin-Yang Lin, Cheng Sun, Fu-En Yang, Min-Hung Chen, Yen-Yu Lin, and Yu-Lun Liu. LongSplat: Robust unposed 3d gaussian splatting for casual long videos. In *ICCV*, pp. 27412–27422, 2025.

Haotong Lin, Sili Chen, Jun Hao Liew, Donny Y. Chen, Zhenyu Li, Yang Zhao, Sida Peng, Hengkai Guo, Xiaowei Zhou, Guang Shi, Jiashi Feng, and Bingyi Kang. Depth Anything 3: Recovering the visual space from any views. In *ICLR*, 2026.

Lu Ling, Yichen Sheng, Zhi Tu, Wentian Zhao, Cheng Xin, Kun Wan, Lantao Yu, Qianyu Guo, Zixun Yu, Yawen Lu, Xuanmao Li, Xingpeng Sun, Rohan Ashok, Aniruddha Mukherjee, Hao Kang, Xiangrui Kong, Gang Hua, Tianyi Zhang, Bedrich Benes, and Aniket Bera. DL3DV-10K: A large-scale scene dataset for deep learning-based 3d vision. In *CVPR*, pp. 22160–22169, 2024.

Yueh-Cheng Liu, Lukas Höllein, Matthias Nießner, and Angela Dai. QuickSplat: Fast 3d surface reconstruction via learned gaussian initialization. In *ICCV*, pp. 27851–27861, 2025.

Yueh-Cheng Liu, Jozef Hladký, Matthias Nießner, and Angela Dai. Diff3R: Feed-forward 3d gaussian splatting with uncertainty-aware differentiable optimization. *arXiv preprint arXiv:2604.01030*, 2026.

Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-GS: Structured 3d gaussians for view-adaptive rendering. In *CVPR*, pp. 20654–20664, 2024.

Jiawei Ren, Michal Jan Tyszkiewicz, Jiahui Huang, and Zan Gojcic. TokenGS: Decoupling 3d gaussian prediction from pixels with learnable tokens. In *CVPR*, pp. 15365–15375, 2026.

Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-GS: Towards consistent real-time rendering with LOD-structured 3d gaussians. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025. doi: 10.1109/TPAMI.2025.3568201.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Brandon Smart, Chuanxia Zheng, Iro Laina, and Victor Adrian Prisacariu. Learning compact 3d gaussians via feed-forward point fusion. In *3DV*, pp. 554–564, 2026. doi: 10.1109/3DV69130.2026.00059.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Chen Wang, Hao Tan, Yifan Wang, Zhiqin Chen, Yuheng Liu, Kalyan Sunkavalli, Sai Bi, Lingjie Liu, and Yiwei Hu. tttLRM: Test-time training for long context and autoregressive 3d reconstruction. In *CVPR*, pp. 36582–36592, 2026a.

Peng Wang, Yuan Liu, Zhaoxi Chen, Lingjie Liu, Ziwei Liu, Taku Komura, Christian Theobalt, and Wenping Wang. $F^2$-NeRF: Fast neural radiance field training with free camera trajectories. In *CVPR*, pp. 4150–4159, 2023.

Weijie Wang, Yeqing Chen, Zeyu Zhang, Hengyu Liu, Haoxiao Wang, Zhiyuan Feng, Wenkang Qin, Feng Chen, Jia-Wang Bian, Zheng Zhu, Donny Y. Chen, and Bohan Zhuang. VolSplat: Rethinking feed-forward 3d gaussian splatting with voxel-aligned prediction. In *ECCV*, 2026b.

Yifan Wang, Jianjun Zhou, Haoyi Zhu, Wenzheng Chang, Yang Zhou, Zizun Li, Junyi Chen, Jiangmiao Pang, Chunhua Shen, and Tong He. $\pi^3$: Permutation-equivariant visual geometry learning. In *ICLR*, 2026c.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. In *CVPR*, pp. 21469–21480, 2025.

Mingwei Xing, Xinliang Wang, and Yifeng Shi. AdaptSplat: Adapting vision foundation models for feed-forward 3d gaussian splatting. *arXiv preprint arXiv:2605.10239*, 2026.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *ICML*, pp. 10524–10533, 2020.

Haofei Xu, Songyou Peng, Fangjinhua Wang, Hermann Blum, Daniel Barath, Andreas Geiger, and Marc Pollefeys. DepthSplat: Connecting gaussian splatting and depth. In *CVPR*, pp. 16453–16463, 2025.

Haofei Xu, Daniel Barath, Andreas Geiger, and Marc Pollefeys. ReSplat: Learning recurrent gaussian splatting. In *ECCV*, 2026.

Bowen Zhang, Yiji Cheng, Jiaolong Yang, Chunyu Wang, Feng Zhao, Yansong Tang, Dong Chen, and Baining Guo. GaussianCube: A structured and explicit radiance representation for 3d generative modeling. In *NeurIPS*, 2024a.

Kai Zhang, Sai Bi, Hao Tan, Yuanbo Xiangli, Nanxuan Zhao, Kalyan Sunkavalli, and Zexiang Xu. GS-LRM: Large reconstruction model for 3d gaussian splatting. In *ECCV*, pp. 1–19, 2024b.

# SUPPLEMENTARY MATERIAL

This supplementary material is organized into three sections. Sec. A discusses limitations and failure cases. Sec. B details calibrated camera alignment, model architecture, the expected voxel-footprint derivation, differentiable learned Adam and AAJ, and the progressive training curriculum. Sec. C presents model-setup ablations, feed-forward evaluation with uncalibrated cameras, qualitative comparisons, and a web viewer for side-by-side fly-through videos.

## A LIMITATIONS AND FAILURE CASES

**Dependence on geometric priors.** Our voxel construction relies on the geometry and features predicted by a frozen geometric backbone. Errors in this proxy geometry can assign observations to incorrect voxels, merge unrelated surfaces, or split a single surface across cells. Learned Adam can move and reshape the decoded Gaussians, but it does not revise the discrete token allocation and may therefore be unable to repair severe geometric or topological errors.

**Camera assumptions.** Our primary experiments assume calibrated context cameras. We align these cameras to the coordinate system predicted by the geometric backbone, but the current pipeline does not jointly refine camera poses and Gaussian attributes. In an early attempt, we extended the state estimator to predict Adam state for the cameras and jointly refined cameras and Gaussians. The improvement was not significant and refinement sometimes degraded rendering quality. The uncalibrated-camera evaluation in Sec. C.2 therefore relies on the backbone's pose estimates. More robust joint camera and scene refinement remains an important extension for less controlled captures.

**Long-context efficiency.** Adaptive voxel allocation reduces the number of decoded Gaussians, but our current Transformer still applies standard self-attention to the concatenated patch and voxel tokens. Its context-encoding cost consequently grows with the token sequence and is slower on full sequences than architectures designed specifically for efficient long-context inference. Sparse, hierarchical, streaming, or linear-attention variants could complement our allocation strategy.

**Refinement tradeoffs.** Learned Adam provides large gains within a few epochs, but learned-only refinement can saturate over longer horizons, especially with many context views. We mitigate this behavior by handing off to empirically accumulated Adam state after eight epochs. This handoff is a heuristic with a fixed scheduling choice. A learnable and differentiable optimizer-annealing procedure designed for long-horizon refinement may provide a more robust transition and further improve reconstruction quality.

**Scene assumptions.** Like standard 3D Gaussian reconstruction pipelines, our representation assumes a static scene and models view-dependent appearance with finite-order spherical harmonics. Dynamic objects, strong non-Lambertian effects, exposure changes, and other transient content can therefore produce inconsistent geometry or residual rendering artifacts.

## B IMPLEMENTATION DETAILS

### B.1 CALIBRATED CAMERA POSE ALIGNMENT

Our main setup assumes offline-calibrated context and held-out cameras. The geometric backbone nevertheless predicts geometry and poses in its own similarity coordinate system. When supported, we condition the backbone on the calibrated context cameras, but its predicted poses may not strictly follow the supplied coordinates. To compute the novel-view loss, we therefore estimate a coordinate transform from the calibrated context poses to the backbone-predicted poses and apply it to every calibrated camera, including the held-out camera. This operation aligns coordinate systems rather than estimating or refining camera poses. Let $(\mathbf{R}_i, \mathbf{t}_i)$ and $(\widehat{\mathbf{R}}_i, \widehat{\mathbf{t}}_i)$ denote the calibrated and backbone-predicted camera-to-world poses, respectively, for $n$ context views, where $\mathbf{R}_i$ is the rotation matrix and $\mathbf{t}_i$ is the camera center. We seek a single similarity transform

$$\mathbf{x}_{\text{pred}} = s\mathbf{R}_a\mathbf{x}_{\text{input}} + \mathbf{b}, \qquad s > 0, \quad \mathbf{R}_a \in \mathrm{SO}(3), \quad \mathbf{b} \in \mathbb{R}^3. \tag{9}$$

**Rotation from camera orientations.** The aligned orientations should satisfy $\widehat{\mathbf{R}}_i \approx \mathbf{R}_a \mathbf{R}_i$. We estimate their chordal least-squares alignment,

$$\mathbf{R}_a^* = \underset{\mathbf{R}\in \mathrm{SO}(3)}{\arg\min} \sum_i \|\widehat{\mathbf{R}}_i - \mathbf{R}\mathbf{R}_i\|_F^2. \tag{10}$$

Let $\mathbf{M} = \sum_i \widehat{\mathbf{R}}_i \mathbf{R}_i^\top$ and $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ be its singular value decomposition. Projection onto $\mathrm{SO}(3)$ gives

$$\mathbf{R}_a = \mathbf{U}\,\mathrm{diag}\big(1, 1, \det(\mathbf{U}\mathbf{V}^\top)\big)\,\mathbf{V}^\top. \tag{11}$$

The determinant correction excludes reflections. Unlike alignment based only on camera centers, this rotation remains observable for two-view, collinear, or coincident-center camera configurations as long as their orientations are valid.

**Scale and translation from camera centers.** After fixing $\mathbf{R}_a$, let $\bar{\mathbf{t}}$ and $\bar{\widehat{\mathbf{t}}}$ denote the mean calibrated and predicted camera centers over the context views, respectively. Defining $\boldsymbol{\delta}_i = \mathbf{t}_i - \bar{\mathbf{t}}$ and $\widehat{\boldsymbol{\delta}}_i = \widehat{\mathbf{t}}_i - \bar{\widehat{\mathbf{t}}}$, least squares gives

$$s = \frac{\sum_i (\mathbf{R}_a \boldsymbol{\delta}_i)^\top \widehat{\boldsymbol{\delta}}_i}{\sum_i \|\boldsymbol{\delta}_i\|_2^2}, \qquad \mathbf{b} = \bar{\widehat{\mathbf{t}}} - s\mathbf{R}_a \bar{\mathbf{t}}. \tag{12}$$

The aligned calibrated poses are therefore $\mathbf{R}_i' = \mathbf{R}_a \mathbf{R}_i$ and $\mathbf{t}_i' = s\mathbf{R}_a \mathbf{t}_i + \mathbf{b}$. These aligned context cameras condition our Transformer, and the same transform aligns the held-out camera used to render the novel-view objective. The context and held-out cameras thus share the coordinate system of the backbone geometry. We use this calibrated-camera protocol throughout the main experiments. The appendix-only uncalibrated evaluation in Sec. C.2 instead uses the backbone-predicted poses directly and is included as a preliminary extension.

### B.2 Architecture Details

**Patch-token camera encoding.** For every image pixel, we represent its camera ray by Plücker coordinates: the ray direction $\mathbf{d}$ and moment $\mathbf{o} \times \mathbf{d}$, where $\mathbf{o}$ is the camera center. Cropping and flattening this six-channel ray map over each image patch gives $R_i$ in Eq. 2. Its learned projection is added directly to the RGB and backbone-feature projections, so camera geometry is encoded without a separate camera token.

**Voxel-key positional encoding.** Let the sparse key of voxel $j$ be $\kappa_j = (\mathbf{c}_j, \ell_j)$, where $\mathbf{c}_j \in \mathbb{Z}^3$ is its integer cell coordinate. Before positional encoding, we map this key to the voxel center and cell width, $\mathbf{v}_j = 2^{\ell_j}(\mathbf{c}_j + \frac{1}{2}\mathbf{1})$ and $s_j = 2^{\ell_j}$, respectively. The notation $\mathrm{VoxelPE}(\kappa_j)$ in Eq. 3 is shorthand for applying a NeRF-style sinusoidal positional encoding with 10 frequency bands to the voxel center $\mathbf{v}_j$ and cell width $s_j$. We concatenate the encoded center and scale and project the result to the token dimension.

**Gaussian output parameterization.** For Gaussian $k$ decoded from voxel $j$, the head emits the unconstrained vector $\Theta_{jk} = (\Delta\mathbf{p}_{jk}, \mathbf{a}_{s,jk}, \mathbf{a}_{q,jk}, a_{o,jk}, \mathbf{a}_{c,jk})$. Before rasterization, the activation map $\mathcal{A}$ converts the prediction to Gaussian attributes, with the position offset defined relative to the voxel center $\mathbf{v}_j$:

$$\boldsymbol{\mu}_{jk} = \mathbf{v}_j + s_j \Delta\mathbf{p}_{jk}, \tag{13}$$

$$\mathbf{s}_{jk} = s_j\,\mathrm{softplus}(\mathbf{a}_{s,jk}), \tag{14}$$

$$\mathbf{q}_{jk} = \frac{\mathbf{a}_{q,jk}}{\|\mathbf{a}_{q,jk}\|_2}, \tag{15}$$

$$o_{jk} = \sigma(a_{o,jk}), \qquad \mathbf{c}_{jk} = \mathbf{a}_{c,jk}. \tag{16}$$

Accordingly, the main paper uses $\mathcal{R}(\Theta, C)$ as shorthand for $\mathcal{R}(\mathcal{A}(\Theta), C)$. The reference configuration initializes scale near $s_j/\sqrt{K}$, rotation to identity, and opacity to $0.1$. Degree-zero color and higher-order spherical-harmonic coefficients are passed to the rasterizer without an activation.

### B.3 Token-Scaling Protocol

For the fixed-coverage statistic in Fig. 3, we sample context sets from the first 50 frames of each DL3DV scene, so added views primarily revisit the same region. For expanding coverage, we uniformly sample 64 frames over the full sequence and progressively add them in temporal order. We set $p = p_v = 16$ so that patch and voxel tokens have matched image footprints.

### B.4 Expected Voxel Footprint

We derive the calibration used by the level selection in Sec. 3.2 for a general geometric grid hierarchy. Let adjacent levels differ in cell width by a factor $b > 1$, so level $\ell$ has world-space cell width $b^\ell$. Recall that $\delta = 1/\sqrt{f_x f_y}$ approximates the angular width of one pixel from the focal lengths $f_x$ and $f_y$. At source-view depth $D$, one pixel therefore spans approximately $D\delta$ world-space units. For a target projected voxel width of $p_v$ pixels, we introduce an unknown calibration constant $\gamma_b$ and select

$$\widetilde{s} = D\delta p_v \gamma_b, \qquad \ell = \lceil \log_b \widetilde{s} \rceil . \tag{17}$$

Let $x = \log_b \widetilde{s}$ and $u = \mathrm{frac}(x) \in [0, 1)$ denote its fractional part. The realized projected cell width in pixels is

$$\begin{aligned} p_{\mathrm{proj}} &= \frac{b^\ell}{D\delta} = \frac{b^{\lceil \log_b \widetilde{s} \rceil}}{D\delta} \\ &= \frac{\widetilde{s}}{D\delta} b^{\lceil \log_b \widetilde{s} \rceil - \log_b \widetilde{s}} = p_v \gamma_b b^{\lceil x \rceil - x}. \end{aligned} \tag{18}$$

Except for the measure-zero case $u = 0$, we have $\lceil x \rceil - x = 1 - u$. Assuming the logarithmic phase is uniformly distributed, $u \sim \mathcal{U}[0, 1)$, gives

$$\mathbb{E}[p_{\mathrm{proj}}] = p_v \gamma_b \int_0^1 b^{1-u} \, \mathrm{d}u = p_v \gamma_b \frac{b-1}{\ln b}. \tag{19}$$

Enforcing the target expected footprint $\mathbb{E}[p_{\mathrm{proj}}] = p_v$ yields $\gamma_b = \ln b/(b-1)$. Our octree uses $b = 2$, for which $\gamma_2 = \ln 2$, recovering Eq. 1.

### B.5 Differentiable Learned Adam

**Learned Adam state stabilization.** The state estimator predicts log-domain second moments and learning rates before the mappings in Eq. 6. We clip the log second moment to $[\log 10^{-16}, \log 10^{2}]$ and the log learning rate to $[\log 10^{-8}, \log 10^{-1}]$ before exponentiation. The final prediction layer is initialized with zero weights. Its first-moment-logit bias is zero, its log-second-moment bias is $\log 10^{-4}$, and its log-learning-rate bias is initialized to the logarithms of the standard attribute-wise 3DGS rates (Kerbl et al., 2023). This bias initialization subsumes the otherwise equivalent addition of a base log learning rate. Consequently, training begins from $m_0 = 0$, $v_0 = 10^{-4}$, and standard group-wise learning rates without arbitrary spatial variation.

**Differentiable Adam update.** For each active Gaussian parameter $\Theta_t$, the detached context gradient $g_t$ advances the learned Adam state elementwise as

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) g_t, \tag{20}$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2) g_t^2, \tag{21}$$

$$\Theta_{t+1} = \Theta_t - \eta \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon}. \tag{22}$$

We use $\beta_1 = 0.9$, $\beta_2 = 0.99$, and $\epsilon = 10^{-15}$ without bias correction because both moments are learned initial conditions. Following sparse Adam, we skip both the parameter and moment updates for entries with zero rendering gradient, preventing invisible Gaussians from drifting.

### B.6 Accumulated Adam Jacobian

The detached context gradients permit an exact backward pass without retaining the full refinement trajectory. We call this procedure the *Accumulated Adam Jacobian* (AAJ). We derive the result for one scalar Gaussian parameter; all operations apply elementwise to the $G$ raw Gaussian parameters. For $t = 0, \ldots, T-1$, learned Adam performs

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1 - \beta_1) g_t, & v_{t+1} &= \beta_2 v_t + (1 - \beta_2) g_t^2, \\ d_{t+1} &= \sqrt{v_{t+1}} + \epsilon, & \Theta_{t+1} &= \Theta_t - \eta m_{t+1}/d_{t+1}, \end{aligned} \tag{23}$$

Table 2: Complexity and measured cost of differentiable learned Adam. The inner rasterizer graph is discarded after each gradient extraction in both versions. Complexity omits the common feed-forward prediction. Measurements use 32 context views and one post-optimization epoch (32 iterations) at $378\times672$, averaged over five paired DL3DV scenes. Peak memory is the maximum allocated-memory increment during post-optimization after feed-forward prediction.

| | Outer-backward time | | Peak memory | |
|---|---|---|---|---|
| Implementation | Complexity | Measured | Complexity | Measured |
| Autograd unroll | $O(TG)$ | 0.127 s | $O(M_R + TG)$ | 29.44 GiB |
| AAJ | $O(G)$ | 0.015 s | $O(M_R + G)$ | 8.75 GiB |

where $g_t$ is detached as in Eq. 7. Consequently, for the outer training gradient, the sequence $\{g_t\}_{t=0}^{T-1}$ is constant. The moments are therefore affine in their predicted initial states:

$$m_t = \beta_1^t m_0 + C_t^m, \qquad C_t^m = (1-\beta_1)\sum_{j=0}^{t-1}\beta_1^{t-1-j} g_j,$$
$$v_t = \beta_2^t v_0 + C_t^v, \qquad C_t^v = (1-\beta_2)\sum_{j=0}^{t-1}\beta_2^{t-1-j} g_j^2. \tag{24}$$

Here, $C_t^m$ and $C_t^v$ depend on the detached gradients but have zero outer derivative.

**AAJ forward accumulation.** To obtain $\Theta_T$ and its derivatives without storing the refinement trajectory, AAJ maintains three accumulators: $S_t$ sums the normalized Adam updates, while $A_t$ and $B_t$ accumulate their sensitivities to the predicted initial moments $m_0$ and $v_0$, respectively. Let $P_t{=}\partial m_t/\partial m_0$ and $Q_t{=}\partial v_t/\partial v_0$ track the corresponding moment sensitivities. Starting from $P_0{=}Q_0{=}1$ and $S_0{=}A_0{=}B_0{=}0$, each forward iteration updates

$$\begin{aligned} P_{t+1} &= \beta_1 P_t, & Q_{t+1} &= \beta_2 Q_t, \\ S_{t+1} &= S_t + m_{t+1}/d_{t+1}, & A_{t+1} &= A_t + P_{t+1}/d_{t+1}, \\ B_{t+1} &= B_t + \frac{m_{t+1} Q_{t+1}}{2\sqrt{v_{t+1}}\, d_{t+1}^2}. \end{aligned} \tag{25}$$

The final Gaussian parameter and its elementwise derivatives are then

$$\Theta_T = \Theta_0 - \eta S_T, \qquad \frac{\partial \Theta_T}{\partial \Theta_0} = 1, \qquad \frac{\partial \Theta_T}{\partial m_0} = -\eta A_T, \qquad \frac{\partial \Theta_T}{\partial v_0} = \eta B_T, \qquad \frac{\partial \Theta_T}{\partial \eta} = -S_T, \tag{26}$$

Thus, given the upstream gradient $h{=}\partial\mathcal{L}/\partial\Theta_T$, the custom backward directly returns $h$, $-h\eta A_T$, $h\eta B_T$, and $-hS_T$ for $\Theta_0$, $m_0$, $v_0$, and $\eta$, respectively. Only the three nontrivial derivative factors in Eq. 26 need to be retained after the forward pass.

**Complexity.** Let $G$ denote the total number of scalar Gaussian parameters and $M_R$ the peak space of one inner rendering vector–Jacobian product. Tab. 2 compares the two implementations. AAJ removes the factor $T$ from retained space, reducing it from $O(M_R + TG)$ to $O(M_R + G)$. In our measurements, this reduces peak memory by $70.3\%$.

### B.7 Progressive Training Curriculum

We train SplatFoundry on DL3DV using 64 NVIDIA H100 GPUs. For each training scene, we randomly sample between 2 and 32 context views. The geometry backbone remains frozen throughout training.

**Feed-forward Gaussian pretraining.** We first train only the Transformer and Gaussian decoder, without learned Adam. Training starts with a 65k-pixel feed-forward budget for $100{,}000$ iterations at a learning rate of $4\times10^{-4}$. We then increase the budget to 114k pixels and optimize for $10{,}000$ iterations at $1\times10^{-5}$, followed by another $10{,}000$ iterations at the final 255k-pixel budget and the same learning rate.

**End-to-end learned Adam fine-tuning.** After feed-forward pretraining, we attach the learned Adam head and fine-tune it jointly with the Transformer for $10{,}000$ iterations. We use a learning rate of $1\times10^{-5}$ for the pretrained Transformer and $1\times10^{-4}$ for the newly initialized learned Adam head.

**Loss weights.** We set the perceptual-loss weight to $\lambda_{\text{perc}}=0.2$ throughout training. In Eq. 8, the novel-view losses for the feed-forward and refined Gaussians both have unit weight, i.e., $\lambda_{\text{pre}}=\lambda_{\text{post}}=1$ whenever the corresponding prediction is trained.

# C Additional Results

## C.1 Model Setup Ablations

We vary the image patch size $p$, target projected voxel footprint $p_v$ in pixels, Transformer depth $L$, number $K$ of Gaussians per retained voxel, and whether voxel metadata is injected into the initial voxel token, immediately before the Gaussian head, or at both locations.

Tab. 3 exposes several practical trade-offs. First, finer voxel footprints improve quality at the cost of more Gaussians and slower rendering. Smaller image patches provide a similar quality benefit but increase Transformer processing time. These two parameters therefore control complementary allocation and processing costs. We use $p=14$ and $p_v=14$ in our final model.

Second, decoder depth is the strongest architectural factor: nine layers recover most of the benefit of the deepest variant and offer a better accuracy–cost balance. In contrast, increasing the number of Gaussians per voxel contributes only marginal quality improvement while substantially increasing representation and rendering costs. Injecting voxel metadata into the initial token is important, whereas re-injecting it immediately before the Gaussian head contributes little. Voxel position and scale are therefore most useful while tokens are being contextualized rather than only during final attribute decoding. Our final model uses $L=9$, $K=32$, and injects voxel metadata only into the initial voxel tokens.

Higher-order spherical harmonics improve appearance modeling without changing allocation or context-processing cost. Most of the gain comes from introducing view-dependent color; increasing the degree further contributes only marginal improvements while continuing to reduce rendering throughput. We therefore use spherical harmonics of degree one in our final model.

The alternative-backbone results confirm that the decoder is not tied to a single geometry model, while also showing sensitivity to the supplied proxy geometry and features. Under the current training setups, the reference Pi3X backbone performs best; DA3 and VGGT-Omega are slightly less accurate, and DA3 is substantially slower. We therefore use Pi3X in our final model.

## C.2 Feed-Forward Evaluation with Uncalibrated Cameras

We additionally evaluate feed-forward reconstruction when no calibrated input cameras are supplied. In this setting, Pi3X estimates the cameras together with the proxy geometry, and we use these predicted cameras throughout our feed-forward pipeline. We report only feed-forward results and leave joint refinement of camera and Gaussian parameters under this setting to future work.

Tab. 4 shows that our feed-forward model consistently achieves the strongest reconstruction quality while using fewer Gaussians than the baselines under the uncalibrated-camera setting. This result indicates that adaptive voxel decoding remains effective with backbone-predicted cameras.

Table 3: Model-setup ablation. Each metric reports its value and signed difference from the reference in adjacent columns. Efficiency is measured on the 32-view full-sequence setup; GS is in thousands and context time is in seconds.

| Model configuration | | Short-sequence PSNR ↑ | | | Full-sequence PSNR ↑ | | | Efficiency at 32 views | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 2 | 4 | 6 | 16 | 32 | 64 | GS (k) ↓ | Ctx. (s) ↓ | FPS ↑ |
| $p$=14, $p_v$=16, $L$=6, $K$=16 | | 21.58 (+0.00) | 24.39 (+0.00) | 25.32 (+0.00) | 22.41 (+0.00) | 23.04 (+0.00) | 23.37 (+0.00) | 569 (+0) | 1.9 (+0.0) | 409 (+0) |
| Voxel footprint | $p_v = 14$ | 21.69 (+0.11) | 24.60 (+0.21) | 25.57 (+0.25) | 22.56 (+0.15) | 23.21 (+0.17) | 23.55 (+0.18) | 790 (+221) | 2.1 (+0.2) | 364 (-45) |
| | $p_v = 12$ | 21.77 (+0.19) | 24.74 (+0.35) | 25.75 (+0.43) | 22.68 (+0.27) | 23.34 (+0.30) | 23.69 (+0.32) | 1166 (+597) | 2.4 (+0.5) | 312 (-97) |
| Patch size | $p = 8$ | 21.75 (+0.17) | 24.84 (+0.45) | 25.88 (+0.56) | 22.66 (+0.25) | 23.33 (+0.29) | 23.68 (+0.31) | 583 (+14) | 3.1 (+1.2) | 415 (+6) |
| Depth | $L = 3$ | 20.96 (-0.62) | 23.28 (-1.11) | 24.01 (-1.31) | 21.50 (-0.91) | 22.04 (-1.00) | 22.30 (-1.07) | 569 (+0) | 1.7 (-0.2) | 409 (+0) |
| | $L = 9$ | 21.79 (+0.21) | 24.79 (+0.40) | 25.81 (+0.49) | 22.75 (+0.34) | 23.42 (+0.38) | 23.77 (+0.40) | 569 (+0) | 2.1 (+0.2) | 412 (+3) |
| | $L = 12$ | 21.85 (+0.27) | 24.87 (+0.48) | 25.92 (+0.60) | 22.84 (+0.43) | 23.52 (+0.48) | 23.89 (+0.52) | 569 (+0) | 2.3 (+0.4) | 409 (+0) |
| GS per voxel | $K = 32$ | 21.71 (+0.13) | 24.59 (+0.20) | 25.57 (+0.25) | 22.55 (+0.14) | 23.19 (+0.15) | 23.52 (+0.15) | 1138 (+569) | 1.9 (+0.0) | 314 (-95) |
| | $K = 64$ | 21.73 (+0.15) | 24.63 (+0.24) | 25.63 (+0.31) | 22.57 (+0.16) | 23.22 (+0.18) | 23.56 (+0.19) | 2275 (+1706) | 1.9 (+0.0) | 233 (-176) |
| Voxel metadata | Head only | 21.29 (-0.29) | 23.95 (-0.44) | 24.85 (-0.47) | 22.12 (-0.29) | 22.74 (-0.30) | 23.07 (-0.30) | 569 (+0) | 1.9 (+0.0) | 415 (+6) |
| | Token only | 21.61 (+0.03) | 24.42 (+0.03) | 25.35 (+0.03) | 22.43 (+0.02) | 23.04 (+0.00) | 23.36 (-0.01) | 569 (+0) | 1.9 (+0.0) | 388 (-21) |
| | Neither | 21.18 (-0.40) | 23.82 (-0.57) | 24.71 (-0.61) | 22.07 (-0.34) | 22.70 (-0.34) | 23.04 (-0.33) | 569 (+0) | 1.9 (+0.0) | 413 (+4) |
| SH degree | $d_{\mathrm{SH}} = 1$ | 21.87 (+0.29) | 24.89 (+0.50) | 25.94 (+0.62) | 22.99 (+0.58) | 23.76 (+0.72) | 24.16 (+0.79) | 569 (+0) | 1.9 (+0.0) | 355 (-54) |
| | $d_{\mathrm{SH}} = 2$ | 21.86 (+0.28) | 24.88 (+0.49) | 25.95 (+0.63) | 23.06 (+0.65) | 23.84 (+0.80) | 24.27 (+0.90) | 569 (+0) | 1.9 (+0.0) | 344 (-65) |
| | $d_{\mathrm{SH}} = 3$ | 21.88 (+0.30) | 24.93 (+0.54) | 26.00 (+0.68) | 23.11 (+0.70) | 23.90 (+0.86) | 24.31 (+0.94) | 569 (+0) | 1.9 (+0.0) | 348 (-61) |
| Geometry backbone | DA3 | 21.48 (-0.10) | 24.16 (-0.23) | 25.06 (-0.26) | 22.22 (-0.19) | 22.81 (-0.23) | 23.12 (-0.25) | 672 (+103) | 7.6 (+5.7) | 419 (+10) |
| | VGGT-Omega | 21.43 (-0.15) | 24.08 (-0.31) | 24.96 (-0.36) | 22.12 (-0.29) | 22.70 (-0.34) | 23.00 (-0.37) | 758 (+189) | 1.2 (-0.7) | 362 (-47) |

Table 4: Feed-forward comparison on DL3DV under the uncalibrated-camera setting. Runtime is measured on a single NVIDIA H100 GPU. Best results are in bold.

| Method | PSNR ↑ | SSIM ↑ | LPIPS ↓ | #GS (M) ↓ | Context (s) ↓ | Render FPS ↑ |
|---|---|---|---|---|---|---|
| **Low resolution** *2 / 4 / 6 views at* 448×448 | | | | | | |
| AnySplat (Jiang et al., 2025) | 13.85 / 17.33 / 18.40 | 0.416 / 0.503 / 0.529 | 0.451 / 0.345 / 0.315 | 0.34 / 0.66 / 0.95 | **0.14 / 0.18 / 0.23** | 416 / 357 / 302 |
| DA3-GS (Lin et al., 2026) | 18.56 / 21.28 / 22.27 | 0.594 / 0.698 / 0.727 | 0.372 / 0.252 / 0.220 | 0.51 / 1.02 / 1.52 | 0.19 / 0.24 / 0.32 | 323 / 273 / 228 |
| Ours | **20.52 / 23.81 / 24.89** | **0.640 / 0.743 / 0.773** | **0.309 / 0.201 / 0.174** | **0.22 / 0.32 / 0.38** | 0.17 / **0.18 / 0.23** | **507 / 475 / 487** |
| **High resolution** *16 / 32 / 64 views at* 540×960 | | | | | | |
| DA3-GS (Lin et al., 2026) | 19.18 / 19.84 / 20.09 | 0.609 / 0.626 / 0.634 | 0.398 / 0.373 / 0.362 | 2.3 / 4.5 / 9.0 | **0.5** / **1.1** / **2.2** | 176 / 121 / 77 |
| Ours | **20.75 / 21.20 / 21.39** | **0.651 / 0.662 / 0.667** | **0.322 / 0.304 / 0.295** | **1.2** / **1.7** / **2.2** | 0.9 / 2.1 / 5.3 | **283 / 256 / 232** |

### C.3 Qualitative Results

Figs. 9–11 compare novel-view renderings in the low-resolution regime, while Figs. 12–14 show the high-resolution, wide-coverage regime. Each column shows a different scene and target view. Red boxes mark the regions enlarged in the insets, and the per-image PSNR is reported in the lower-left corner. Figs. 15–20 extend the comparison to Mip-NeRF 360, Free, and Tanks & Temples, each evaluated with 16 and 64 context views. The supplementary material also includes a web viewer at `qualitative_viewer/index.html` for side-by-side comparison of fly-through videos from different methods.

Across the short sequences, the voxel-adaptive feed-forward prediction preserves scene structure and fine content more faithfully than the pixel-based and fixed-codebook baselines. Learned-Adam refinement typically sharpens boundaries, repeated patterns, text, and thin structures. The same trend holds as the context expands to 16–64 views: the feed-forward model produces a coherent reconstruction, while refinement recovers additional high-frequency detail from the high-resolution context images. The out-of-distribution comparisons exhibit the same pattern: our feed-forward prediction provides a coherent initialization, and learned Adam recovers sharper local detail across datasets with distinct capture conditions and scene content.




Figure 9: Low-resolution qualitative comparison with two context views at $256\times448$. Rows show the ground truth, MVSplat, DepthSplat, TokenGS, our feed-forward prediction ($E{=}0$), and our result after 16 epochs of learned-Adam refinement ($E{=}16$).




Figure 10: Low-resolution qualitative comparison with four context views at $256\times448$. The row ordering and annotations follow Fig. 9.

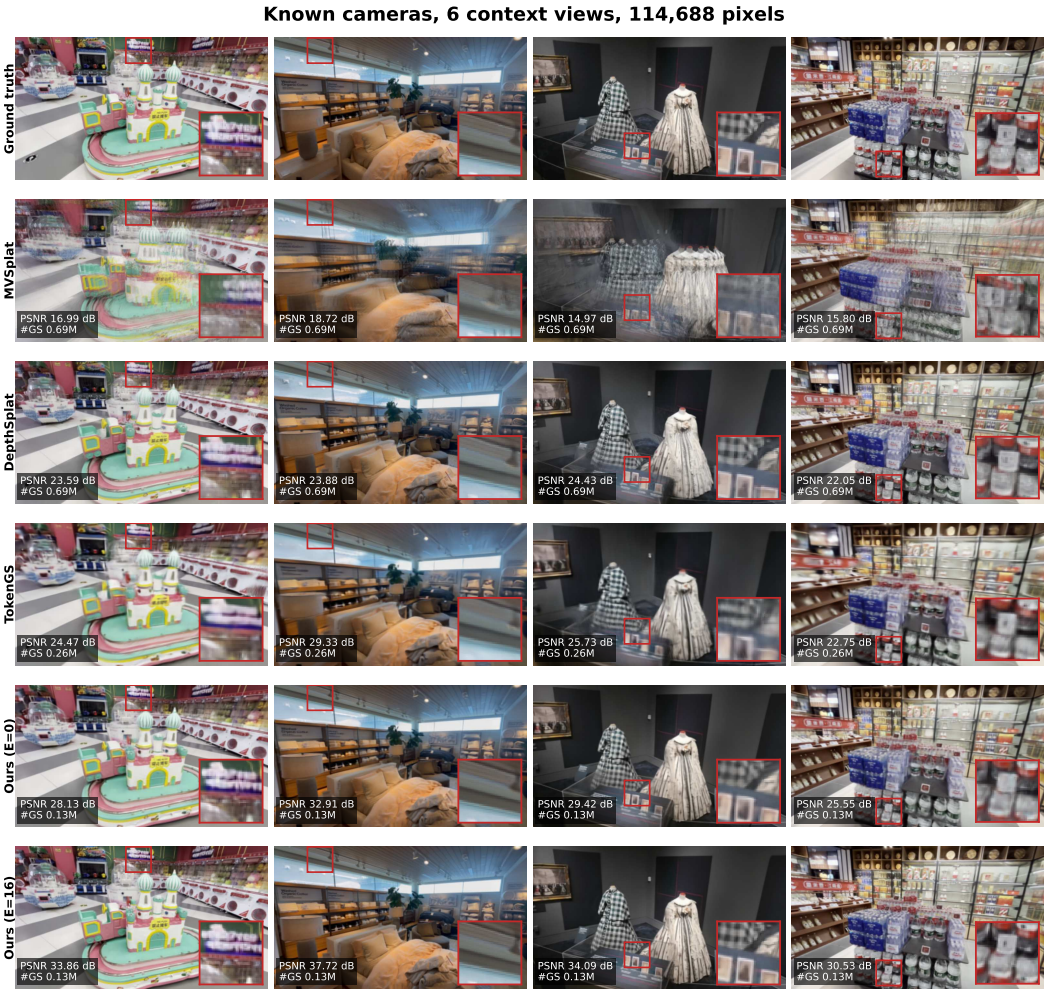


Figure 11: Low-resolution qualitative comparison with six context views at $256\times448$. The row ordering and annotations follow Fig. 9.




Figure 12: High-resolution qualitative comparison with 16 context views at $540\times960$. Rows show the ground truth, Long-LRM, iLRM, tttLRM, MVP, our feed-forward prediction ($E$=0), and our result after 16 epochs of learned-Adam refinement ($E$=16).

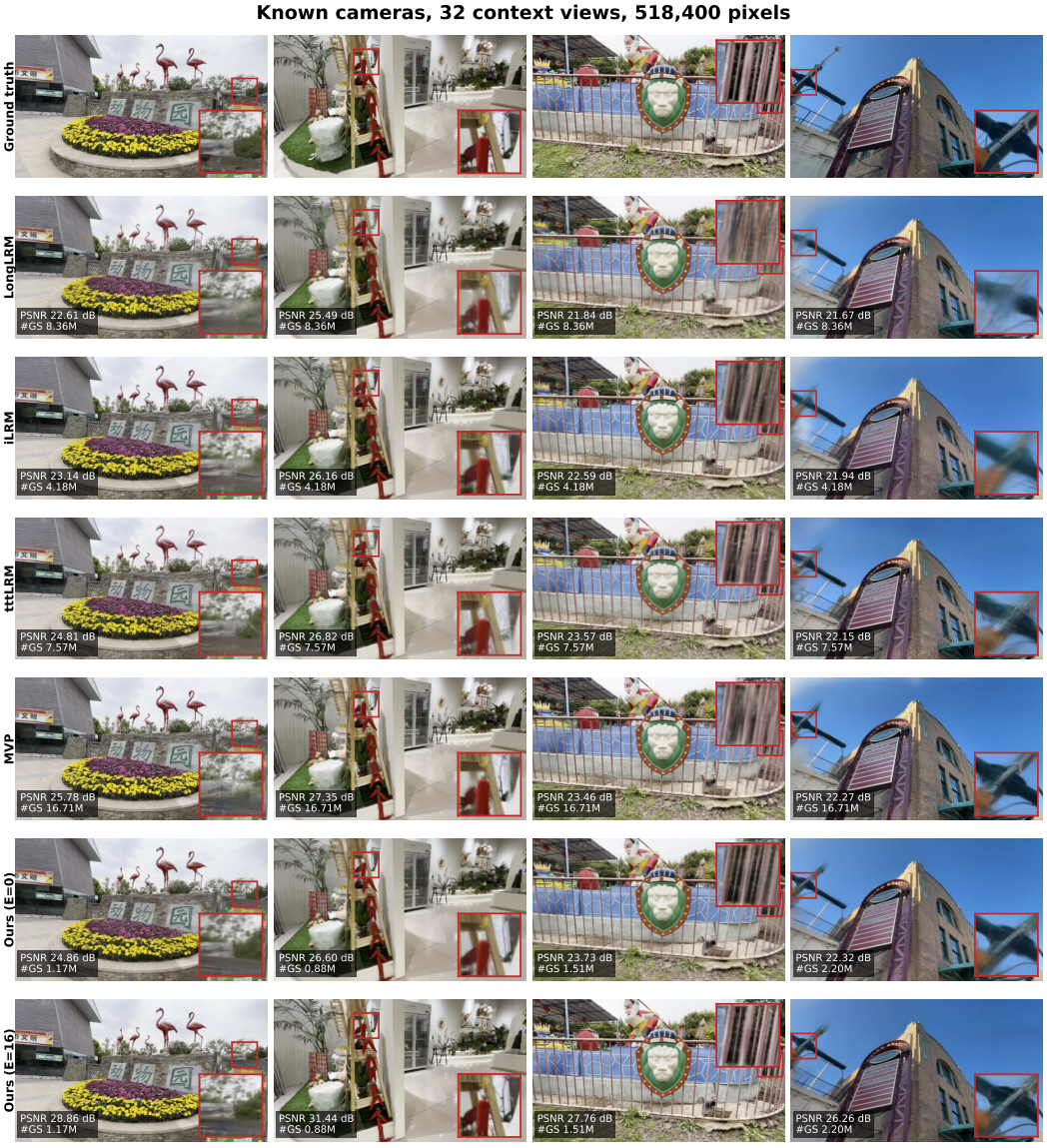


Figure 13: High-resolution qualitative comparison with 32 context views at $540 \times 960$. The row ordering and annotations follow Fig. 12.




Figure 14: High-resolution qualitative comparison with 64 context views at $540\times960$. The row ordering and annotations follow Fig. 12.

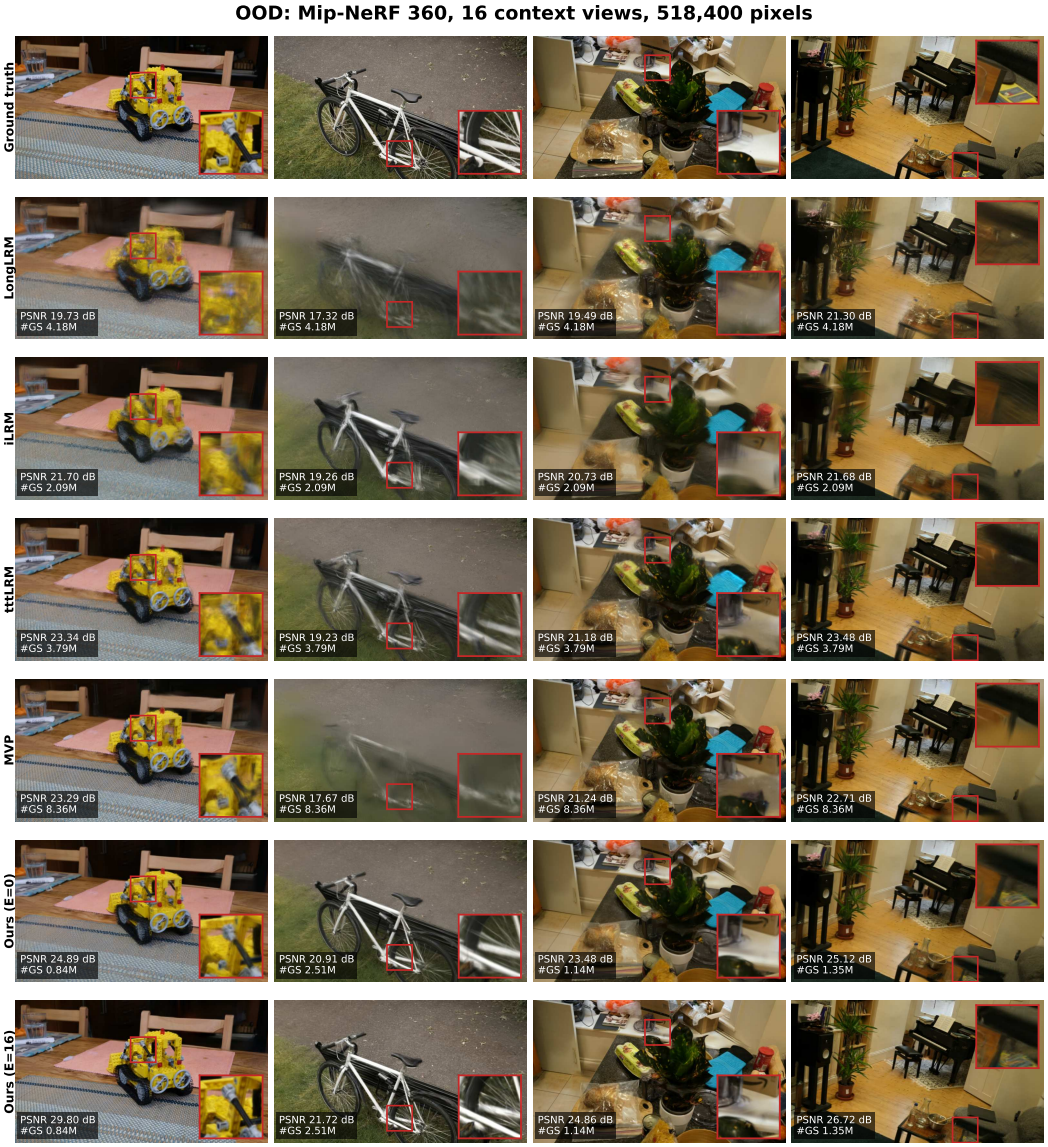


Figure 15: Out-of-distribution qualitative comparison on Mip-NeRF 360 with 16 context views at $540\times960$. Rows show the ground truth, Long-LRM, iLRM, tttLRM, MVP, our feed-forward prediction ($E{=}0$), and our result after 16 epochs of learned-Adam refinement ($E{=}16$).




Figure 16: Out-of-distribution qualitative comparison on Mip-NeRF 360 with 64 context views. The row ordering and evaluation resolution follow Fig. 15.




Figure 17: Out-of-distribution qualitative comparison on Free with 16 context views. The row ordering and evaluation resolution follow Fig. 15.

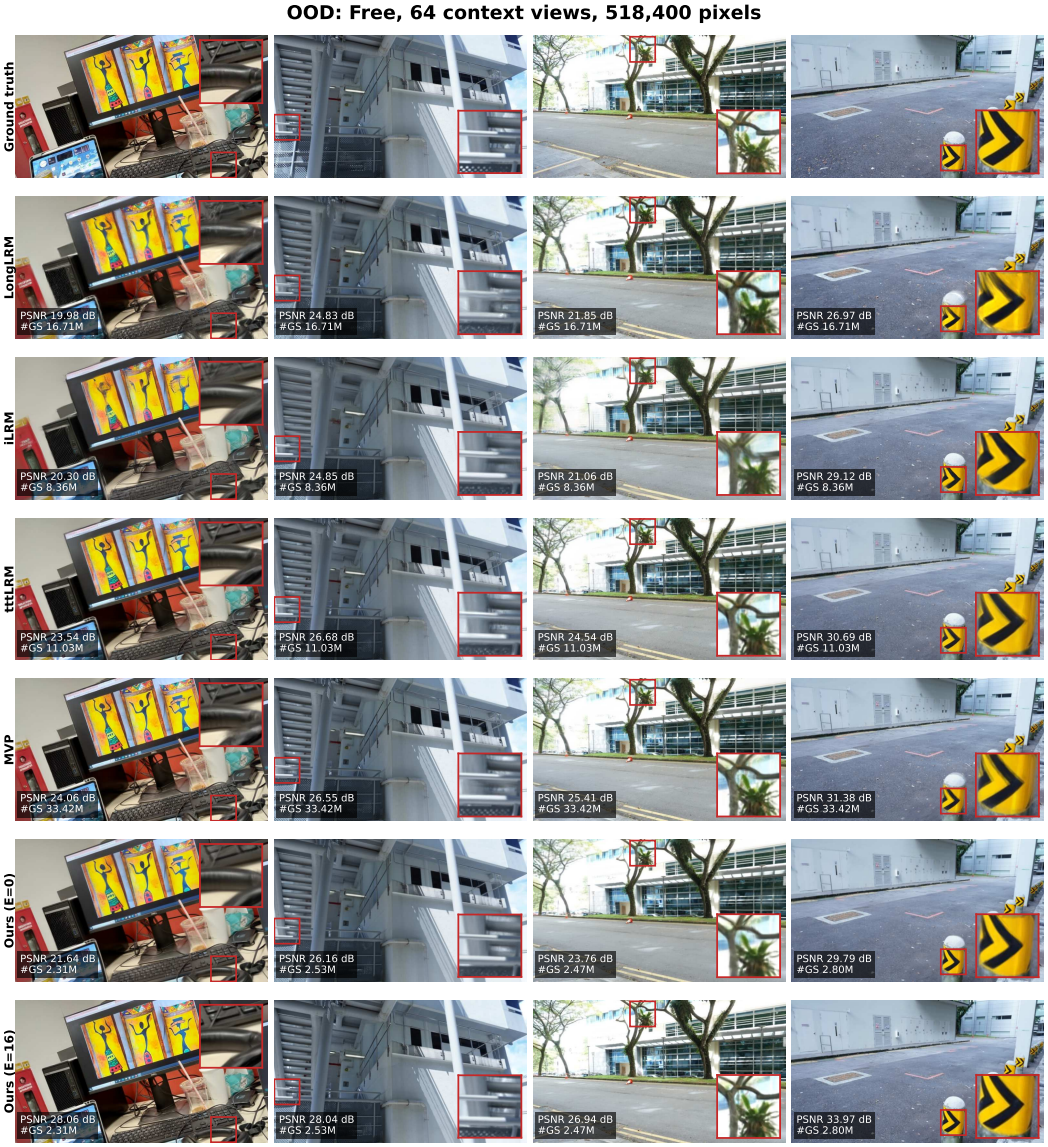


Figure 18: Out-of-distribution qualitative comparison on Free with 64 context views. The row ordering and evaluation resolution follow Fig. 15.




Figure 19: Out-of-distribution qualitative comparison on Tanks & Temples with 16 context views. The row ordering and evaluation resolution follow Fig. 15.

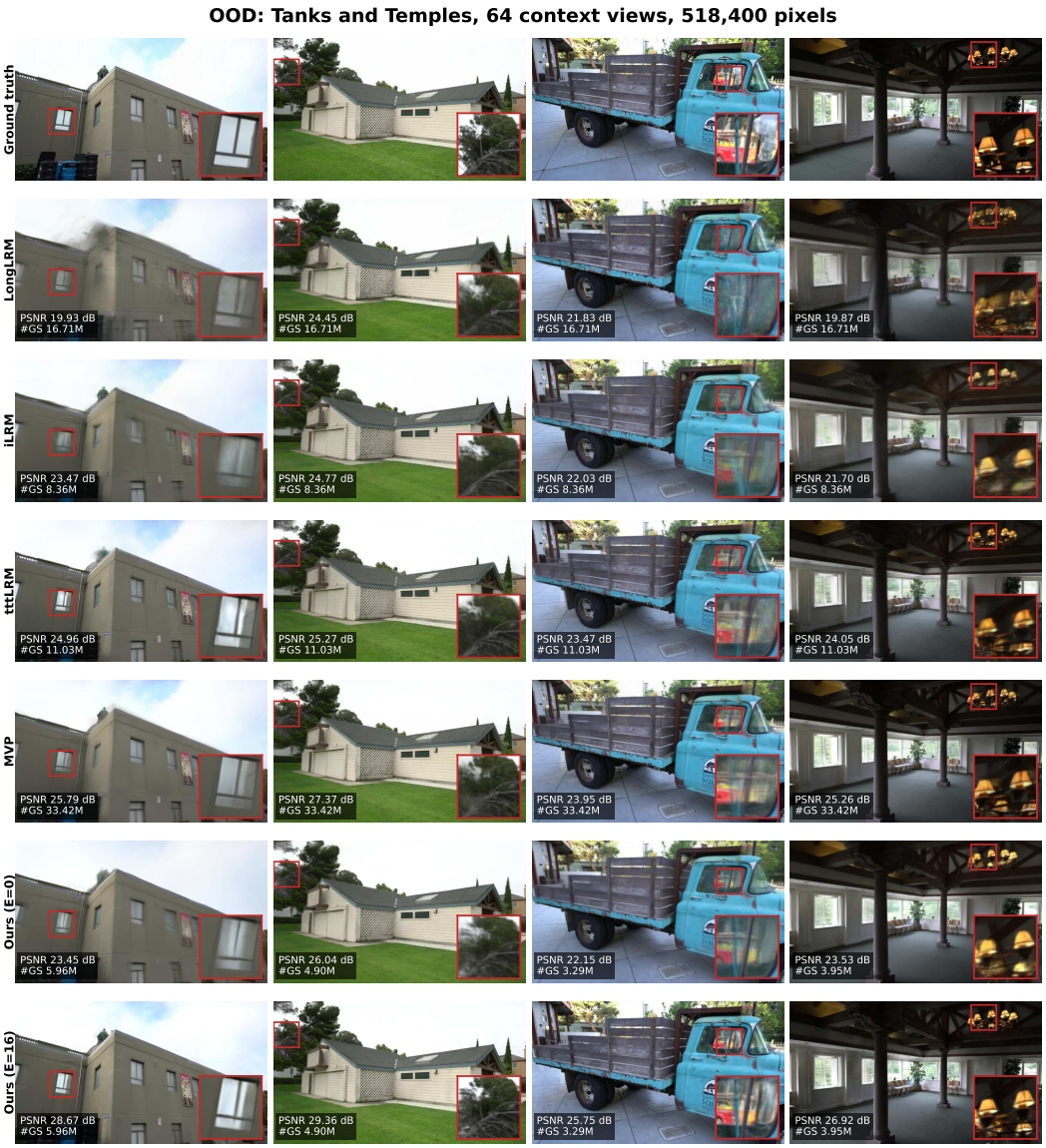


Figure 20: Out-of-distribution qualitative comparison on Tanks & Temples with 64 context views. The row ordering and evaluation resolution follow Fig. 15.