HASHMOD: PLUG-AND-PLAY PATCH-WISE HASH-GUIDED REFINEMENT FOR FEED-FORWARD 3DGS

Anonymous authors
Paper under double-blind review

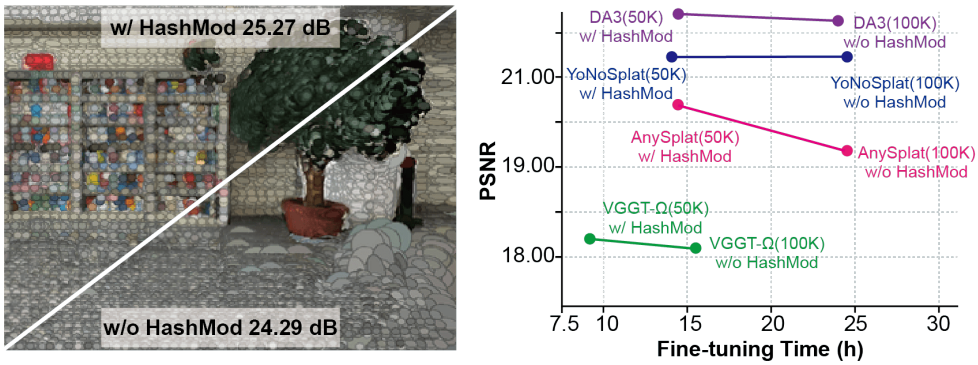


Figure 1: HashMod is a lightweight plug-and-play module for feed-forward 3D Gaussian Splatting models. **Left:** DA3 Gaussian primitives with and without HashMod. **Right:** In the 24-view setting, HashMod maintains or improves PSNR across four models while reducing training time.

ABSTRACT

Feed-forward 3D Gaussian Splatting (3DGS) has emerged as an important approach to efficient and generalizable 3D scene reconstruction, typically decoding image features into Gaussian primitives in a single forward pass. However, reconstruction quality may degrade on unseen scenes with different geometric structures, texture patterns. Fine-tuning only the Gaussian head offers a low-cost adaptation strategy, but its ability to correct local geometric and appearance errors can be limited by insufficient emphasis on relevant backbone features.To address this limitation, we propose HashMod, a lightweight modulation module that uses patch-wise binary query–key matching to adaptively select both the number and identity of backbone feature levels contributing to each local correction. HashMod aggregates the selected feature-derived offsets to multiplicatively refine the initial Gaussian predictions. Across three public benchmarks and cross-dataset settings, HashMod achieves competitive or improved reconstruction quality across AnySplat, YoNoSplat, VGGT-Ω, and Depth Anything 3, while introducing only approximately 8M additional parameters and reducing training time by approximately 41–42%. Furthermore, we conduct fine-tuning experiments on two self-collected crop datasets, each comprising 1,000 videos. The adapted feed-forward 3DGS models improve reconstruction quality on real-world crop scenes, demonstrating the practical applicability of HashMod beyond standard benchmarks.

1 INTRODUCTION

3D Gaussian Splatting (3DGS) represents scenes with anisotropic Gaussian primitives and achieves photorealistic novel-view synthesis through efficient rendering (Kerbl et al. (2023)). However, conventional 3DGS requires iterative optimization process, which limits its use in scenarios demanding real-time results. Feed-forward 3DGS addresses this limitation by learning to predict Gaussian parameters directly from input images (Charatan et al. (2024); Chen et al. (2024)). Recent models combine pretrained reconstruction backbones, which extract geometric and appearance features,

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.

with Gaussian heads that decode these features into scene representations (Xu et al. (2025); Jiang et al. (2025); Lin et al. (2025)). These models reconstruct unseen scenes in a single forward pass without per-scene optimization. Nevertheless, reconstruction quality can degrade when target scenes differ from the training data in geometry, texture, or occlusion patterns (Mo et al. (2025)).

Adapting pretrained models to target datasets therefore remains important. Fine-tuning the entire model is computationally expensive, whereas updating only the Gaussian head offers a lower-cost alternative that keeps the pretrained backbone fixed (Lin et al. (2025); Smart et al. (2024)). However, backbone features extracted from target scenes may differ in distribution from those encountered by the Gaussian head during pretraining. Head-only fine-tuning must therefore adapt the mapping from these features to Gaussian parameters. This adaptation can be challenging because feature shifts and reconstruction errors vary spatially, and their correction may require different combinations of geometric and appearance cues.

To address this challenge, we propose HashMod, a lightweight plug-and-play module for spatially adaptive Gaussian refinement. Its design is motivated by the differing feature requirements of regions containing thin structures, repeated textures, or occlusion boundaries. Such regions may benefit from different combinations of fine detail and broader geometric context. Local refinement should therefore determine both which feature levels contribute and how many are needed. Aggregating all levels may introduce irrelevant information, whereas selecting a fixed subset may omit useful cues. HashMod uses patch-wise binary query–key matching to select a variable number of backbone feature levels for each local correction. Level-specific branches predict candidate offsets, which are selectively aggregated through the resulting masks to multiplicatively refine the initial Gaussian parameters. This single-step refinement preserves the original pretrained prediction pathway while making local feature selection and correction explicit. Our main contributions are:

(1) **Lightweight hash-guided modulation.** We introduce HashMod, which adaptively selects both the number and identity of backbone feature levels used for local Gaussian refinement. It adds approximately 8M parameters without modifying the original backbone or Gaussian head architecture.

(2) **Reconstruction quality and fine-tuning efficiency.** Experiments on AnySplat, YoNoSplat, VGGT-Ω, and Depth Anything 3 across multiple public benchmarks demonstrate the effectiveness of HashMod. It achieves competitive or improved performance with half as many fine-tuning iterations and provides further gains for most architectures under the same iteration budget.

(3) **Explicit and inspectable refinement.** Spatial selection masks and level-specific offsets expose which backbone feature levels contribute to each local correction. These components support visualization and quantitative analysis of feature selection and its effects on reconstruction quality.

(4) **Real applications on crop datasets.** On two self-collected crop datasets containing 1,000 scenes each, HashMod improves DA3 by 0.93 and 0.40 dB on the millet and maize test scenes, respectively, demonstrating its applicability to real-world crop reconstruction.

2 Related Work

Per-scene optimized Gaussian Splatting. 3DGS represents a scene as a collection of learnable anisotropic Gaussian primitives and optimizes their geometric and appearance attributes through differentiable splatting for novel-view synthesis (Kerbl et al. (2023)). Building upon this representation, subsequent studies have improved computational efficiency through compact Gaussian representations, adaptive densification and pruning (Ren et al. (2026); Hanson et al. (2025)), and accelerated rendering pipelines (Huang et al. (2025); Du et al. (2026)). In parallel, reconstruction fidelity has been advanced through frequency-aware representation decomposition (Lavi et al. (2026)), geometrically exact Gaussian rendering (Huang et al. (2026)), and anti-aliasing techniques (Li et al. (2025)). Despite these advances, such methods still require iterative optimization for every individual scene and cannot directly generalize their learned representations to unseen inputs.

Feed-forward Gaussian Splatting. In contrast, recent studies have extended 3DGS toward feed-forward reconstruction by directly predicting Gaussian representations from input images. Whether developed as dedicated feed-forward 3DGS models, such as DepthSplat (Xu et al. (2025)), AnySplat (Jiang et al. (2025)), and YoNoSplat (Ye et al. (2026)), or constructed by equipping large-scale 3D reconstruction models, such as DA3 (Lin et al. (2025)) and VGGT-Ω (Wang et al. (2026)),

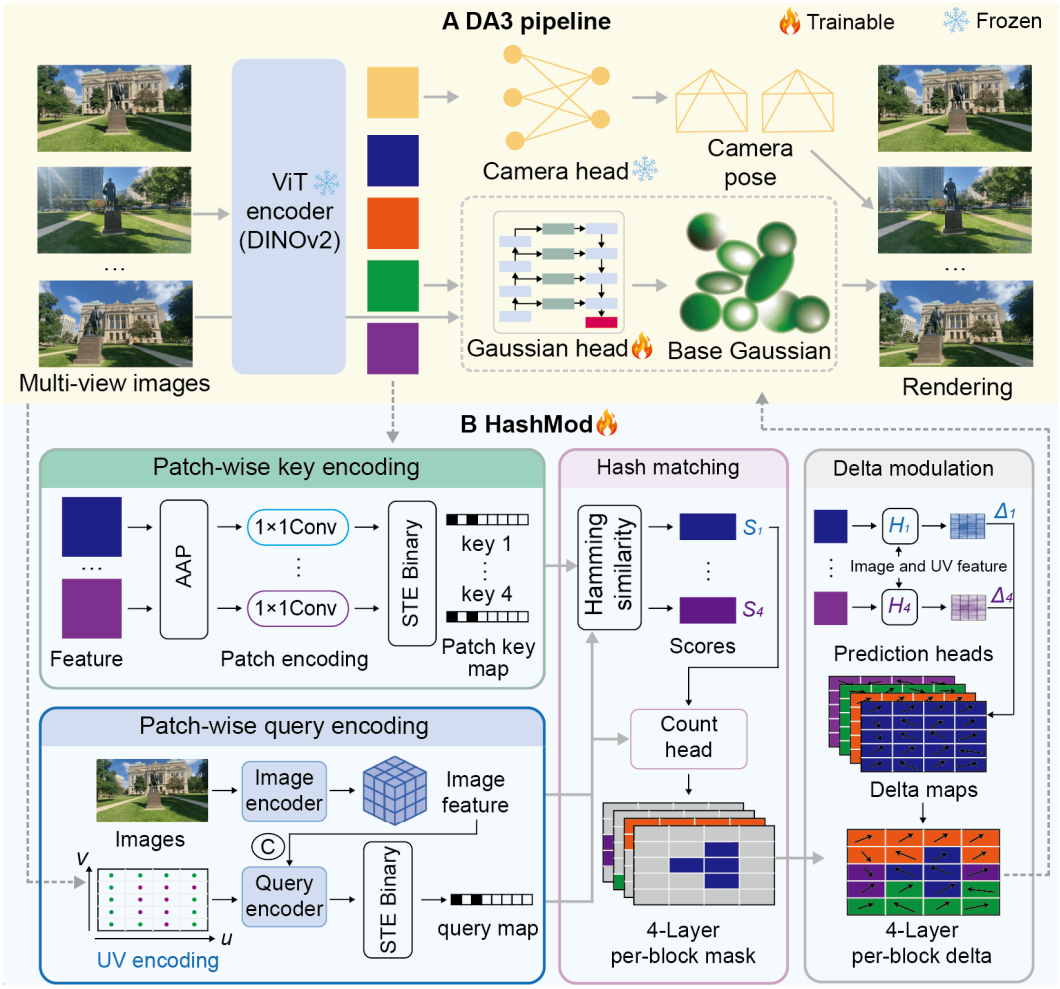


Figure 2: **Overview of the proposed HashMod framework.** (A) The pretrained DA3 backbone extracts multi-level features, while the original camera and Gaussian heads predict camera poses and the base Gaussian. (B) HashMod encodes the selected backbone features and image into patch-wise binary keys and queries. Hash matching produces a patch-wise feature-level selection mask. Each backbone-level feature is combined with the image and UV features to predict a delta correction, and the resulting offsets are fused patch by patch according to the selection mask to modulate the base Gaussian prediction.

with Gaussian prediction heads, these approaches can generally be abstracted into image feature extraction, cross-view interaction, and Gaussian parameter decoding. Within this pipeline, powerful pretrained backbones provide rich hierarchical representations, whereas Gaussian heads commonly rely on predefined feature levels and fixed fusion operations for one-shot prediction, potentially underutilizing scale-specific information (Xing et al. (2026)). To address this limitation, HashMod serves as a lightweight, plug-and-play module that selectively combines local corrections from different feature levels to refine Gaussian predictions in a single modulation step.

3 METHOD

Section 3.1 introduces the main components of feed-forward 3DGS architectures. Section 3.2 details the integration of HashMod into DA3 (Fig. 2), and the integration to other feed-forward 3DGS architectures are illustrated in Appendix Figure A1. Section 3.3 describes the fine-tuning procedure.

3.1 PRELIMINARIES

Feed-forward backbone. Given N input views of a scene, $\mathcal{I} = \{I_i\}_{i=1}^{N}$, the pretrained backbone $\mathcal{F}$ extracts hierarchical feature:

$$\mathcal{Z} = \mathcal{F}(\mathcal{I}), \tag{1}$$

where $\mathcal{Z}$ denotes the collection of features extracted at different backbone depths. Among these, $\mathcal{Z}_{\ell_s}$ denotes the feature from selected layer ℓ_s, with $s = 1, \ldots, S$ and S denoting the number of selected feature levels. The selected layer indices $\{\ell_s\}_{s=1}^{S}$ may vary across base architectures. Table A1 in the appendix summarizes the backbones used by different models. In our pose-adaptive setting, the reconstruction model estimates camera parameters internally from the input views.

Feed-forward Gaussian head. The Gaussian prediction head $\mathcal{H}_{\mathrm{GS}}$ processes the selected backbone features $\{\mathcal{Z}_{\ell_s}\}_{s=1}^{S}$ and predicts the base Gaussian representation:

$$\Theta_{\mathrm{base}} = \mathcal{H}_{\mathrm{GS}}\left(\{\mathcal{Z}_{\ell_s}\}_{s=1}^{S}\right) = \{\mathcal{G}_k\}_{k=1}^{K}, \tag{2}$$

where K denotes the number of predicted Gaussian primitives. Each primitive is parameterized as $\mathcal{G}_k = (\boldsymbol{\mu}_k, \mathbf{q}_k, \mathbf{s}_k, \alpha_k, \mathbf{c}_k)$, representing its center, rotation, scale, opacity, and color, respectively. Depending on the base architecture, the Gaussian head projects, aligns, and fuses the selected features before decoding the Gaussian parameters. The Gaussian head designs used by different base models are summarized in Appendix Table A1. The resulting Θ_{base} serves as the base Gaussian prediction for subsequent refinement by HashMod.

3.2 INTEGRATING HASHMOD INTO DA3

DA3 pipeline. Given multi-view images $\mathcal{I}$, DA3 uses a DINOv2-based ViT encoder to extract features from four selected Transformer layers, denoted by $\{\mathcal{Z}_{\ell_s}\}_{s=1}^{4}$. The DPT-style Gaussian head $\mathcal{H}_{\mathrm{GS}}$ converts these features into spatial feature maps, aligns their dimensions and resolutions, and progressively fuses them from the deepest to the shallowest level. Incorporating RGB features from the input images, the head decodes the fused representation into the base Gaussian parameters:

$$\Theta_{\mathrm{base}} = \mathcal{H}_{\mathrm{GS}}\left(\{\mathcal{Z}_{\ell_s}\}_{s=1}^{4}, \mathcal{I}\right). \tag{3}$$

The resulting Θ_{base} serves as the initial Gaussian representation for subsequent modulation by HashMod. The camera estimation branch remains unchanged, and its predicted camera poses are used for rendering without modification.

Patch-wise key encoding. The key encoder constructs a patch-wise binary key map for each selected feature level k_{ℓ_s}. Specifically, each feature $\mathcal{Z}_{\ell_s}$ is mapped to the shared patch grid using adaptive average pooling (AAP), followed by a level-specific 1×1 projection and STE-based binarization (Su et al. (2018)):

$$k_{\ell_s} = \mathcal{B}\left(\mathcal{P}_{\ell_s}\left(\mathrm{AAP}(\mathcal{Z}_{\ell_s})\right)\right), \qquad s = 1, \ldots, 4. \tag{4}$$

Here, $\mathcal{P}_{\ell_s}$ denotes the level-specific projection, and $\mathcal{B}$ produces a b-bit code in $\{-1, +1\}^b$ at each routing location using straight-through estimation.

Patch-wise query encoding. The query encoder constructs a patch-wise query map from image and UV features for matching across all feature levels q. Let g denote the image feature extracted by the image encoder from the input views $\mathcal{I}$. We pool g to the patch grid and concatenate it with the normalized two-channel UV map p to obtain

$$q = \mathcal{B}\left(\mathcal{Q}\left(\mathrm{Concat}\left[\mathrm{AAP}(g), p\right]\right)\right). \tag{5}$$

Here, $\mathcal{Q}$ comprises two 1×1 convolutions with an intermediate GELU activation.

Adaptive hash matching. We compare the binary query q with each feature-level key k_{ℓ_s} at corresponding patches. Their matching score is computed as

$$\mathrm{score}_{\ell_s} = \frac{1}{b} q^{\top} k_{\ell_s}, \qquad s = 1, \ldots, 4, \tag{6}$$

The resulting score measures the local compatibility between the image query and the corresponding feature level, with a higher value indicating a stronger match.

For each patch, a two-layer MLP takes the concatenated similarity scores, RGB features, and UV coordinates to predict the number of selected levels $n \in \{1, \ldots, 4\}$. The selector assigns binary masks $\{M_s\}_{s=1}^{4}$, setting $M_s = 1$ for the top-n levels ranked by similarity and $M_s = 0$ otherwise. Selection is trained end-to-end using straight-through estimation.

Hash-guided delta modulation. HashMod predicts a candidate Gaussian correction at each pyramid level. For level ℓ_s, the shared RGB feature is bilinearly downsampled to the resolution of $\mathcal{Z}_{\ell_s}$ and concatenated with this feature and the normalized UV coordinates. A level-specific refinement head then predicts the offset:

$$\delta_{\ell_s} = \mathcal{H}^{\Delta}_{\ell_s}\left(\mathrm{Concat}\left[\mathcal{Z}_{\ell_s}, g_{\ell_s}, p_{\ell_s}\right]\right), \qquad s = 1, \ldots, 4, \tag{7}$$

where g_{ℓ_s} is the aligned RGB feature, p_{ℓ_s} is the aligned UV feature, and δ_{ℓ_s} is predicted at the native resolution of level ℓ_s to correct the Gaussian center, rotation, scale, and color attributes $(\boldsymbol{\mu}_k, \mathbf{q}_k, \mathbf{s}_k, \mathbf{c}_k)$, while leaving opacity unchanged. The refinement heads share the same lightweight residual architecture but have independent parameters, each comprising an input projection, residual blocks, and a linear output projection.

The patch-wise mask M controls the local contribution of each level-specific offset. Before aggregation, each mask channel M_s and its corresponding offset δ_{ℓ_s} are spatially aligned to the finest pyramid resolution; for simplicity, we retain their original notation after alignment. The selected offsets are aggregated as

$$\Delta_f = \sum_{s=1}^{4} M_s \odot \delta_{\ell_s}. \tag{8}$$

The aggregated offset Δ_f is then aligned to the resolution of the base Gaussian output and denoted by $\Delta\Theta$. The final Gaussian prediction is obtained through multiplicative modulation:

$$\Theta_{\mathrm{final}} = \Theta_{\mathrm{base}} \odot (1 + \Delta\Theta). \tag{9}$$

3.3 Model Training

Fine-tuning strategy. During fine-tuning, we keep the pretrained backbone $\mathcal{F}$ frozen and jointly optimize the original Gaussian head and HashMod for novel-view synthesis. HashMod is initialized from scratch, while the output projection of each offset refinement head is initialized to 0. Consequently, HashMod initially produces zero corrections and preserves the base Gaussian prediction. The model is then optimized using the view-synthesis objective described below.

View-synthesis objective. Given an input view set $\mathcal{I}$, the model predicts the final Gaussian representation Θ_{final} and camera parameters cam. The target views $\widehat{\mathcal{I}}$ are rendered using the differentiable Gaussian splatting operator $\mathcal{R}$:

$$\widehat{\mathcal{I}} = \mathcal{R}\left(\Theta_{\mathrm{final}}, \mathrm{cam}\right). \tag{10}$$

The training objective combines image reconstruction and gradient consistency:

$$\mathcal{L} = \mathrm{SmoothL1}_{\beta=0.05}\left(\widehat{\mathcal{I}}, \mathcal{I}\right) + 2\left\|\nabla\overline{\widehat{\mathcal{I}}} - \nabla\overline{\mathcal{I}}\right\|_1, \tag{11}$$

where $\overline{\mathcal{I}}$ denotes image smoothing with a 3×3 average-pooling operator, and ∇ computes finite differences along the two spatial directions.

4 Experiment

Datasets. For public-benchmark experiments, all models are fine-tuned exclusively on the DL3DV-10K training split Ling et al. (2024), excluding the DL3DV-Benchmark scenes. Following the DA3 evaluation protocol, we evaluate on the official 140-scene DL3DV-Benchmark, all Tanks and Temples training scenes except *Courthouse* Knapitsch et al. (2017), and Mip-NeRF 360 Barron et al. (2022). For crop-domain adaptation, we additionally fine-tune and evaluate the models on our self-collected datasets containing 1,000 millet scenes and 1,000 maize scenes, each split 8:2 for training and testing. Camera poses are estimated internally by the models from unposed input images.

Table 1: Quantitative comparison of different methods with and without HashMod across varying numbers of input views and fine-tuning budgets. For each method and view setting, the best and second-best results for each metric are highlighted in light yellow and light red, respectively.

Method	Original (100K)			HashMod (50K)			HashMod (100K)		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
16 Views									
AnySplat	18.76	0.609	0.366	19.00	0.620	0.359	19.12	0.624	0.356
YoNoSplat	20.61	0.640	0.305	20.57	0.637	0.304	20.58	0.637	0.306
VGGT-Ω	17.55	0.547	0.472	17.69	0.553	0.467	17.81	0.557	0.454
DA3	21.33	0.726	0.303	21.31	0.726	0.305	21.53	0.733	0.300
24 Views									
AnySplat	19.44	0.636	0.358	19.72	0.648	0.349	19.82	0.652	0.347
YoNoSplat	21.24	0.675	0.286	21.24	0.674	0.286	21.20	0.672	0.288
VGGT-Ω	18.10	0.569	0.467	18.20	0.573	0.463	18.37	0.579	0.447
DA3	21.62	0.738	0.292	21.64	0.739	0.296	21.86	0.745	0.290
32 Views									
AnySplat	19.77	0.650	0.354	20.07	0.664	0.344	20.17	0.667	0.342
YoNoSplat	21.27	0.681	0.284	21.32	0.682	0.283	21.22	0.678	0.287
VGGT-Ω	18.29	0.578	0.466	18.38	0.582	0.463	18.57	0.588	0.446
DA3	21.66	0.742	0.288	21.70	0.744	0.292	21.90	0.749	0.286

Table 2: Cross-dataset evaluation with 24 views after 100K iterations on DL3DV. Best results are highlighted in yellow.

Method	Original (100K)			HashMod (100K)		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
DL3DV → Tanks and Temples						
AnySplat	15.16	0.510	0.478	15.41	0.520	0.473
YoNoSplat	15.19	0.447	0.512	15.97	0.471	0.465
VGGT-Ω	15.06	0.452	0.543	15.22	0.463	0.531
DA3	17.65	0.601	0.424	17.71	0.601	0.428
DL3DV → Mip-NeRF 360						
AnySplat	17.97	0.559	0.426	18.19	0.565	0.421
YoNoSplat	19.54	0.494	0.438	20.21	0.516	0.384
VGGT-Ω	17.80	0.479	0.522	17.86	0.481	0.510
DA3	20.98	0.601	0.410	21.04	0.603	0.414

Table 3: Fine-tuning efficiency of 100K-iteration baselines and 50K-iteration HashMod variants. Params. and Time report total parameters (M) and training hours.

Method	PSNR↑	Params. (M)↓	Time (h)↓
AnySplat (100K)	19.44	1190.736	∼24.4
w/ HashMod (50K)	19.72	1199.386	∼14.5
YoNoSplat (100K)	21.24	964.974	∼24.5
w/ HashMod (50K)	21.24	966.498	∼14.2
VGGT-Ω (100K)	18.10	1176.809	∼15.6
w/ HashMod (50K)	18.20	1185.436	∼9.1
DA3 (100K)	21.62	1355.674	∼24.9
w/ HashMod (50K)	21.70	1364.394	∼14.5

Implementation details. We initialize the backbone and Gaussian head from official pretrained weights where available. When pretrained Gaussian-head weights are unavailable, as for VGGT-Ω, we attach a DA3-style Gaussian head and pretrain it on DL3DV for 50K iterations. We use 8-bit binary codes for both key and query encoding. Optimization uses AdamW with a cosine learning-rate schedule and learning rates of 1×10^{-5} for the Gaussian head and 1×10^{-4} for HashMod. All models are implemented in PyTorch and trained with mixed precision, FlashAttention, and a batch size of 1. For each architecture, the baseline and HashMod share the same base-model checkpoint, training data, objectives, and optimization settings for their common components. Both variants are fine-tuned for 100K iterations for comparison under equal iteration budgets. We additionally evaluate HashMod at 50K iterations to assess reconstruction quality with half the fine-tuning iterations. All experiments run on a single NVIDIA A5000 Pro GPU, with reconstruction quality measured using PSNR, SSIM, and LPIPS.



Figure 3: Qualitative comparison of four feed-forward 3DGS models with and without HashMod under the multi-view setting. Red boxes indicate the regions selected for detailed comparison.

4.1 Comparisons

We evaluate HashMod on AnySplat, YoNoSplat, VGGT-Ω, and DA3 across different input view counts and fine-tuning budgets. Cross-dataset generalization is evaluated by training on DL3DV and testing on Tanks and Temples and Mip-NeRF 360 with 24 input views.

Comparisons with varying input views. Table 1 reports results with 16, 24, and 32 input views. At 50K iterations, HashMod improves AnySplat and VGGT-Ω while remaining competitive on YoNoSplat and DA3 against their 100K baselines. At 100K iterations, it further improves AnySplat, VGGT-Ω, and DA3 across all view settings, whereas YoNoSplat performs best at 50K, suggesting earlier convergence. Overall, HashMod improves fine-tuning efficiency and reconstruction quality, although the optimal training budget varies by architecture. Figure 3 further shows improved thin structures, repetitive patterns, occluded regions, and complex textures.

Cross-dataset generalization. Table 2 evaluates the generalization of DL3DV-trained models to Tanks and Temples and Mip-NeRF 360 using 24 input views. HashMod consistently improves AnySplat, YoNoSplat, and VGGT-Ω across all metrics on both datasets, with the largest gains observed for YoNoSplat. DA3 also achieves higher PSNR, while SSIM remains similar or improves slightly and LPIPS slightly worsens. These results demonstrate that HashMod transfers effectively to unseen datasets across different reconstruction frameworks.

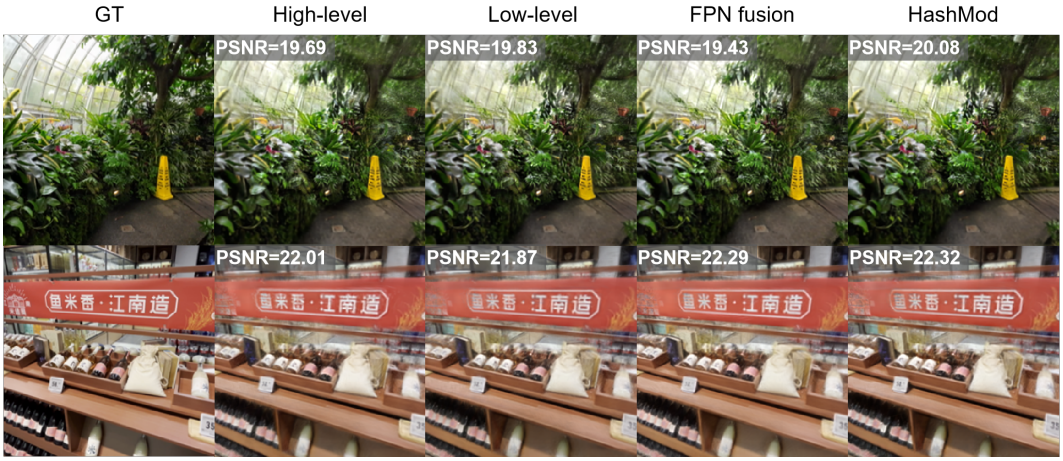


Figure 4: **Visual ablation study.** Predicting offsets using only high-level or low-level features yields insufficient detail recovery. FPN fusion improves reconstruction in some regions, while HashMod produces the most accurate results with clearer structures and finer details.

Comparison of parameters and training efficiency. Table 3 compares the 50K-iteration HashMod variants with their corresponding baselines fine-tuned for 100K iterations. Despite using only half the iteration budget, HashMod achieves higher PSNR for AnySplat, VGGT-Ω, and DA3, while matching YoNoSplat. This reduces the measured fine-tuning time by approximately 41–42%. The additional parameter count ranges from 1.5M to 8.7M and remains below 0.74% of each base model. YoNoSplat incurs the smallest overhead because HashMod operates on only three selected feature levels in this architecture.

4.2 Ablation Study

We investigate four feature configurations using 16 input views: modulation from the highest-level features only, modulation from the lowest-level features only, multi-level FPN fusion, and the proposed HashMod. We further assess the contribution of the learned corrections by enabling and disabling modulation at inference using the same jointly trained checkpoint across different input view counts.

Feature-level selection. Table 4 compares feature selection and fusion strategies using 16 input views. Low-level features slightly outperform high-level features, while FPN fusion further improves all three metrics, suggesting complementary information across levels. HashMod achieves the highest PSNR and SSIM with comparable LPIPS. The qualitative comparisons in Fig. 4 further show better preservation of fine textures and object boundaries, supporting the effectiveness of adaptive feature selection for Gaussian refinement.

Effect of modulation at inference. We compare inference with and without modulation using the same jointly trained checkpoint. Disabling modulation retains the coarse Gaussian prediction, while

Table 4: Ablation of feature selection and fusion strategies using 16 input views. The gray row denotes the full HashMod model, and the best result for each metric is shown in bold.

High-level features	Low-level features	FPN fusion	HashMod	PSNR↑	SSIM↑	LPIPS↓
✓				21.41	0.730	0.301
	✓			21.44	0.731	0.300
✓	✓	✓		21.47	0.732	**0.299**
✓	✓		✓	**21.53**	**0.733**	0.300

Table 5: Effect of disabling or enabling modulation at inference using the same jointly trained checkpoint. Best results are bold.

Views	Modulation Disabled			Modulation Enabled		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
16	20.69	0.716	0.335	**21.53**	**0.733**	**0.300**
24	20.87	0.723	0.329	**21.86**	**0.745**	**0.290**
32	20.86	0.725	0.328	**21.90**	**0.749**	**0.286**

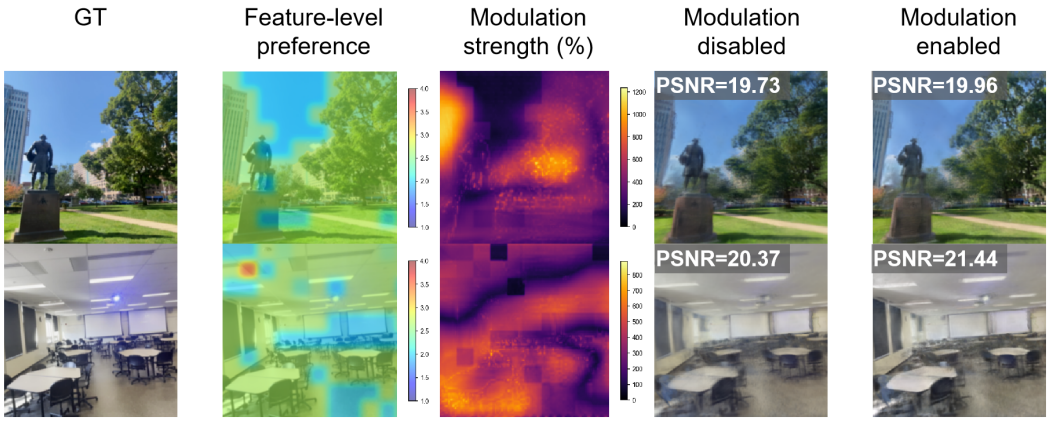


Figure 5: Qualitative visualization of the proposed modulation module. From left to right, the columns show the ground-truth, the module's preference for shallow or deep features, the magnitude of its Gaussian-parameter modulation, and the rendered results without and with modulation. In the feature-preference map, larger values indicate a stronger preference for deeper features. In the modulation-strength map, brighter regions receive larger parameter adjustments.

Table 6: Reconstruction results on the self-collected millet and maize datasets.

Dataset	DA3			DA3 w/ HashMod		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Millet	22.48	0.844	0.341	**23.41**	**0.851**	**0.327**
Maize	22.07	0.750	0.373	**22.47**	**0.761**	**0.356**

enabling it applies the learned corrections and consistently improves all three metrics across view counts (Table 5). Fig. 5 further shows that HashMod selects feature levels according to local content, with larger corrections concentrated around detailed textures and geometric boundaries, producing clearer contours and local structures.

4.3 Application to real-world crop reconstruction

As reported in Table 6, HashMod consistently improves DA3 across all metrics on both self-collected crop datasets. On millet, it increases PSNR by 0.93 dB and SSIM by 0.007 while reducing LPIPS by 0.014. On maize, it improves PSNR by 0.40 dB and SSIM by 0.011 while reducing LPIPS by 0.017. The qualitative results in Fig. A2 show an overall improvement in the millet reconstruction and clearer local details in the red-boxed regions of the maize scenes. These results demonstrate the applicability of HashMod to challenging real-world crop reconstruction.

5 Conclusion

This paper presents HashMod, a lightweight modulation framework for feed-forward 3DGS. We demonstrate that, without modifying or retraining the pretrained backbone, introducing an approximately 8M-parameter module is sufficient to improve Gaussian prediction across diverse architectures. HashMod adaptively selects multi-level backbone features according to local correction demands and uses their aggregated offsets to refine the base Gaussian representation, enabling more effective use of pretrained features during fine-tuning. Extensive experiments show consistent improvements across multiple models and public benchmarks, with 41–42% less training time to reach comparable performance. Further gains on self-collected millet and maize datasets demonstrate its applicability to real-world crop reconstruction.

AI USE STATEMENT

We did not use generative AI tools for any tasks requiring mandatory disclosure. Generative AI was used solely to improve the grammar, fluency, and readability of the manuscript, without altering its technical content. All AI-assisted edits were reviewed by the authors. We take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide the complete formulation of HashMod and its training objective in Sec. 3, together with the dataset splits, evaluation protocols, and implementation details in Sec. 4. Appendix A.2 describes how HashMod is integrated into each evaluated model architecture.

REFERENCES

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5460–5469. IEEE, 2022.

David Charatan, Sizhe Lester Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelsplat: 3d gaussian splats from image pairs for scalable generalizable 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19457–19467. IEEE, 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 *European conference on computer vision (ECCV)*, pp. 370–386. Springer, 2024.

Xiaobiao Du, Yida Wang, Kun Zhan, and Xin Yu. Mobile-gs: Real-time gaussian splatting for mobile devices. *arXiv preprint arXiv:2603.11531*, 2026.

Alex Hanson, Allen Tu, Geng Lin, Vasu Singla, Matthias Zwicker, and Tom Goldstein. Speedy-splat: Fast 3d gaussian splatting with sparse pixels and sparse primitives. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21537–21546. IEEE, 2025.

Xiaotong Huang, He Zhu, Zihan Liu, Weikai Lin, Xiaohong Liu, Zhezhi He, Jingwen Leng, Minyi Guo, and Yu Feng. Seele: A unified acceleration framework for real-time gaussian splatting. *arXiv preprint arXiv:2503.05168*, 2025.

Zixun Huang, Cho-Ying Wu, Yuliang Guo, Xinyu Huang, and Liu Ren. 3dgeer: 3d gaussian rendering made exact and efficient for generic cameras. In *International Conference on Learning Representations (ICLR)*, 2026.

Lihan Jiang, Yucheng Mao, Linning Xu, Tao Lu, Kerui Ren, Yichen Jin, Xudong Xu, Mulin Yu, Jiangmiao Pang, Feng Zhao, et al. Anysplat: Feed-forward 3d gaussian splatting from unconstrained views. *ACM Transactions on Graphics (TOG)*, 44(6):1–16, 2025.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, George Drettakis, et al. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42(4):139–1, 2023.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

Yishai Lavi, Leo Segre, and Shai Avidan. Frequency-aware gaussian splatting decomposition. In *2026 International Conference on 3D Vision (3DV)*, pp. 1133–1142. IEEE, 2026.

Jiameng Li, Yue Shi, Jiezhang Cao, Bingbing Ni, Wenjun Zhang, Kai Zhang, and Luc Van Gool. Mipmap-gs: Let gaussians deform with scale-specific mipmap for anti-aliasing rendering. In *2025 International Conference on 3D Vision (3DV)*, pp. 935–945. IEEE, 2025.

Haotong Lin, Sili Chen, Junhao Liew, Donny Y Chen, Zhenyu Li, Guang Shi, Jiashi Feng, and Bingyi Kang. Depth anything 3: Recovering the visual space from any views. *arXiv preprint arXiv:2511.10647*, 2025.

Lu Ling, Yichen Sheng, Zhi Tu, Wentian Zhao, Cheng Xin, Kun Wan, Lantao Yu, Qianyu Guo, Zixun Yu, Yawen Lu, et al. Dl3dv-10k: A large-scale scene dataset for deep learning-based 3d vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22160–22169. IEEE, 2024.

Yifan Mo, Youcheng Cai, and Ligang Liu. Plug-and-play pde optimization for 3d gaussian splatting: Toward high-quality rendering and reconstruction. *arXiv preprint arXiv:2509.13938*, 2025.

Shiwei Ren, Tianci Wen, Yongchun Fang, and Biao Lu. Fastgs: Training 3d gaussian splatting in 100 seconds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26094–26103, 2026.

Brandon Smart, Chuanxia Zheng, Iro Laina, and Victor Adrian Prisacariu. Splatt3r: Zero-shot gaussian splatting from uncalibrated image pairs. *arXiv preprint arXiv:2408.13912*, 2024.

Shupeng Su, Chao Zhang, Kai Han, and Yonghong Tian. Greedy hash: Towards fast optimization for accurate hash coding in cnn. *Advances in neural information processing systems*, 31, 2018.

Jianyuan Wang, Minghao Chen, Shangzhan Zhang, Nikita Karaev, Johannes Schönberger, Patrick Labatut, Piotr Bojanowski, David Novotny, Andrea Vedaldi, and Christian Rupprecht. VGGT-Ω, 2026. URL https://arxiv.org/abs/2605.15195.

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

Haofei Xu, Songyou Peng, Fangjinhua Wang, Hermann Blum, Daniel Barath, Andreas Geiger, and Marc Pollefeys. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16453–16463. IEEE, 2025.

Botao Ye, Boqi Chen, Haofei Xu, Daniel Barath, and Marc Pollefeys. Yonosplat: You only need one model for feedforward 3d gaussian splatting. In *International Conference on Learning Representations (ICLR)*, volume 2026, pp. 39852–39871, 2026.