DENSITOK: MAKING FEED-FORWARD 3D GAUSSIAN SPLATTING SEE MORE VIEWS THAN IT IS GIVEN

Anonymous authors
Paper under double-blind review

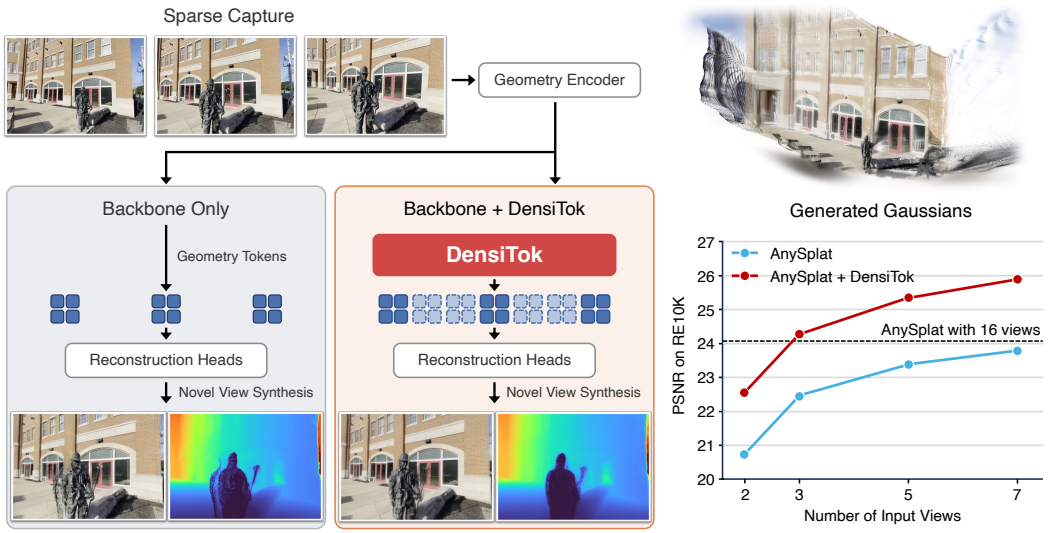

Figure 1: **DensiTok** augments geometry tokens for unobserved views while keeping the pretrained encoder and reconstruction heads frozen. With only three input views, it enables AnySplat to exceed its 16-view baseline in PSNR on RealEstate10K.

ABSTRACT

Feed-forward 3D Gaussian Splatting (3DGS) reconstructs a scene in a single forward pass, replacing per-scene optimization with a network trained across many scenes. Its quality, however, degrades sharply as the number of input images drops. The bottleneck is upstream of the reconstruction heads: from a few unposed views, the internal representation they read carries no evidence for unobserved regions, leaving holes, floaters, and blur. The common remedy supplies that evidence as pixels, synthesizing extra views with an image or video generator and re-encoding them, which is costly and not 3D-consistent by construction. *We instead densify the evidence itself.* We present **DensiTok**, a plug-in module for pretrained feed-forward 3DGS models that densifies their internal geometry tokens directly, making a frozen backbone behave as though it had observed many more views than it was given. DensiTok compresses those tokens into a compact latent space, completes the latents of the unobserved viewpoints in a single flow-matching step conditioned on camera geometry, and decodes them back into tokens that the original reconstruction heads. The same module design can be integrated into different pretrained predictors while keeping each backbone and its reconstruction heads frozen. Completion in a low-dimensional latent space requires no image synthesis or additional encoder passes. Across three pretrained backbones and two benchmarks, DensiTok consistently improves sparse-view reconstruction and recovers much of the gap to dense-view reconstruction.

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.

1 INTRODUCTION

Feed-forward 3D Gaussian Splatting predicts cameras, geometry, and appearance in a single network pass, replacing per-scene optimization with a model amortized over many scenes. This makes reconstruction practical from unposed, unconstrained image collections, and recent backbones accept an arbitrary number of input views. Such models (Jiang et al., 2025; Liu et al., 2025; Lin et al., 2025) share a common anatomy: a geometry encoder maps the input images to a set of view-aligned geometry tokens, and camera, depth, and Gaussian heads decode that shared representation into the final reconstruction. Their quality, however, tracks the visual evidence they are given. The right side of Fig. 1 plots reconstruction quality against the number of input views for several recent backbones, and it degrades in the few-view regime that motivated feed-forward reconstruction in the first place. The cause sits upstream of the heads: a handful of unposed images gives the encoder little cross-view evidence, so its tokens describe only the surfaces that were actually observed. Occluded and uncovered surfaces have no tokens at all, and the heads turn that gap into holes, floaters, and blurred novel views. *What the model lacks, then, is not pixels but tokens.* This suggests a different target: rather than acquiring the views it never saw, synthesize the geometry tokens those views would have produced.

Existing remedies differ mainly in where they supply the missing evidence. The dominant line of work operates in image space, either synthesizing extra input views with a multi-view or video diffusion model (Gao et al., 2024; Zhong et al., 2025), or repairing those renderings, and refitting the scene to them (Chen et al., 2024b; Yin et al., 2025). Both incur two separate costs: running a large generative model at full image resolution, and then running the geometry encoder again over the enlarged set of views. Because pixels are not a shared 3D representation, consistency across generated views is encouraged but never enforced, so the reconstruction must fit views that disagree with one another. More fundamentally, pixels are a detour. *The reconstruction needs only tokens for the missing viewpoints.* These methods produce entire RGB frames instead, which the encoder then compresses into those tokens, so most of what the generator produced is discarded immediately. A second line of work supplies no new evidence at all, and instead trains feed-forward architectures specialized for sparse views, extracting as much as possible from the correspondences the input views do provide (Chen et al., 2024a; Xu et al., 2025a; Smart et al., 2024). Such models remain bounded by what those views cover, and they are standalone architectures tied to a small, fixed number of views. Closest to our setting, recent work (Jang et al., 2026) generates directly in token space, but without a compressed latent it must run cascaded multi-step generators on features with over a thousand channels, so generation stays expensive.

We introduce **DensiTok**, a plug-in that densifies the geometry tokens of a pretrained feed-forward 3DGS model so that its heads reconstruct as if from far more views. As shown in Fig. 1, DensiTok sits between the geometry encoder and the reconstruction heads, with the encoder frozen and the heads either frozen or minimally adapted. The encoder's output is naturally organized as a sequence of per-view token sets, one for each input image. DensiTok augments token sets for viewpoints that were never observed, so the heads receive the representation of a dense capture even though only a few images were encoded. Doing this directly on the tokens is costly: the heads read tokens from several layers of the encoder, and each layer has over a thousand channels in current backbones. DensiTok instead treats the whole stack of layers as one object: it compresses them into a single compact latent per view, completes the unobserved views in that latent space, and expands the result back into tokens for every layer.

Two components realize this design. The first is a variational autoencoder (VAE) (Kingma & Welling, 2013) with **Level-Adaptive Modulation (LAM-VAE)**. Its encoder predicts how much each feature level should contribute for each token, fuses the features accordingly, and compresses the fused tokens into compact latents. This lets a single latent stand in for all layers without fixing in advance which layers matter, since that varies across backbones and even across scenes. The second is a Diffusion Transformer (DiT) (Peebles & Xie, 2023) for **Geometry Latent Augmentation**: it extends the latent sequence with the unobserved views, generating their latents from the observed ones in a single flow-matching (Lipman et al., 2022) step, conditioned on their cameras. Since the input is unposed, these cameras are not given. The backbone's own camera head predicts them for the observed views, and the cameras of the new views are interpolated along the path through those predictions, so the new views densify the captured trajectory rather than extrapolate beyond it. The VAE decoder then expands the completed latent sequence back into tokens at every layer the heads

expect, and the camera, depth, and Gaussian heads decode them. Image-space methods supply the missing evidence indirectly, as pixels that the encoder must process again. DensiTok produces the evidence itself, once and in a compact latent space, in exactly the form the heads already consume.

We evaluate DensiTok on RealEstate10K (Zhou et al., 2018) and DL3DV (Ling et al., 2024) with three representative feed-forward 3DGS backbones: AnySplat (Jiang et al., 2025), WorldMirror (Liu et al., 2025), and Depth Anything 3 (Lin et al., 2025). DensiTok consistently improves novel-view rendering quality from sparse, unposed inputs. This comes at a bounded cost: the encoder still processes only the sparse inputs, and generation is a single step in a compact latent space, so where DensiTok reaches dense-view quality, it does so in less time than dense-view reconstruction itself.

2 Related Work

2.1 Feed-Forward Gaussian Splatting

Feed-forward 3DGS amortizes reconstruction across scenes by using a single network to predict Gaussian attributes, cameras, and depth from a set of images in one pass. Posed methods build explicit correspondence machinery, with pixelSplat (Charatan et al., 2024) using epipolar matching, MVSplat (Chen et al., 2024a) constructing cost volumes, and DepthSplat (Xu et al., 2025a) adding monocular depth priors. Splatt3R (Smart et al., 2024), NoPoSplat (Ye et al., 2025), PF3plat (Hong et al., 2025), and SelfSplat (Kang et al., 2025) remove the pose requirement for sparse unposed inputs. More recent general backbones, including AnySplat (Jiang et al., 2025), WorldMirror (Liu et al., 2025), and Depth Anything 3 (Lin et al., 2025), accept an arbitrary number of unconstrained views and jointly predict cameras, geometry, and Gaussians, typically by pairing a large geometry transformer with lightweight decoding heads. All of these models, however, are bounded by what their inputs show. With only a few views, overlap is limited, pose and depth are ambiguous, so the geometry tokens describe only the visible parts of the scene. Therefore, they render everything the tokens do not cover as holes, floaters, and blurred novel views. DensiTok instead treats sparse-view reconstruction as a completion problem inside the representation, augmenting token sets for additional viewpoints.

2.2 Generative Priors for 3D Reconstruction

Image-space generative priors are typically incorporated in two ways. The first augments the sparse input set with pose-controlled synthetic views before reconstruction, as in CAT3D (Gao et al., 2024), VistaDream (Wang et al., 2025a), and Bolt3D (Szymanowicz et al., 2025). The second starts from a coarse 3DGS reconstruction, renders it along novel camera trajectories, and uses a video or multi-view generative model to refine or complete the rendered sequence, as in MVSplat360 (Chen et al., 2024b), FlowR (Fischer et al., 2025), VideoScene (Wang et al., 2025b), GSFixer (Yin et al., 2025), and G4Splat (Ni et al., 2026). Both can synthesize plausible content for unobserved regions, but the generative step operates in RGB or generic video latent spaces with no shared 3D representation to respect. Consequently, inconsistencies across the generated views turn into geometric artifacts. Both also incur two separate costs: a generative model runs for many sampling steps at image resolution, and the scene must then be reconstructed from the enlarged set of views. DensiTok instead generates in the backbone's own token space, so the completed representation is decoded into one Gaussian scene by the same heads, with no image synthesis and no second reconstruction.

2.3 Generation in 3D-Aware Latent Spaces

A separate line avoids image synthesis altogether by generating in a latent space that decodes straight to 3D. LN3Diff (Lan et al., 2025) diffuses a 3D-structured latent decoded into neural fields, Prometheus (Yang et al., 2025) generates multi-view RGB-D latents that a Gaussian decoder turns into pixel-aligned Gaussians, and Wonderland (Liang et al., 2025) lifts camera-controlled video latents directly to 3DGS. Most recently, GLD (Jang et al., 2026) uses the features of a pretrained geometry encoder (Wang et al., 2025c) as the latent space itself, applying flow matching to selected feature levels for feature-space novel-view synthesis and deriving the remaining levels by propagating the result through the frozen encoder. This removes the need for a VAE, but the generated levels keep the encoder's full width of over a thousand channels. Therefore, GLD runs cascaded multi-step

generators on them, decodes RGB with a separately trained decoder. DensiTok differs on each point. It compresses all feature levels into a single four-channel latent and completes the sequence in one step to produce a Gaussian scene rather than images.

3 METHOD

3.1 PROBLEM FORMULATION

A feed-forward 3DGS model takes N unposed images $I_1, \ldots, I_N$ and reconstructs the scene in a single forward pass. Internally, a geometry encoder $\mathcal{E}_{\text{geo}}$, a deep transformer, processes all N images jointly and represents each image as K tokens, each a d_f-dimensional vector. Such tokens exist after every layer of the encoder, but the pretrained reconstruction heads do not read only the last one: they read a fixed set of L intermediate layers (e.g., 4 of the 24 layers), chosen when the backbone was trained. We call these L layers *levels* and index them by $\ell = 1, \ldots, L$ from shallow to deep. The part of the encoder's output that the heads consume is therefore

$$(\mathbf{F}_1, \ldots, \mathbf{F}_L) = \mathcal{E}_{\text{geo}}(I_1, \ldots, I_N), \qquad \mathbf{F}_\ell \in \mathbb{R}^{N \times K \times d_f}, \tag{1}$$

where $\mathbf{F}_\ell$ collects the tokens of all N views at level ℓ. The camera, depth, and Gaussian heads decode $(\mathbf{F}_1, \ldots, \mathbf{F}_L)$ jointly into cameras, depth maps, and a Gaussian scene. Everything the heads know about the scene is contained in these tokens.

In our setting, only n ($< N$) images are available. We still reason about N views in total: the n observed views, for which we have images, and $(N - n)$ unobserved views, which are virtual viewpoints along the camera path that have no image. We index all N views by $i = 1, \ldots, N$ and record which ones are observed with a binary mask $\mathbf{m} \in \{0, 1\}^N$, with $m_i = 1$ for observed views and $m_i = 0$ for unobserved ones. Running the encoder on the n observed images gives tokens for those views only,

$$(\mathbf{F}_1^{\text{obs}}, \ldots, \mathbf{F}_L^{\text{obs}}) = \mathcal{E}_{\text{geo}}(\{I_i \mid m_i = 1\}), \qquad \mathbf{F}_\ell^{\text{obs}} \in \mathbb{R}^{n \times K \times d_f}, \tag{2}$$

and the unobserved views have no tokens at all. DensiTok fills this gap. For every level ℓ it produces a full-size token tensor $\tilde{\mathbf{F}}_\ell \in \mathbb{R}^{N \times K \times d_f}$ that covers all N views, generating the tokens of the $(N - n)$ unobserved views from the observed ones. The target is what the encoder would have produced had all N images been available, so that the heads can decode $(\tilde{\mathbf{F}}_1, \ldots, \tilde{\mathbf{F}}_L)$ exactly as they decode $(\mathbf{F}_1, \ldots, \mathbf{F}_L)$ in the full-view case. We keep the geometry encoder and reconstruction heads frozen by default. However, due to the structure of feed-forward 3DGS reconstruction heads, we also consider minimal appearance adaptation, as detailed later.

3.2 VAE WITH LEVEL-ADAPTIVE MODULATION

As shown in Fig. 2, Level-Adaptive Modulation (LAM-VAE) combines level-adaptive fusion with the VAE encoder. We refer to this combined encoding module as the **LAM-VAE encoder**.

Level-adaptive fusion. Because the heads read all L levels jointly, DensiTok has to produce $\tilde{\mathbf{F}}_\ell$ for every level, and generating each level with its own model would multiply the cost of an already high-dimensional problem by L. We instead fuse the L levels of every token into a single vector before compressing it into a d_z-dimensional latent, so that one VAE and one generator handle all L levels. Which levels carry the most useful information can vary across backbones and even across scenes, so rather than fixing a subset of levels in advance (Jang et al., 2026), we let each token decide how much every level contributes. All operations in this paragraph act on one token at a time, so we drop the view and token indices and write $\mathbf{f}_\ell \in \mathbb{R}^{d_f}$ for the token's feature at level ℓ. Each $\mathbf{f}_\ell$ is first normalized with a LayerNorm (Ba et al., 2016) specific to its level, so that no level dominates the others by magnitude. A gating network then decides how much each level contributes. It summarizes each $\mathbf{f}_\ell$ with a shared linear layer, concatenates the L summaries, and outputs one logit per level through an MLP. Because the MLP sees all L summaries at once, the logit of each level depends on the others, so the gate compares levels rather than scoring each in isolation. A softmax turns the logits into weights $\alpha_1, \ldots, \alpha_L$ that sum to one. We use gains $g_\ell = L\,\alpha_\ell$ rather than the weights themselves: the gains average to one, so uniform weights recover the plain concatenation

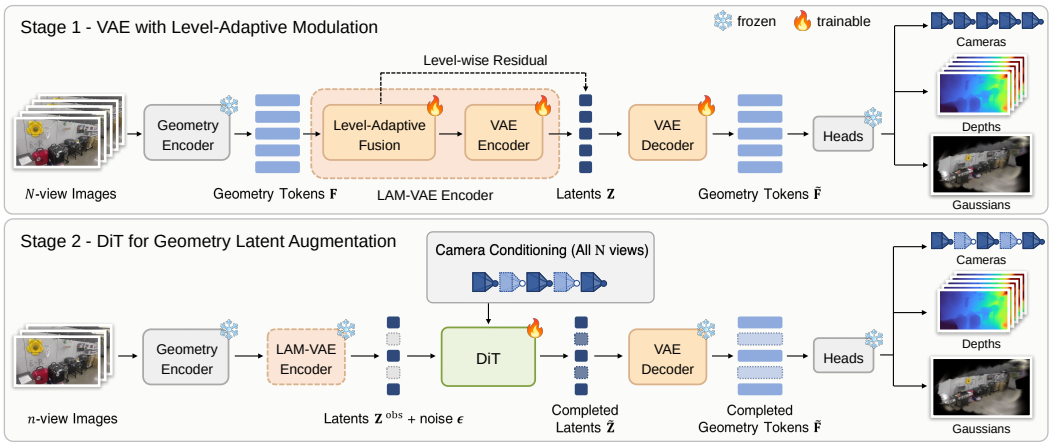

Figure 2: **Geometry token compression and completion.** Stage 1 trains a VAE with Level-Adaptive Modulation (LAM-VAE) to compress and reconstruct geometry tokens. level-adaptive fusion and the VAE encoder together form the LAM-VAE encoder, with a level-wise residual into the posterior mean. Stage 2 freezes the VAE and trains a camera-conditioned DiT to complete latents for unobserved views. The geometry encoder remains frozen throughout, while the reconstruction heads remain frozen or undergo minimal appearance adaptation.

of all levels, and gating acts as a reweighting around that baseline instead of shrinking the input by a factor of L. The fused token is then a gain-weighted sum of level-wise linear projections,

$$\mathbf{h} = \sum_{\ell=1}^{L} g_\ell \, \mathbf{W}_\ell \mathbf{f}_\ell, \tag{3}$$

where $\mathbf{W}_\ell$ is a learnable linear projection that maps level ℓ to the width of the VAE encoder transformer. Equivalently, one can concatenate the gain-weighted levels and apply a single linear layer. The fused tokens of all n observed views form one sequence that enters the VAE encoder transformer, which follows the alternating frame and global attention of VGGT (Wang et al., 2025c). Frame blocks attend among the tokens of one view, global blocks attend across all views, and 2D rotary position embeddings (Su et al., 2024) encode each token's position on the patch grid. For every token the transformer outputs a Gaussian posterior over a d_z-dimensional latent, parameterized by a mean $\boldsymbol{\mu}_0 \in \mathbb{R}^{d_z}$ and a log-variance $\log \boldsymbol{\sigma}^2 \in \mathbb{R}^{d_z}$, with $d_z = 4$ in our main setting.

Level-wise residual into the posterior mean. Compressing the fused token $\mathbf{h}$ into a d_z-dimensional latent, is what makes generation in a compact latent space possible. However, such a narrow bottleneck can erase detail that is specific to a single level, especially since the levels have already been mixed in $\mathbf{h}$ before the transformer sees them. Each normalized level feature $\mathbf{f}_\ell$ is projected into the latent space by a level-specific linear layer $\mathbf{P}_\ell \in \mathbb{R}^{d_z \times d_f}$, and the projections are combined with the same weights α_ℓ from the fusion step,

$$\boldsymbol{\mu} = \boldsymbol{\mu}_0 + \sum_{\ell=1}^{L} \alpha_\ell \, \mathbf{P}_\ell \mathbf{f}_\ell, \tag{4}$$

so the mean of each token is corrected by exactly the levels that dominate its fusion. The residual touches only the mean; the log-variance $\log \boldsymbol{\sigma}^2$ predicted by the VAE encoder is kept as is. During VAE training, the latent of each token is sampled as $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}_{\text{vae}}$ with $\boldsymbol{\epsilon}_{\text{vae}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. At inference we drop the noise and use the mean itself as the deterministic latent representation, $\mathbf{z} = \boldsymbol{\mu} \in \mathbb{R}^{d_z}$. Stacking the latents of all K tokens of all encoded N views gives $\mathbf{Z} \in \mathbb{R}^{N \times K \times d_z}$; in the completion setting only the n observed views are encoded, yielding $\mathbf{Z}^{\text{obs}} \in \mathbb{R}^{n \times K \times d_z}$, which is the input to the generator described next. The LAM-VAE encoder includes this level-wise residual, so its complete path from geometry tokens to $\mathbf{Z}$ comprises level-adaptive fusion, the VAE encoder transformer, and the residual into the posterior mean.

3.3 DiT for Geometry Latent Augmentation

We use a Diffusion Transformer (DiT) (Peebles & Xie, 2023) for geometry latent augmentation: given the observed latents $\mathbf{Z}^{\text{obs}}$, it generates latents for the $(N-n)$ unobserved views and returns a completed tensor $\tilde{\mathbf{Z}} \in \mathbb{R}^{N\times K\times d_z}$ whose observed views are $\mathbf{Z}^{\text{obs}}$. The generated views are trained to match the corresponding views of $\mathbf{Z}$, the latents obtained when all N images are encoded, which are available only at training time. The DiT is conditioned on three inputs: the observed latents $\mathbf{Z}^{\text{obs}}$; the mask $\mathbf{m}$ from the problem formulation (Sec. 3.1), which marks the observed views; and cameras $\mathbf{g}_1, \ldots, \mathbf{g}_N$ for all N views, since an unobserved view has no image and its camera is the only information that specifies which view to generate. How the cameras are obtained differs between training, where ground truth exists for all N views, and inference, where only the cameras of the observed views can be predicted.

Latent sequence. $\mathbf{Z}^{\text{obs}}$ covers only the n observed views, whereas the DiT works on all N views at once. We therefore lay out an N-view sequence along the camera path. The latents of each observed view sit at that view's position and stay clean throughout; they are the condition. The positions of unobserved views are what the DiT fills in. We use rectified flow matching (Lipman et al., 2022; Liu et al., 2022), with $t = 0$ denoting clean latents and $t = 1$ pure noise. During training, the latents of view i at time t are

$$\mathbf{x}_{t,i} = m_i\, \mathbf{z}_i^{\text{obs}} + (1-m_i)\big[(1-t)\,\mathbf{z}_i + t\,\boldsymbol{\epsilon}_i\big], \qquad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{5}$$

where $\mathbf{z}_i$ and $\mathbf{z}_i^{\text{obs}} \in \mathbb{R}^{K\times d_z}$ are the latents of view i in $\mathbf{Z}$ and $\mathbf{Z}^{\text{obs}}$. At inference, sampling starts from the $t = 1$, which requires only $\mathbf{Z}^{\text{obs}}$ and noise. Each view carries its own time $\tau_i = t(1-m_i)$: observed views stay at the clean time zero, unobserved views follow the global time t. This lets a single network handle both kinds of views in one sequence.

Camera conditioning and latent completion. Unobserved views enter as pure noise, so their cameras are the only signal that tells the DiT which view to generate. We therefore encode the cameras of all N views as Plücker rays. For every patch token of every view, we compute the ray through that patch center, so the Plücker ray embedding has the same K-token layout as the latents. The few tokens that do not correspond to a patch, such as the camera token, receive a learnable embedding instead. These embeddings, together with the time τ_i (zero for observed views, t for unobserved ones), condition every DiT block through adaptive LayerNorm (AdaLN) (Peebles & Xie, 2023). The blocks alternate frame and global attention in the same pattern as the VAE encoder, so that noisy unobserved views gather evidence from the clean observed views in the global blocks. The DiT predicts the flow velocity with respect to increasing t, from clean latents toward noise, and the sampler integrates this velocity backward from $t = 1$ to $t = 0$ with Euler steps, resetting the observed views to $\mathbf{Z}^{\text{obs}}$ after every step so they never drift; the result is the completed tensor $\tilde{\mathbf{Z}}$. Training supervises only the unobserved views, including the finetuning that reduces sampling to a single Euler step.

3.4 Camera Conditioning at Training and Inference

The DiT is conditioned on the cameras of all N views, yet at inference only the n observed views come with images, so the N cameras have to be obtained differently at training and inference. In both regimes, the backbone's pretrained camera head first predicts the cameras of the observed views from the observed tokens $\mathbf{F}^{\text{obs}}$; we denote these predicted cameras by $\{\hat{\mathbf{g}}_i^{\text{obs}}\}_{i=1}^{n}$.

Training. During training, the dataset provides ground-truth cameras $\{\mathbf{g}_i^{\text{gt}}\}_{i=1}^{N}$ for all N views. They cannot be used as n-view conditions directly: the backbone reconstructs at an arbitrary scale, so its predicted cameras $\{\hat{\mathbf{g}}_i^{\text{obs}}\}_{i=1}^{n}$ live in a differently scaled world, and a DiT trained on raw ground-truth cameras would face a mismatch at inference, where only predicted cameras exist. Both sets of cameras are expressed relative to the first observed view, following the convention of the backbone's camera head, so the two worlds already share their global rotation and translation and differ only in scale. We remove this last difference by rescaling all ground-truth camera centers by a single factor, the ratio of the spread of the predicted centers to that of the ground-truth centers over the observed views, and denote the aligned cameras by $\{\mathbf{g}_i^{\text{aligned}}\}_{i=1}^{N}$. The DiT is trained on these aligned cameras and thus only ever sees cameras in the world it will encounter at inference.

Table 1: Quantitative results on RealEstate10K and DL3DV with $n \in \{2, 3, 5, 7\}$ sparse input views and dense-view references ($N = 16$).

Methods	RealEstate10K (Zhou et al., 2018)							DL3DV (Ling et al., 2024)							Runtime
	Appearance			Camera		Depth		Appearance			Camera		Depth		
	PSNR↑	SSIM↑	LPIPS↓	AUC@3° ↑	AUC@30° ↑	$\delta_{1.25}$ ↑	AbsRel↓	PSNR↑	SSIM↑	LPIPS↓	AUC@3° ↑	AUC@30° ↑	$\delta_{1.25}$ ↑	AbsRel↓	(s)↓
Dense-view ($N = 16$)															
AnySplat (Jiang et al., 2025)	24.04	0.792	0.189	20.4	77.5	0.835	0.151	22.30	0.683	0.246	24.1	85.4	0.803	0.158	0.434
WorldMirror (Liu et al., 2025)	25.11	0.856	0.161	41.8	83.3	0.880	0.110	24.47	0.819	0.173	69.0	94.0	0.825	0.141	0.495
Sparse-view ($n = 2$)															
AnySplat	20.73	0.706	0.250	20.0	77.1	0.828	0.168	19.85	0.626	0.287	22.5	85.8	0.774	0.186	0.100
+ DensiTok	**22.56**	**0.766**	**0.223**	**26.7**	**78.9**	**0.846**	**0.153**	**22.40**	**0.699**	**0.255**	**30.4**	**88.9**	**0.801**	**0.170**	0.258
WorldMirror	21.56	0.775	0.221	48.3	86.8	0.850	0.136	20.60	0.706	0.251	63.8	93.3	0.784	0.171	0.109
+ DensiTok	**22.06**	**0.791**	**0.215**	**50.0**	**87.2**	**0.854**	**0.133**	**21.40**	**0.729**	**0.242**	**65.8**	**94.2**	**0.806**	**0.158**	0.263
Sparse-view ($n = 3$)															
AnySplat	22.47	0.748	0.215	21.1	77.0	0.830	0.159	21.66	0.677	0.245	24.2	86.4	0.792	0.167	0.117
+ DensiTok	**24.28**	**0.813**	**0.184**	**25.8**	**79.7**	**0.853**	**0.146**	**24.28**	**0.755**	**0.205**	**34.4**	**89.6**	**0.809**	**0.158**	0.267
WorldMirror	23.29	0.816	0.190	48.3	86.4	0.868	0.119	22.54	0.768	0.201	66.7	94.3	0.810	0.152	0.133
+ DensiTok	**23.81**	**0.836**	**0.181**	**49.6**	**87.0**	**0.874**	**0.117**	**23.35**	**0.787**	**0.189**	**70.0**	**95.5**	**0.823**	**0.147**	0.277
Sparse-view ($n = 5$)															
AnySplat	23.39	0.768	0.203	20.6	76.6	0.832	0.155	22.58	0.690	0.234	24.3	85.5	0.799	0.161	0.160
+ DensiTok	**25.35**	**0.838**	**0.162**	**25.2**	**78.8**	**0.852**	**0.141**	**25.33**	**0.782**	**0.182**	**33.5**	**87.7**	**0.816**	**0.153**	0.299
WorldMirror	24.29	0.837	0.175	46.0	84.8	0.876	0.112	23.67	0.798	0.180	66.5	93.7	0.818	0.146	0.182
+ DensiTok	**24.91**	**0.861**	**0.162**	**47.3**	**85.3**	**0.880**	**0.109**	**24.38**	**0.825**	**0.167**	**68.9**	**94.3**	**0.833**	**0.138**	0.320
Sparse-view ($n = 7$)															
AnySplat	23.79	0.780	0.195	20.9	76.7	0.831	0.153	23.01	0.709	0.225	23.1	84.9	0.806	0.157	0.207
+ DensiTok	**25.89**	**0.851**	**0.150**	**24.5**	**78.5**	**0.853**	**0.137**	**25.71**	**0.796**	**0.169**	**34.0**	**86.7**	**0.817**	**0.151**	0.333
WorldMirror	24.75	0.847	0.167	44.5	84.1	0.880	0.111	24.28	0.816	0.170	63.7	92.7	0.824	0.143	0.235
+ DensiTok	**25.31**	**0.866**	**0.155**	**45.8**	**84.7**	**0.884**	**0.107**	**25.17**	**0.851**	**0.152**	**67.3**	**93.3**	**0.832**	**0.137**	0.361

Inference. At inference no ground-truth cameras exist, and the predicted cameras $\{\hat{\mathbf{g}}_i^{\text{obs}}\}_{i=1}^{n}$ are themselves the reference, so no alignment is needed. What is missing are the cameras of the $(N-n)$ unobserved views. We obtain them by fitting a smooth trajectory through the predicted cameras, and sampling it at the unobserved views: camera centers are interpolated with a natural cubic spline, rotations with spherical interpolation. We denote the resulting cameras by $\{\hat{\mathbf{g}}_i^{\text{interp}}\}_{i=1}^{N-n}$ for $m_i = 0$. Because the unobserved views are virtual, we place all of them between the first and last observed cameras, so the trajectory is only ever interpolated to densify the observed camera path. In summary, the cameras that condition the DiT are the aligned ground-truth cameras $\{\mathbf{g}_i^{\text{aligned}}\}_{i=1}^{N}$ during training, and at inference $\{\hat{\mathbf{g}}_i^{\text{obs}}\}_{i=1}^{n}$ for observed views and $\{\hat{\mathbf{g}}_i^{\text{interp}}\}_{i=1}^{N-n}$ for unobserved views.

3.5 Overall Training Strategy

DensiTok is trained in three stages, with the geometry encoder frozen throughout. In Stage 1, the VAE with **Level-Adaptive Modulation** is trained to reconstruct the geometry tokens of the frozen geometry encoder. In Stage 2, the DiT for **geometry latent augmentation** is trained on top of the frozen VAE with the rectified flow-matching objective on the unobserved views. At this stage, we can keep the Gaussian head frozen or minimally finetune it, as detailed in Appendix A.2. In Stage 3, the model is finetuned for one-step sampling by fixing the time of the unobserved views at $t = 1$, so that a single Euler step maps noise directly to the completed latents.

4 Experiments

4.1 Setup

Datasets. We train and evaluate DensiTok separately on RealEstate10K (Zhou et al., 2018) and DL3DV-10K (Ling et al., 2024), using ordered sequences of $N = 16$ view in the main setting. During training, we sample views from both datasets following WorldMirror (Liu et al., 2025). For evaluation, we use fixed sequences spanning three difficulty levels based on endpoint image overlap, with nested sets of $n \in \{2, 3, 5, 7\}$ condition views.

Implementation Details. We train DensiTok separately for three pretrained backbones: AnySplat (Jiang et al., 2025), WorldMirror (Liu et al., 2025), and Depth Anything 3 (Lin et al., 2025). Following WorldMirror, we randomly sample a patch-aligned input resolution for each training batch.

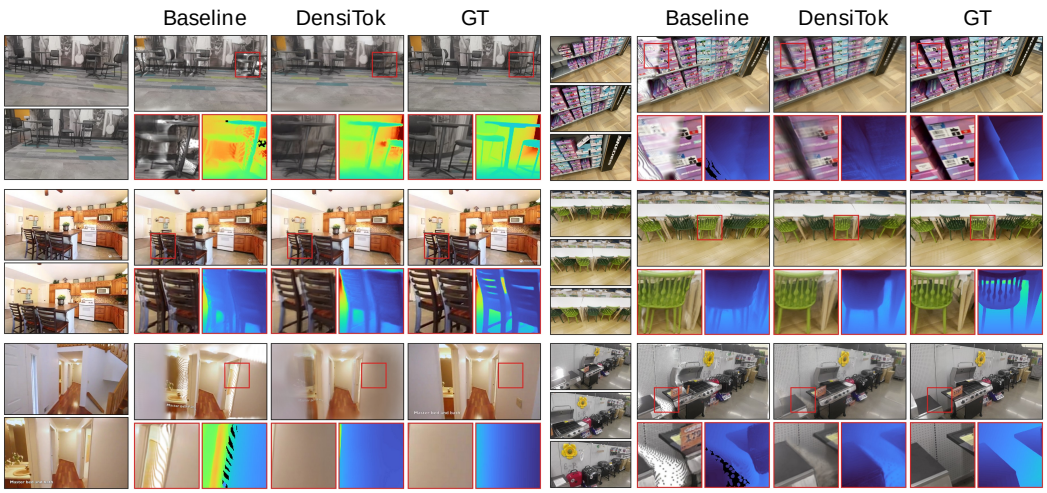

Figure 3: Qualitative novel-view RGB and depth comparisons for AnySplat with and without DensiTok on RE10K and DL3DV, using $n = 2$ input views (left) and $n = 3$ input views (right).

We evaluate the models at a fixed resolution of 294×518 pixels. Also, we randomly sample training sequences, draw n uniformly from $\{2, \ldots, 8\}$, and select the n observed views uniformly without replacement. We train LAM-VAE and the DiT for 200k iterations each, then finetune the DiT for one-step inference for an additional 20k iterations. All training and evaluation are conducted on four NVIDIA RTX 6000 Ada GPUs. Other training details are provided in Appendix B.

Evaluation Metrics. Since DensiTok enhances geometry tokens shared by the Gaussian, camera, and depth heads, we evaluate all three outputs. We use PSNR, SSIM, and LPIPS for rendering quality. Following the evaluation protocols of VGGT (Wang et al., 2025c) and VGGT-Ω (Wang et al., 2026), we measure camera pose accuracy using AUC at 3° and 30° and depth accuracy using $\delta_{1.25}$ and absolute relative error (AbsRel). Since RE10K and DL3DV do not provide official ground-truth depth, we use MoGe-2 (Wang et al., 2025d) predictions as pseudo-labels for depth evaluation.

4.2 Results

Quantitative results. Tab. 1 compares appearance quality, camera pose accuracy, depth accuracy, and runtime for AnySplat and WorldMirror on RE10K and DL3DV with $n \in \{2, 3, 5, 7\}$ sparse input views and dense-view references with $N = 16$. Corresponding results for DA3 are available in Appendix Tab. 6. For all appearance evaluations, we follow the scale-alignment procedure used by AnySplat (Jiang et al., 2025). DensiTok improves all three appearance metrics for every backbone, dataset, and evaluated sparse-view setting. With only $n = 3$ inputs, AnySplat with DensiTok also surpasses the $N = 16$ AnySplat reference in all three appearance metrics on both datasets. DensiTok consistently improves depth estimation across all two backbones, both datasets, and all evaluated sparse-view settings. Camera estimation also improves in most cases. These results suggest that completing the shared geometry representation benefits both appearance and geometry.

Qualitative results. Fig. 3 compares novel-view RGB and depth results under sparse-view conditions on RE10K and DL3DV using AnySplat with and without DensiTok. In challenging scenes, DensiTok improves alignment with the target views, geometric reconstruction, and RGB rendering quality. This is particularly evident with only $n = 2$ inputs, where the baseline leaves missing or fragmented structures. DensiTok completes geometric information in these gaps and occluded regions, yielding more coherent surfaces and clearer object boundaries. Additional qualitative results for other backbones are provided in Appendix C.2.

4.3 Ablation Studies

Effect of LAM-VAE encoder design. Tab. 2 compares LAM-VAE encoder designs with $n = 2$ observed views. Uniform fusion fixes the gating weights to $\alpha_\ell = 0.25$ for all four encoder levels,

Table 2: Ablations of Level-Adaptive Modulation with AnySplat and $n = 2$ observed views.

	RE10K							DL3DV						
LAM-VAE encoder	PSNR	SSIM	LPIPS	AUC@3°	AUC@30°	$\delta_{1.25}$	AbsRel	PSNR	SSIM	LPIPS	AUC@3°	AUC@30°	$\delta_{1.25}$	AbsRel
Uniform fusion	19.76	0.673	0.266	18.3	76.3	0.808	0.184	19.53	0.611	0.314	21.7	85.2	0.760	0.199
MLP fusion	21.19	0.725	0.243	20.8	77.4	0.831	0.164	20.53	0.644	0.291	23.8	86.1	0.781	0.181
+ level-adaptive fusion	22.11	0.756	0.228	25.0	78.1	0.836	0.159	21.89	0.684	0.267	28.8	87.6	0.787	0.177
+ level-wise residual	21.86	0.747	0.234	24.2	78.6	0.841	0.156	21.56	0.675	0.274	27.5	88.2	0.794	0.173
+ both	**22.56**	**0.766**	**0.223**	**26.7**	**78.9**	**0.846**	**0.153**	**22.40**	**0.699**	**0.255**	**30.4**	**88.9**	**0.801**	**0.170**

Table 3: Effect of the total number of views N with AnySplat and $n = 2$ observed views.

	RE10K					DL3DV					Efficiency	
Total views (N)	PSNR	SSIM	LPIPS	AUC@3°	AUC@30°	PSNR	SSIM	LPIPS	AUC@3°	AUC@30°	Runtime	Peak GPU mem.
2	20.94	0.714	0.245	20.8	77.5	20.03	0.631	0.278	22.9	86.3	**0.109**	**5.527**
4	22.08	0.757	0.239	22.9	78.1	21.28	0.636	0.257	23.3	87.6	0.116	6.196
8	22.49	0.762	0.231	25.4	78.6	22.18	0.687	**0.251**	28.8	89.1	0.158	7.175
12	22.31	**0.768**	0.226	25.0	78.8	22.37	0.695	0.253	30.4	**89.3**	0.209	8.407
16	**22.56**	0.766	**0.223**	**26.7**	**78.9**	**22.40**	**0.699**	0.255	**30.4**	88.9	0.258	9.471

while MLP fusion compresses concatenated level features with a simple MLP. We evaluate level-adaptive fusion and the level-wise residual separately and together relative to the uniform fusion baseline. Each component improves on uniform and MLP fusion, while combining them yields the best results across all metrics on both datasets. This supports their complementary roles.

Effect of the number of views. To examine how the number of completed views affects reconstruction quality and efficiency, we vary the total number of views N with $n = 2$ observed views (Tab. 3). At $N = n = 2$, no tokens are generated for unobserved views, allowing us to assess the effect of refining the original geometry tokens through LAM-VAE. The limited gains over the original AnySplat baseline indicate that most of DensiTok's improvement comes from generating additional geometry tokens. Increasing N generates tokens for more unobserved views and generally improves reconstruction quality, at the cost of longer runtime and higher peak GPU memory. The additional views provide the reconstruction heads with geometry tokens for a denser set of viewpoints while keeping the observed inputs fixed. In our experiments, gains diminish beyond $N = 12$, making the tradeoff between quality and computational cost less favorable.

5 Limitations and Future Work

DensiTok currently restricts geometry token augmentation to view interpolation and does not implement view extrapolation beyond the observed camera interval. Virtual cameras are interpolated between the first and last observed cameras, and our evaluation likewise considers novel views within this interval. Our results therefore do not establish reconstruction quality for extrapolated viewpoints. Future work could extend virtual camera sampling beyond the observed trajectory and adapt latent completion to the weaker visual constraints at these viewpoints. A key challenge is to infer newly visible geometry and appearance while maintaining consistency with the observed scene.

6 Conclusion

DensiTok improves unposed feed-forward 3D Gaussian Splatting by completing the geometry tokens of unobserved views. LAM-VAE compresses multi-level tokens, and a camera-conditioned DiT completes missing-view latents in one step. Across three backbones on RealEstate10K and DL3DV, DensiTok improves rendering, camera, and depth accuracy, with sparse inputs surpassing dense-view references in some settings.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on the design of our ablation experiments. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on any other part of our research methodology or experiments, implement methods, assist with translation, or interpret results, and generating synthetic data sets, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used a generative AI agent (*Claude/Opus5.1*) to edit the paper for readability. All drafts were written by the authors; the authors then reviewed each draft and wrote their own revision feedback, and the AI agent was used only to apply this author-provided feedback to the text. We have reviewed all AI-assisted work. The final set of ablation experiments was selected by the authors, and all experiments were implemented, run, and analyzed by the authors without AI assistance. All AI-applied edits were manually checked by the authors to ensure that they faithfully reflect the intended revisions, preserve the original meaning, and do not introduce new or unsupported claims. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 3 describes the proposed method and its training and inference procedures, and Appendix A provides architecture details, model configurations, and the appearance adaptation of the Gaussian head. Section 4.1 describes the datasets, pretrained backbones, input resolution, hardware, and evaluation metrics, including the use of predicted depth as pseudo-labels. Appendix B specifies the training objectives, loss weights, and optimization settings, with the configurations for the main AnySplat experiments summarized in Table 5. The evaluation view-sampling procedure is detailed in Appendix B. The main ablation studies are described in Section 4.3, and additional ablation and generalization protocols are provided in Appendix D.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *2022 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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19457–19467. IEEE, 2024.

Sipeng Chen, Junliang Liu, Hewei Tang, and Shibo Li. Multi-fidelity flow matching: Cascaded refinement of pde solutions. *arXiv preprint arXiv:2605.16118*, 2026.

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*, pp. 370–386. Springer, 2024a.

Yuedong Chen, Chuanxia Zheng, Haofei Xu, Bohan Zhuang, Andrea Vedaldi, Tat-Jen Cham, and Jianfei Cai. Mvsplat360: Feed-forward 360 scene synthesis from sparse views. *Advances in Neural Information Processing Systems*, 37:107064–107086, 2024b.

Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 2432–2443. IEEE, 2017.

Johan Edstedt, Qiyu Sun, Georg Bökman, Mårten Wadenbäck, and Michael Felsberg. Roma: Robust dense feature matching. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19790–19800. IEEE, 2024.

Tobias Fischer, Samuel Rota Bulò, Yung-Hsu Yang, Nikhil Keetha, Lorenzo Porzi, Norman Müller, Katja Schwarz, Jonathon Luiten, Marc Pollefeys, and Peter Kontschieder. Flowr: Flowing from sparse to dense 3d reconstructions. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 27702–27712. IEEE, 2025.

Ruiqi Gao, Aleksander Hoł yński, Philipp Henzler, Arthur Brussee, Ricardo Martin-Brualla, Pratul Srinivasan, Jonathan T. Barron, and Ben Poole. Cat3d: Create anything in 3d with multi-view diffusion models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 75468–75494. Curran Associates, Inc., 2024. doi: 10.52202/079017-2403. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/89e4433fec4b99f1d859db57af1e0a0f-Paper-Conference.pdf.

Gonzalo Martin Garcia, Karim Abou Zeid, Christian Schmidt, Daan De Geus, Alexander Hermans, and Bastian Leibe. Fine-tuning image-conditional diffusion models is easier than you think. In *Proceedings of the Winter Conference on Applications of Computer Vision*, pp. 753–762, 2025.

Jing He, Haodong Li, Wei Yin, Yixun Liang, Leheng Li, Kaiqiang Zhou, Hongbo Zhang, Bingbing Liu, and YingCong Chen. Lotus: Diffusion-based visual foundation model for high-quality dense prediction. In *International Conference on Learning Representations*, volume 2025, pp. 89454–89467, 2025.

Sunghwan Hong, Jaewoo Jung, Heeseong Shin, Jisang Han, Jiaolong Yang, Chong Luo, and Seungryong Kim. Pf3plat: Pose-free feed-forward 3d gaussian splatting for novel view synthesis. In *Forty-second International Conference on Machine Learning*, 2025.

Ranran Huang and Krystian Mikolajczyk. No pose at all: Self-supervised pose-free 3d gaussian splatting from sparse views. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 27947–27957. IEEE, 2025.

Wooseok Jang, Seonghu Jeon, Jisang Han, Jinhyeok Choi, Minkyung Kwon, Seungryong Kim, Saining Xie, and Sainan Liu. Repurposing geometric foundation models for multi-view diffusion. In *European Conference on Computer Vision (ECCV)*, 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.

Gyeongjin Kang, Jisang Yoo, Jihyeon Park, Seungtae Nam, Hyeonsoo Im, Sangheon Shin, Sangpil Kim, and Eunbyung Park. Selfsplat: Pose-free and 3d prior-free generalizable 3d gaussian splatting. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22012–22022. IEEE, 2025.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Yushi Lan, Fangzhou Hong, Shangchen Zhou, Shuai Yang, Xuyi Meng, Yongwei Chen, Zhaoyang Lyu, Bo Dai, Xingang Pan, and Chen Change Loy. Ln3diff++: Scalable latent neural fields diffusion for speedy 3d generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

Lingen Li, Zhaoyang Zhang, Yaowei Li, Jiale Xu, Wenbo Hu, Xiaoyu Li, Weihao Cheng, Jinwei Gu, Tianfan Xue, and Ying Shan. Nvcomposer: Boosting generative novel view synthesis with multiple sparse and unposed images. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 777–787. IEEE, 2025.

Hanwen Liang, Junli Cao, Vidit Goel, Guocheng Qian, Sergei Korolev, Demetri Terzopoulos, Konstantinos N Plataniotis, Sergey Tulyakov, and Jian Ren. Wonderland: Navigating 3d scenes from a single image. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 798–810. 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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22160–22169. IEEE, 2024.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Yifan Liu, Zhiyuan Min, Zhenwei Wang, Junta Wu, Tengfei Wang, Yixuan Yuan, Yawei Luo, and Chunchao Guo. Worldmirror: Universal 3d world reconstruction with any-prior prompting. *arXiv preprint arXiv:2510.10726*, 2025.

Yuanxun Lu, Jingyang Zhang, Tian Fang, Jean-Daniel Nahmias, Yanghai Tsin, Long Quan, Xun Cao, Yao Yao, and Shiwei Li. Matrix3d: Large photogrammetry model all-in-one. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11250–11263. IEEE, 2025.

Junfeng Ni, Yixin Chen, Zhifei Yang, Yu Liu, Ruijie Lu, Song-Chun Zhu, and Siyuan Huang. G4splat: Geometry-guided gaussian splatting with generative prior. In *International Conference on Learning Representations*, volume 2026, pp. 138924–138951, 2026.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Thomas Schops, Johannes L Schonberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3260–3269, 2017.

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.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Stanislaw Szymanowicz, Jason Y Zhang, Pratul Srinivasan, Ruiqi Gao, Arthur Brussee, Aleksander Holynski, Ricardo Martin-Brualla, Jonathan T Barron, and Philipp Henzler. Bolt3d: Generating 3d scenes in seconds. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 24846–24857. IEEE, 2025.

Haiping Wang, Yuan Liu, Ziwei Liu, Wenping Wang, Zhen Dong, and Bisheng Yang. Vistadream: Sampling multiview consistent images for single-view scene reconstruction. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 26772–26782. IEEE, 2025a.

Hanyang Wang, Fangfu Liu, Jiawei Chi, and Yueqi Duan. Videoscene: Distilling video diffusion model to generate 3d scenes in one step. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16475–16485. IEEE, 2025b.

Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5294–5306. IEEE, 2025c.

Jianyuan Wang, Minghao Chen, Shangzhan Zhang, Nikita Karaev, Johannes Schönberger, Patrick Labatut, Piotr Bojanowski, David Novotny, Andrea Vedaldi, and Christian Rupprecht. Vggt-Ω. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21486–21499, June 2026.

Ruicheng Wang, Sicheng Xu, Yue Dong, Yu Deng, Jianfeng Xiang, Zelong Lv, Guangzhong Sun, Xin Tong, and Jiaolong Yang. Moge-2: Accurate monocular geometry with metric scale and sharp details. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 35928–35959. Curran Associates, Inc., 2025d. doi: 10.52202/085713-1207. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/336572db3e99930814d6b328d4220cb6-Paper-Conference.pdf.

Weijie Wang, Yeqing Chen, Zeyu Zhang, Hengyu Liu, Haoxiao Wang, Zhiyuan Feng, Wenkang Qin, Feng Chen, Jia-Wang Bian, Zheng Zhu, et al. Volsplat: Rethinking feed-forward 3d gaussian splatting with voxel-aligned prediction. *arXiv preprint arXiv:2509.19297*, 2025e.

Haofei Xu, Songyou Peng, Fangjinhua Wang, Hermann Blum, Daniel Barath, Andreas Geiger, and Marc Pollefeys. Depthsplat: Connecting gaussian splatting and depth. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16453–16463. IEEE, 2025a.

Jiale Xu, Shenghua Gao, and Ying Shan. Freesplatter: Pose-free gaussian splatting for sparse-view 3d reconstruction. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 25442–25452. IEEE, 2025b.

Yuanbo Yang, Jiahao Shao, Xinyang Li, Yujun Shen, Andreas Geiger, and Yiyi Liao. Prometheus: 3d-aware latent diffusion models for feed-forward text-to-3d scene generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2857–2869. IEEE, 2025.

Zengwei Yao, Wei Kang, Han Zhu, Liyong Guo, Lingxuan Ye, Fangjun Kuang, Weiji Zhuang, Zhaoqing Li, Zhifeng Han, Long Lin, et al. Flow2gan: Hybrid flow matching and gan with multi-resolution network for few-step high-fidelity audio generation. In *International Conference on Learning Representations*, volume 2026, pp. 747–761, 2026.

Botao Ye, Sifei Liu, Haofei Xu, Xueting Li, Marc Pollefeys, Ming-Hsuan Yang, and Songyou Peng. No pose, no problem: Surprisingly simple 3d gaussian splats from sparse unposed images. In *International Conference on Learning Representations*, volume 2025, pp. 54009–54033, 2025.

Xingyilang Yin, Qi Zhang, Jiahao Chang, Ying Feng, Qingnan Fan, Xi Yang, Chi-Man Pun, Huaqi Zhang, and Xiaodong Cun. Gsfixer: Improving 3d gaussian splatting with reference-guided video diffusion priors. *arXiv preprint arXiv:2508.09667*, 2025.

Yingji Zhong, Zhihao Li, Dave Zhenyu Chen, Lanqing Hong, and Dan Xu. Taming video diffusion prior with scene-grounding guidance for 3d gaussian splatting from sparse inputs. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6133–6143. IEEE, 2025.

Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *arXiv preprint arXiv:1805.09817*, 2018.

APPENDIX

A ARCHITECTURE DETAILS

A.1 FRAME AND GLOBAL DIT BLOCKS

The DiT in Sec. 3.3 stacks pairs of blocks, each applying frame attention followed by global attention, as in the VAE encoder transformer (Sec. 3.2). Let d_h denote the DiT hidden width. A frame block attends independently to each view's $K \times d_h$ token matrix, whereas a global block attends jointly to all N views by flattening their tokens into a single $NK \times d_h$ matrix. Cross-view exchange therefore happens in the global blocks, where noisy unobserved views gather evidence from clean observed views. Both block types use the architecture in Fig. 4, with separate parameters. For view i and token $k = 1, \ldots, K$, we denote a block's input and output hidden tokens by $\mathbf{h}^{\text{in}}_{i,k}, \mathbf{h}^{\text{out}}_{i,k} \in \mathbb{R}^{d_h}$.

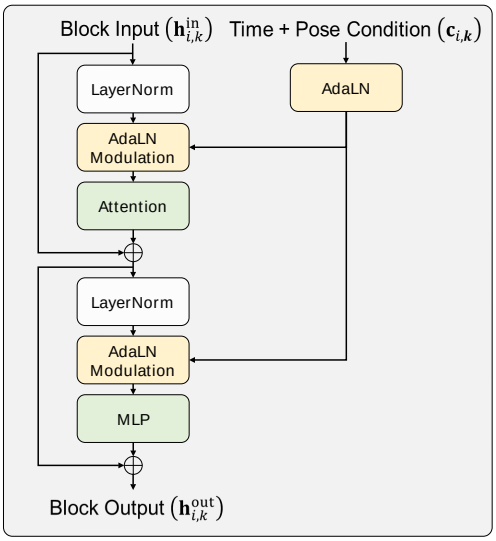

Figure 4: Architecture of a DiT block for geometry latent augmentation. Time and camera embeddings produce per-token AdaLN shift, scale, and gate parameters for the attention and MLP branches. Frame and global blocks differ only in their attention scope.

Every block is conditioned through adaptive LayerNorm (AdaLN) (Peebles & Xie, 2023). For token k of view i, the conditioning vector is $\mathbf{c}_{i,k} = \mathbf{e}^{\text{time}}(\tau_i) + \mathbf{e}^{\text{cam}}_{i,k} \in \mathbb{R}^{d_h}$. The time embedding $\mathbf{e}^{\text{time}}(\tau_i)$ is shared by all tokens of view i, with $\tau_i = t(1-m_i)$: zero for observed views and t for unobserved ones. The camera embedding $\mathbf{e}^{\text{cam}}_{i,k}$ encodes the Plücker ray through the patch center, or is a learnable embedding for tokens without a patch location, as described in Sec. 3.3.

Each block maps $\mathbf{c}_{i,k}$ to its own shift, scale, and gate vectors for the attention and MLP branches. This modulation supplies the noise level and camera information, allowing the same blocks to process clean observed views and noisy unobserved views. After the last global block, a final LayerNorm modulated by the same conditioning vectors and a linear projection back to d_z dimensions produce the predicted flow velocity $\widehat{\mathbf{v}}_{\theta,i} \in \mathbb{R}^{K \times d_z}$ for each view i.

A.2 APPEARANCE FEATURE FUSION FOR THE GAUSSIAN HEAD

Some feed-forward backbones, including AnySplat, need an appearance input in addition to the geometry tokens. Their Gaussian reconstruction head $\mathcal{H}_{\text{G}}$ embeds each view's RGB image with a convolutional image embedding layer h_{img} and adds the result to its intermediate features, supplying high-frequency appearance for the Gaussian colors. This assumes an image for every view. In our setting, however, the $(N - n)$ unobserved views have completed geometry tokens but no RGB images at inference (Sec. 3.3). We therefore supply their missing appearance input by warping the n observed images into these views.

Warped color prior. The pretrained camera and depth heads decode $(\tilde{\mathbf{F}}_1, \ldots, \tilde{\mathbf{F}}_L)$ into cameras and depth maps for all N views. We denote these outputs by $\hat{\mathbf{g}}^{\text{dec}}_i$ and $\hat{\mathbf{D}}_i$, distinguishing the decoded cameras from the cameras that condition the DiT. For an unobserved view i with $m_i = 0$, each pixel $\mathbf{u}$ is first unprojected using its predicted camera and depth to a 3D point $\mathbf{y}_i(\mathbf{u})$. We then reproject this point into an observed view c with $m_c = 1$ to obtain the warped color $\mathcal{W}_{c \to i}(\mathbf{u}) = I_c\big(\pi_c(\mathbf{y}_i(\mathbf{u}))\big)$, where π_c is the projection defined by $\hat{\mathbf{g}}^{\text{dec}}_c$. A binary indicator $\nu_{c \to i}(\mathbf{u})$ accepts a sample only if the reprojection falls inside view c with positive depth and agrees, up to a relative tolerance, with

Table 4: Shared VAE and DiT architecture for all three backbones. Each pair contains a frame block followed by a global block.

Setting	VAE encoder	VAE decoder	DiT
Hidden channels	512	512	768
Frame/global block pairs	6	6	12
Total transformer blocks	12	12	24
Attention heads	8	8	12

$\hat{\mathbf{D}}_c$, rejecting occluded samples. Averaging the accepted colors gives a per-pixel color prior and its validity mask,

$$\begin{aligned} \mathbf{C}_i^{\text{prior}}(\mathbf{u}) &= \frac{\sum_{c:m_c=1} \nu_{c\to i}(\mathbf{u})\, \mathcal{W}_{c\to i}(\mathbf{u})}{\max\big(1, \sum_{c:m_c=1} \nu_{c\to i}(\mathbf{u})\big)}, \\ M_i^{\text{prior}}(\mathbf{u}) &= \mathbb{1}\Big[\textstyle\sum_{c:m_c=1} \nu_{c\to i}(\mathbf{u}) > 0\Big]. \end{aligned} \tag{6}$$

This uses only the n observed images at inference. The cameras and depths of all N views come from the completed tokens, which are themselves produced from the observed views. No unobserved RGB image is therefore needed.

Fusion and finetuning. Let J_i denote the RGB appearance input to the image embedding layer h_{img} for view i. The appearance validity mask $M_i^{\text{app}}(\mathbf{u}) \in \{0, 1\}$ equals one if J_i provides a valid RGB value at pixel $\mathbf{u}$ and zero otherwise. For an observed view ($m_i = 1$), the appearance input is $J_i = I_i$ and every pixel is valid, $M_i^{\text{app}}(\mathbf{u}) = 1$. For an unobserved view ($m_i = 0$), we use $J_i = \mathbf{C}_i^{\text{prior}}$ and $M_i^{\text{app}}(\mathbf{u}) = M_i^{\text{prior}}(\mathbf{u})$. Pixels with no valid warped color use a learned missing-view embedding $\mathbf{e}^{\text{miss}}$ in place of the image embedding. The appearance feature map is thus

$$\mathbf{F}_i^{\text{app}}(\mathbf{u}) = M_i^{\text{app}}(\mathbf{u})\, h_{\text{img}}(J_i)(\mathbf{u}) + \big(1 - M_i^{\text{app}}(\mathbf{u})\big)\, \mathbf{e}^{\text{miss}}, \tag{7}$$

where the same vector $\mathbf{e}^{\text{miss}}$ is used at every invalid pixel. The map $\mathbf{F}_i^{\text{app}}$ enters $\mathcal{H}_{\text{G}}$ where the embedded RGB image would normally be added. These holes are out of distribution for the pretrained head, which was trained on complete images, and can corrupt colors if the appearance input is left unadapted. We therefore finetune h_{img} and learn $\mathbf{e}^{\text{miss}}$ under the Stage 2 rendering losses (Appendix B), so that the Gaussian colors at holes can be inferred from the surrounding observed appearance rather than from the placeholder alone.

A.3 Model Configurations

Table 4 summarizes the VAE and DiT configurations shared across all three backbones on both datasets. The latent width is $d_z = 4$ by default and $d_z = 16$ in the latent channel ablation, with unchanged hidden widths and block counts.

Within LAM-VAE, level-adaptive fusion uses a shared projection to 64 channels and a gating MLP with hidden width 128. We extract four feature levels at zero-based encoder indices $4, 11, 17, 23$ for AnySplat and WorldMirror, and $19, 27, 33, 39$ for DA3. Each selected feature level has 2048 channels for AnySplat and WorldMirror, and 3072 channels for DA3.

B Training and Evaluation Details

Stage 1. VAE training. As described in Sec. 3.5, we train the VAE with Level-Adaptive Modulation while keeping the geometry encoder and the pretrained reconstruction heads frozen. Given the geometry tokens $(\mathbf{F}_1, \ldots, \mathbf{F}_L)$ of an input sequence, the VAE samples a latent and decodes it back into tokens at all L levels. We denote the reconstructed tokens at level ℓ by $\hat{\mathbf{F}}_\ell$. The primary objective is a level-wise normalized reconstruction error,

$$\mathcal{L}_{\text{feat}} = \frac{1}{L} \sum_{\ell=1}^{L} \Big\| \text{LN}_\ell\big(\hat{\mathbf{F}}_\ell\big) - \text{LN}_\ell\big(\mathbf{F}_\ell\big) \Big\|_2^2, \tag{8}$$

where LN_ℓ denotes LayerNorm at level ℓ. This normalization balances token magnitudes across levels. We also use the KL divergence $\mathcal{L}_{\mathrm{KL}}$ between the posterior and a standard Gaussian prior.

Token reconstruction alone, however, does not ensure that the pretrained heads can use the reconstructed tokens. We therefore pass $(\hat{\mathbf{F}}_1, \ldots, \hat{\mathbf{F}}_L)$ through the frozen heads and compare their outputs with those obtained from the original tokens $(\mathbf{F}_1, \ldots, \mathbf{F}_L)$ without the VAE. The losses $\mathcal{L}_{\mathrm{rgb}}$ and $\mathcal{L}_{\mathrm{LPIPS}}$ measure the mean squared error and LPIPS distance between the two sets of rendered images. The losses $\mathcal{L}_{\mathrm{pose}}$ and $\mathcal{L}_{\mathrm{depth}}$ match the predicted cameras and depths. Together, these losses train the VAE to preserve the reconstruction produced by the backbone. The total objective is

$$\mathcal{L}_{\mathrm{VAE}} = \lambda_{\mathrm{feat}}\mathcal{L}_{\mathrm{feat}} + \lambda_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}} + \lambda_{\mathrm{rgb}}\mathcal{L}_{\mathrm{rgb}} + \lambda_{\mathrm{perc}}\mathcal{L}_{\mathrm{LPIPS}} + \lambda_{\mathrm{pose}}\mathcal{L}_{\mathrm{pose}} + \lambda_{\mathrm{depth}}\mathcal{L}_{\mathrm{depth}}, \quad (9)$$

with $(\lambda_{\mathrm{feat}}, \lambda_{\mathrm{KL}}, \lambda_{\mathrm{rgb}}, \lambda_{\mathrm{perc}}, \lambda_{\mathrm{pose}}, \lambda_{\mathrm{depth}}) = (0.1, 10^{-3}, 1, 0.05, 10, 1)$ in our main setting. The KL weight increases linearly from 0 to 10^{-3} over the first 10,000 training steps. The KL loss uses a free-bits threshold of 0.5 nats per latent dimension.

Stage 2. DiT training. We train the DiT for geometry latent augmentation and its camera-conditioning embeddings. We also finetune the Gaussian head's image embedding layer h_{img} and learn the missing-view embedding $\mathbf{e}^{\mathrm{miss}}$, as described in Sec. A.2. As described in Sec. 3.5, the geometry encoder, VAE, and remaining pretrained head weights stay frozen. For each training scene, we draw a global time $t = \mathrm{sigmoid}(u)$ with $u \sim \mathcal{N}(0, 1)$. As in Eq. (5), observed views retain their clean latents, while unobserved views interpolate between full-view latents and noise,

$$\mathbf{x}_{t,i} = m_i\, \mathbf{z}_i^{\mathrm{obs}} + (1 - m_i)\big[(1 - t)\, \mathbf{z}_i + t\, \boldsymbol{\epsilon}_i\big], \qquad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (10)$$

where $\mathbf{z}_i$ and $\mathbf{z}_i^{\mathrm{obs}}$ are the per-view latents in $\mathbf{Z}$ and $\mathbf{Z}^{\mathrm{obs}}$, respectively. At $t = 1$, unobserved views contain pure noise and observed views remain clean, matching the starting state at inference. The primary objective is the flow-matching loss over unobserved views,

$$\mathcal{L}_{\mathrm{FM}} = \frac{1}{(N - n)\, K\, d_z} \sum_{i:m_i=0} \big\|\widehat{\mathbf{v}}_{\theta,i} - \mathbf{v}_i^\star\big\|_2^2, \qquad \mathbf{v}_i^\star = \boldsymbol{\epsilon}_i - \mathbf{z}_i, \quad (11)$$

where $\widehat{\mathbf{v}}_{\theta,i} \in \mathbb{R}^{K \times d_z}$ is the DiT's predicted velocity for view i and $\mathbf{v}_i^\star$ is its training target. The denominator averages the error over all latent entries of the $(N - n)$ unobserved views. Observed views are excluded from this loss and stay at time $\tau_i = 0$.

The velocity loss, however, measures error only in latent space. To supervise the reconstruction itself, we estimate the clean latents of each view as $\hat{\mathbf{z}}_{0,i}^{(t)} = \mathbf{x}_{t,i} - \tau_i\, \widehat{\mathbf{v}}_{\theta,i}$, with $\tau_i = t(1 - m_i)$. Here, $\hat{\mathbf{z}}_{0,i}^{(t)}$ denotes the clean latents of view i estimated from the flow state at time t. At every training step, the VAE decoder maps these estimates to completed geometry tokens $(\tilde{\mathbf{F}}_1, \ldots, \tilde{\mathbf{F}}_L)$, which are then passed through the reconstruction heads. We randomly select a set of view indices $\mathcal{U} \subseteq \{1, \ldots, N\}$ for supervision. The losses $\mathcal{L}_{\mathrm{rgb}}$ and $\mathcal{L}_{\mathrm{LPIPS}}$ compare images rendered from the resulting Gaussians at these views with their ground-truth images.

The frozen camera and depth heads applied to the original full-view tokens $(\mathbf{F}_1, \ldots, \mathbf{F}_L)$ also provide teacher predictions for the same views. The pose loss $\mathcal{L}_{\mathrm{pose}}$ is a Huber loss between the pose encodings predicted from the completed tokens and the teacher's pose encodings at every refinement iteration of the camera head. Earlier iterations receive geometrically smaller weights. The depth loss $\mathcal{L}_{\mathrm{depth}}$ is the squared depth error on pixels selected by the teacher's depth confidence. The total objective is

$$\mathcal{L}_{\mathrm{DiT}} = \lambda_{\mathrm{FM}}\mathcal{L}_{\mathrm{FM}} + \lambda_{\mathrm{rgb}}\mathcal{L}_{\mathrm{rgb}} + \lambda_{\mathrm{perc}}\mathcal{L}_{\mathrm{LPIPS}} + \lambda_{\mathrm{pose}}\mathcal{L}_{\mathrm{pose}} + \lambda_{\mathrm{depth}}\mathcal{L}_{\mathrm{depth}}, \quad (12)$$

with $(\lambda_{\mathrm{FM}}, \lambda_{\mathrm{rgb}}, \lambda_{\mathrm{perc}}, \lambda_{\mathrm{pose}}, \lambda_{\mathrm{depth}}) = (1, 1, 0.05, 10, 1)$ in our main setting.

Stage 3. One-step finetuning. Recently, one-step adaptation has been explored in conditional diffusion models (Garcia et al., 2025; He et al., 2025; Chen et al., 2026; Yao et al., 2026). Following this principle, we start from the Stage 2 checkpoint and finetune the model for one-step reconstruction by fixing the time of the unobserved views at $t = 1$. We retain the same loss terms and weights as in Eq. (12), evaluated at this fixed time. For an unobserved view, a single explicit Euler step from $t = 1$ to $t = 0$ gives the completed latents $\tilde{\mathbf{z}}_i = \boldsymbol{\epsilon}_i - \widehat{\mathbf{v}}_{\theta,i}$, where $\tilde{\mathbf{z}}_i$ denotes view i in the completed

Table 5: Training configurations for the three stages of the main AnySplat experiments.

Setting	VAE	DiT	One-step finetuning
Optimizer	AdamW	AdamW	AdamW
Maximum learning rate	2×10^{-4}	2×10^{-4}	4×10^{-5}
Minimum learning rate	10^{-6}	2×10^{-6}	2×10^{-6}
Learning-rate schedule	Cosine decay	Cosine decay	Cosine decay
Optimizer betas	$(0.9, 0.95)$	$(0.9, 0.95)$	$(0.9, 0.95)$
Weight decay	0.01	0.01	0.01
Training iterations	200,000	200,000	20,000
Batch size per GPU (sequences)	$\lfloor 16/N_{\text{vae}} \rfloor$	1	1
Mixed precision	BF16	BF16	BF16
LPIPS start iteration	0	2,000	0

tensor $\tilde{\mathbf{Z}}$. At this fixed time, the flow-matching loss is therefore equivalent to direct regression onto the full-view latents,

$$\mathcal{L}_{\text{FM}}|_{t=1} = \frac{1}{(N-n)\,K\,d_z} \sum_{i:m_i=0} \left\|\tilde{\mathbf{z}}_i - \mathbf{z}_i\right\|_2^2. \tag{13}$$

The rendering losses also supervise the one-step reconstruction end to end. Together, these losses optimize one-step reconstruction quality without imposing consistency with the Stage 2 model's multi-step ODE solution.

Training configuration. Table 5 summarizes the configuration shared by the main AnySplat experiments on RE10K and DL3DV. Learning rates are effective values for four GPUs, including the square-root scaling with global batch size used in Stages 2 and 3. Let N_{vae} denote the number of input views in a Stage 1 training sequence, which can vary during VAE training. We use $\lfloor 16/N_{\text{vae}} \rfloor$ sequences per GPU to keep at most 16 images per GPU. Stages 2 and 3 use one sequence per GPU, giving a global batch size of 4. LPIPS is enabled after the first $2{,}000$ iterations in Stage 2 and from the start in Stages 1 and 3.

Evaluation view sampling. Following NoPoSplat (Ye et al., 2025), we estimate image overlap using bidirectional RoMa (Edstedt et al., 2024) matching. The overlap between the first and last views determines the difficulty of an interval. We group intervals into three overlap ranges $(0.55, 0.80]$, $(0.30, 0.55]$, and $[0.05, 0.30]$, with lower overlap indicating less shared visual evidence. We assign these ranges to sequences with probabilities 25%, 50%, and 25%, respectively, and exclude sequences without a suitable interval in their assigned range. The resulting evaluation subsets follow these target proportions subject to the availability of eligible sequences.

Within each selected interval, we place $N = 16$ reconstruction views along the camera trajectory according to both translation and rotation, and reserve four separate novel views inside the interval for evaluation. We evaluate 80 sequences for each of RE10K and DL3DV, yielding 320 evaluation images per dataset. The $n \in \{2, 3, 5, 7\}$ observed views form nested subsets of the N reconstruction views and always include both endpoints. Increasing n therefore adds observations while retaining the existing ones. Because estimated image overlap varies with resolution, we sample views separately at each evaluation resolution.

C Additional Results

C.1 Results with DA3 Baseline

Table 6 evaluates DensiTok with Depth Anything 3 (DA3) (Lin et al., 2025) on RE10K and DL3DV using the same evaluation protocol as Table 1. DensiTok consistently improves PSNR, SSIM, and LPIPS across both datasets and all evaluated sparse-view settings. The gains in rendering quality grow as the number of observed views increases.

Depth estimation also improves consistently on both datasets at every evaluated input count. Camera AUC improves at both angular thresholds on RE10K and at the stricter threshold on DL3DV. At the

Table 6: Quantitative results for DA3 on RealEstate10K and DL3DV with $n \in \{2, 3, 5, 7\}$ sparse input views and dense-view references ($N = 16$). Camera AUC is reported in percent. Runtime is reported in seconds in a shared column.

Methods	RealEstate10K (Zhou et al., 2018)							DL3DV (Ling et al., 2024)							Runtime
	Appearance			Camera		Depth		Appearance			Camera		Depth		
	PSNR↑	SSIM↑	LPIPS↓	AUC@3° ↑	AUC@30° ↑	$\delta_{1.25}$ ↑	AbsRel↓	PSNR↑	SSIM↑	LPIPS↓	AUC@3° ↑	AUC@30° ↑	$\delta_{1.25}$ ↑	AbsRel↓	(s)↓
Dense-view ($N = 16$)															
DA3 (Lin et al., 2025)	21.28	0.712	0.249	43.8	86.2	0.538	0.346	22.73	0.717	0.222	87.8	97.2	0.556	0.322	0.514
Sparse-view ($n = 2$)															
DA3	19.69	0.667	0.300	55.8	89.6	0.369	0.471	20.86	0.646	0.287	88.3	**98.8**	0.402	0.452	0.112
+ DensiTok	**22.95**	**0.791**	**0.213**	**61.7**	**93.2**	**0.523**	**0.409**	**22.16**	**0.707**	**0.256**	**88.8**	**98.8**	**0.525**	**0.387**	0.255
Sparse-view ($n = 3$)															
DA3	20.50	0.696	0.268	52.2	89.0	0.453	0.405	21.65	0.674	0.256	88.8	97.9	0.458	0.392	0.135
+ DensiTok	**25.01**	**0.841**	**0.168**	**64.9**	**90.4**	**0.638**	**0.342**	**24.23**	**0.781**	**0.196**	**88.9**	**98.0**	**0.607**	**0.330**	0.308
Sparse-view ($n = 5$)															
DA3	20.81	0.709	0.255	49.5	87.6	0.480	0.387	22.12	0.690	0.242	86.4	**96.9**	0.485	0.363	0.187
+ DensiTok	**26.58**	**0.876**	**0.140**	**61.6**	**90.1**	**0.739**	**0.256**	**25.62**	**0.824**	**0.163**	**86.6**	96.7	**0.684**	**0.278**	0.333
Sparse-view ($n = 7$)															
DA3	21.08	0.714	0.248	47.9	87.4	0.500	0.382	22.40	0.703	0.233	85.0	**96.2**	0.496	0.350	0.244
+ DensiTok	**27.19**	**0.889**	**0.127**	**51.1**	**88.5**	**0.770**	**0.212**	**26.26**	**0.847**	**0.147**	**85.3**	**96.2**	**0.717**	**0.250**	0.367

larger threshold on DL3DV, results remain close to the baseline, with some settings showing no change or a slight decrease.

Even with sparse inputs, DensiTok can surpass the dense-view DA3 reference in all appearance metrics on both datasets. At the larger evaluated input counts, it also exceeds the reference in both depth metrics on both datasets. In these settings, camera AUC is higher than the reference on RE10K but remains lower on DL3DV. DensiTok increases runtime relative to sparse-view DA3, while remaining faster than the dense-view reference across all evaluated settings.

C.2 Additional Qualitative Results

Figures 5 and 6 present additional novel-view RGB and depth comparisons on RE10K and DL3DV for AnySplat, WorldMirror, and DA3, each with and without DensiTok. Figure 5 shows results with $n = 2$ and $n = 3$ input views, while Figure 6 covers $n = 5$ and $n = 7$. These examples extend the qualitative comparison in the main paper to all three backbones and all evaluated input counts.

With only two or three observed views, DensiTok reduces RGB blur and produces more coherent depth regions in the highlighted examples, including clearer object contours around monitors and train seats. The benefits remain visible with five or seven views, where the crops show sharper image details and reduced local noise in the depth predictions. These changes are particularly visible in the architectural and furniture examples with DA3.

C.3 Visualization of Gaussian Reconstructions and Camera Poses

Figure 7 compares the Gaussian scenes and camera poses predicted by AnySplat with and without DensiTok from the same sparse input views. The Gaussian visualizations show that DensiTok reconstructs more coherent scene structure with clearer object boundaries.

The camera visualizations show improvements in both camera positions and viewing orientations. In the illustrated examples, DensiTok's predicted camera centers lie closer to the ground-truth centers, and the predicted camera frusta align more closely with their ground-truth counterparts. The reduced positional displacement and angular mismatch are particularly visible in the outdoor example. These observations are consistent with the camera AUC improvements reported in Table 1.

The joint improvement in scene reconstruction and camera estimation reflects their dependence on a shared geometry representation. As described in Section A.2, the pretrained camera head decodes the final camera poses from the completed geometry tokens. The predicted and interpolated cameras in Section 3.4 provide conditions for latent generation, while the final output cameras are predicted again from the enriched tokens. Geometry token completion therefore benefits both Gaussian recon-

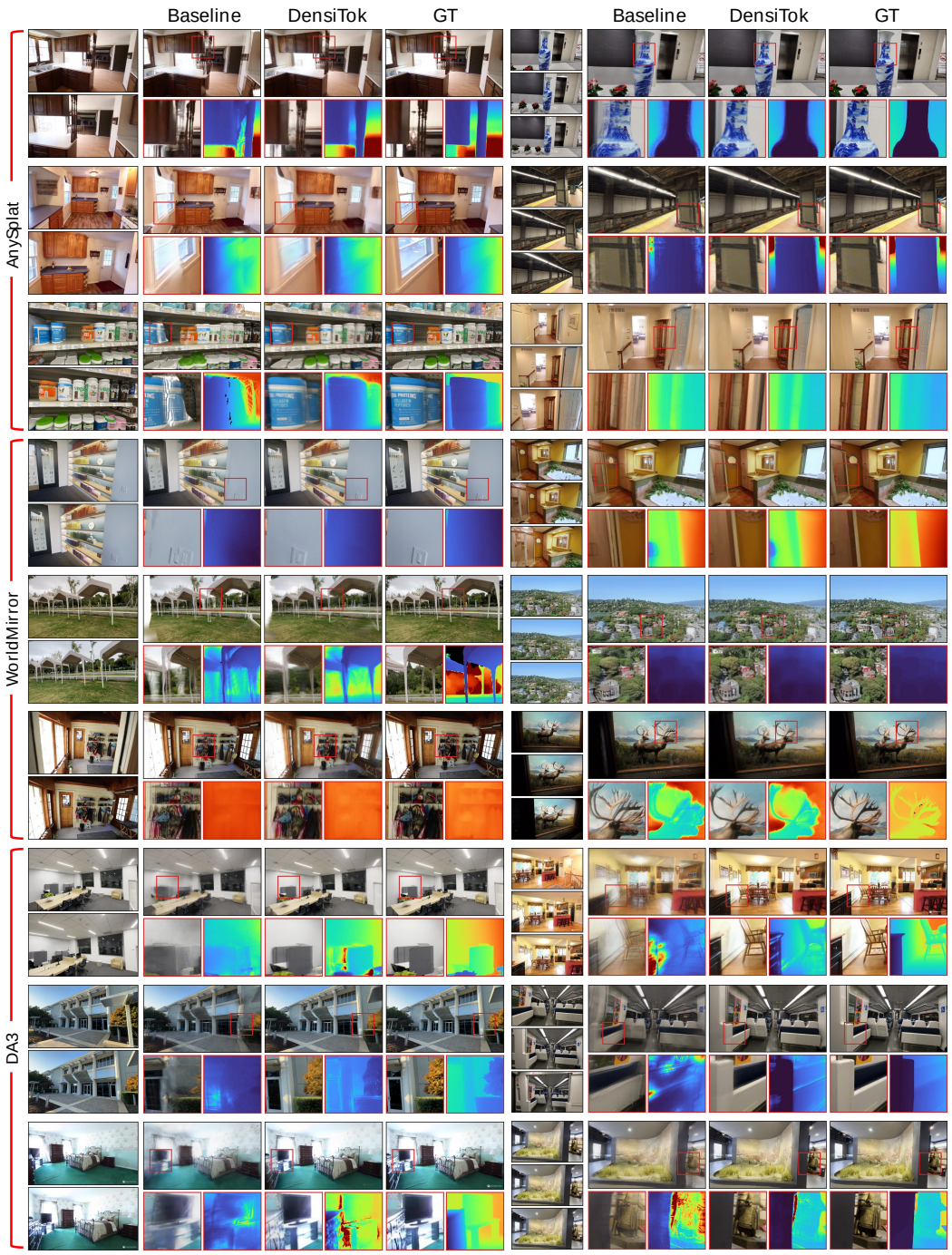

Figure 5: Qualitative novel-view RGB and depth comparisons on RE10K and DL3DV with $n = 2$ input views (left) and $n = 3$ input views (right), each with and without DensiTok. Red boxes mark the enlarged RGB and depth crops. GT denotes the ground-truth target image.

struction and camera localization and orientation, even though the pretrained camera head remains frozen.

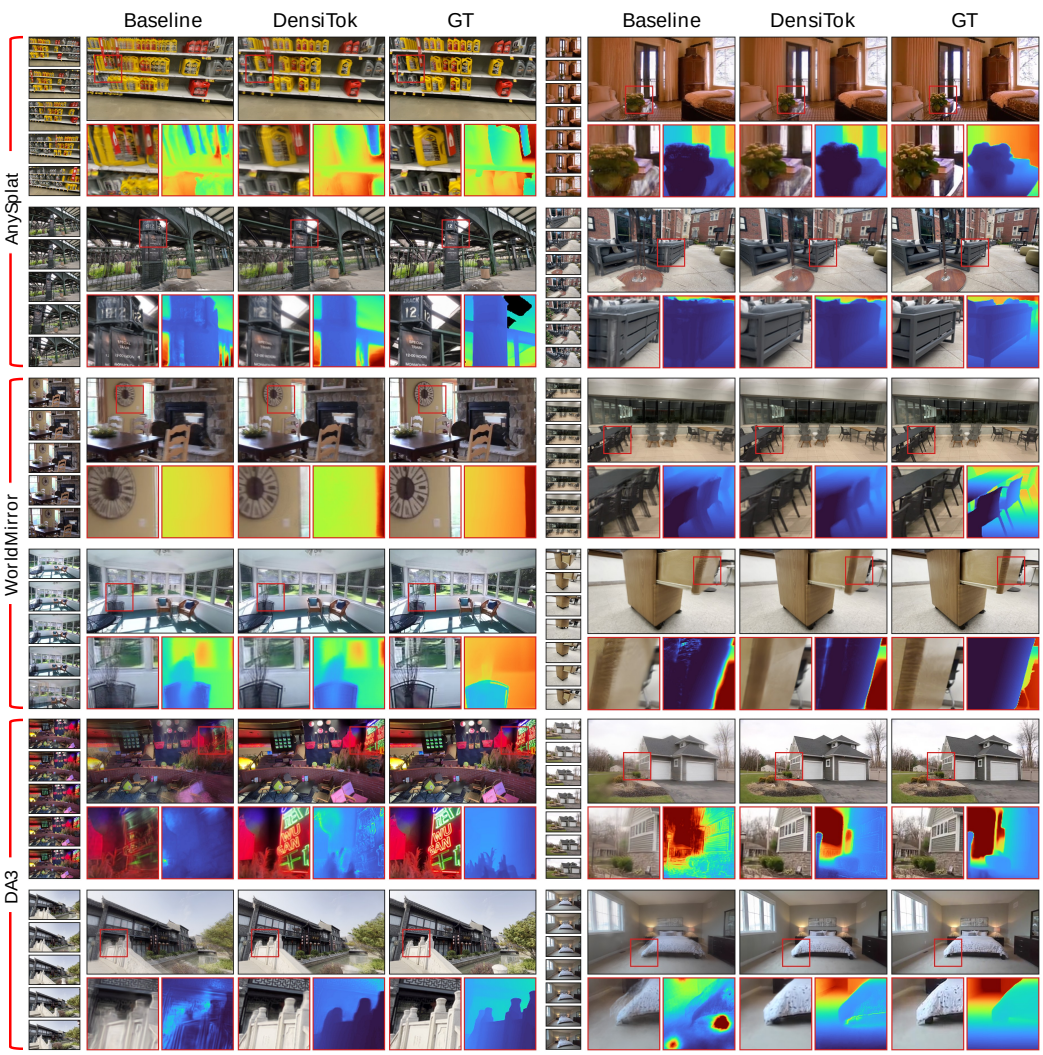

Figure 6: Qualitative novel-view RGB and depth comparisons on RE10K and DL3DV with $n = 5$ input views (left) and $n = 7$ input views (right), each with and without DensiTok. Red boxes mark the enlarged RGB and depth crops. GT denotes the ground-truth target image.

C.4 Comparison with Sparse-View Feed-Forward Gaussian Splatting Models

Evaluation protocol. We compare DensiTok with existing feed-forward Gaussian Splatting models designed for sparse-view reconstruction, including pixelSplat (Charatan et al., 2024), MVSplat (Chen et al., 2024a), DepthSplat (Xu et al., 2025a), VolSplat (Wang et al., 2025e), SelfSplat (Kang et al., 2025), PF3plat (Hong et al., 2025), SPFSplat (Huang & Mikolajczyk, 2025), NoPoSplat (Ye et al., 2025), Splatt3R (Smart et al., 2024), and FreeSplatter (Xu et al., 2025b) on RE10K and DL3DV. Most existing sparse-view methods have been evaluated at 256×256 resolution (Charatan et al., 2024; Chen et al., 2024a; Xu et al., 2025a; Hong et al., 2025), so we evaluate all methods, including DensiTok, at this resolution. For DensiTok, which requires input image dimensions divisible by 14, we perform inference at 252×252 and resize the predictions to the 256×256 evaluation resolution. Because this resolution differs from that of our original evaluation, we perform test-view sampling again. The resulting sampling indices differ from those used in Tables 1 and 6.

We consider $n \in \{2, 3, 5, 7\}$ and evaluate each model only at input image counts supported by its official implementation. Table 7 lists the evaluated method and input-count combinations. Unlisted

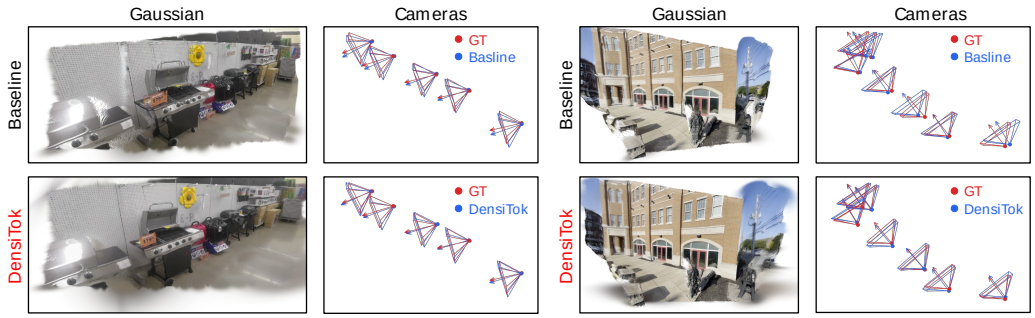

Figure 7: Visualization of Gaussian reconstructions and camera poses for AnySplat with and without DensiTok. The top row shows the original AnySplat backbone and the bottom row shows DensiTok. Each scene is paired with a camera visualization, where red denotes ground truth and blue denotes predictions. DensiTok produces clearer scene structure and closer agreement with ground-truth camera positions and viewing orientations.

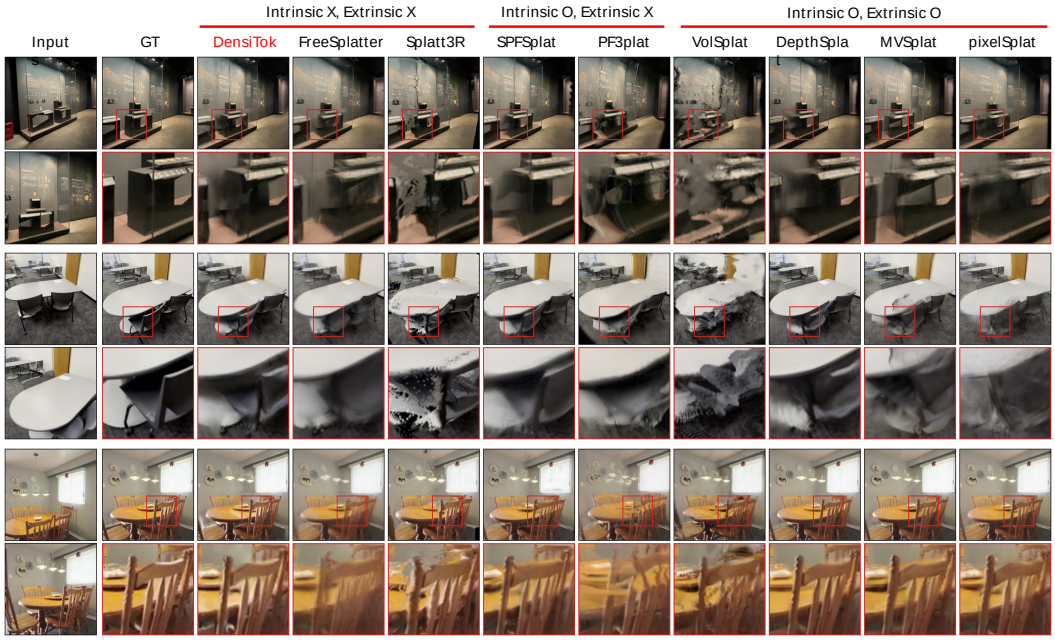

Figure 8: Qualitative novel-view RGB comparisons with sparse-view feed-forward Gaussian Splatting models. DensiTok uses the AnySplat backbone and requires neither input camera intrinsics nor extrinsics. Methods are grouped by the input camera information they require. The leftmost column shows two input views for each scene, and GT denotes the ground-truth target image. Red boxes mark regions enlarged below each rendering.

combinations were not evaluated. The table reports PSNR, SSIM, LPIPS, depth accuracy $\delta_{1.25}$, AbsRel, and runtime, using the metric notation in Section 4.1.

Camera inputs. As shown in the Table 7, the Intr. and Extr. columns distinguish externally supplied input camera intrinsics and extrinsics required for reconstruction. These labels describe the evaluated configurations and exclude target cameras used only for rendering and evaluation. pixelSplat, MVSplat, and DepthSplat use both intrinsics and extrinsics. SelfSplat, PF3plat, and the evaluated SPFSplat configuration use intrinsics but estimate poses. NoPoSplat also uses input intrinsics but does not require input extrinsics. Splatt3R, FreeSplatter, and all three DensiTok variants require neither input intrinsics nor input extrinsics. This distinction makes the different camera information available to the compared methods explicit.

Table 7: Comparison with sparse-view feed-forward Gaussian Splatting methods on RE10K and DL3DV at 256×256 with $n \in \{2, 3, 5, 7\}$. NoPoSplat results are included for $n \in \{2, 3\}$. Intr. and Extr. indicate whether the evaluated configuration requires externally supplied input camera intrinsics and extrinsics for reconstruction. Orange and yellow mark the best and second-best values for each metric at the same n, respectively. Rankings are determined before rounding, with ties sharing the same color.

			RE10K					DL3DV					
	Input cameras		Appearance			Depth		Appearance			Depth		
Method	Intr.	Extr.	PSNR ↑	SSIM ↑	LPIPS ↓	$\delta_{1.25}$ ↑	AbsRel ↓	PSNR ↑	SSIM ↑	LPIPS ↓	$\delta_{1.25}$ ↑	AbsRel ↓	Runtime (s)↓
Sparse-view ($n = 2$)													
pixelSplat	○	○	26.125	0.864	0.131	0.608	0.359	23.358	0.721	0.239	0.489	0.467	0.088
MVSplat	○	○	26.319	0.865	0.127	0.769	0.220	22.894	0.704	0.235	0.687	0.275	0.039
DepthSplat	○	○	26.636	0.881	0.117	0.783	0.193	24.078	0.759	0.196	0.765	0.193	0.067
VolSplat	○	○	21.730	0.787	0.217	0.740	0.223	17.911	0.555	0.419	0.584	0.307	0.093
PF3plat	○	×	20.011	0.638	0.261	0.806	0.173	17.743	0.500	0.348	0.710	0.231	0.327
SPFSplat	○	×	21.850	0.707	0.178	0.814	0.174	20.892	0.610	0.238	0.799	0.170	0.053
NoPoSplat	○	×	22.470	0.721	0.161	0.836	0.159	21.905	0.665	0.208	0.806	0.159	0.052
Splatt3R	×	×	20.731	0.688	0.272	0.835	0.131	19.635	0.622	0.320	0.818	0.140	0.811
FreeSplatter	×	×	22.973	0.739	0.206	0.809	0.165	22.097	0.642	0.271	0.734	0.222	0.201
DensiTok (AnySplat)	×	×	24.628	0.830	0.158	0.864	0.127	22.111	0.711	0.218	0.798	0.167	0.071
DensiTok (WorldMirror)	×	×	25.079	0.838	0.162	0.876	0.108	24.056	0.770	0.202	0.807	0.151	0.071
DensiTok (DA3)	×	×	24.233	0.818	0.166	0.846	0.137	21.816	0.702	0.226	0.780	0.180	0.081
Sparse-view ($n = 3$)													
pixelSplat	○	○	28.184	0.902	0.102	0.692	0.287	25.721	0.813	0.167	0.564	0.366	0.138
MVSplat	○	○	24.644	0.866	0.136	0.658	0.329	22.184	0.728	0.228	0.586	0.400	0.054
DepthSplat	○	○	26.399	0.901	0.105	0.786	0.182	24.425	0.819	0.155	0.793	0.166	0.092
VolSplat	○	○	25.532	0.871	0.144	0.782	0.193	21.960	0.676	0.322	0.715	0.223	0.131
SelfSplat	○	×	22.001	0.732	0.231	0.723	0.225	16.440	0.446	0.478	0.607	0.279	0.057
PF3plat	○	×	20.976	0.706	0.228	0.822	0.166	18.114	0.519	0.337	0.715	0.230	0.505
SPFSplat	○	×	23.402	0.768	0.157	0.802	0.179	20.915	0.621	0.255	0.754	0.201	0.059
NoPoSplat	○	×	22.298	0.702	0.152	0.833	0.158	20.889	0.588	0.228	0.804	0.165	0.060
FreeSplatter	×	×	20.401	0.639	0.308	0.767	0.184	20.147	0.560	0.349	0.702	0.240	0.238
DensiTok (AnySplat)	×	×	26.919	0.873	0.124	0.866	0.123	24.948	0.800	0.154	0.824	0.148	0.086
DensiTok (WorldMirror)	×	×	26.818	0.869	0.137	0.888	0.100	26.013	0.830	0.154	0.832	0.134	0.090
DensiTok (DA3)	×	×	26.634	0.864	0.133	0.848	0.136	24.716	0.793	0.161	0.811	0.157	0.098
Sparse-view ($n = 5$)													
MVSplat	○	○	22.219	0.836	0.166	0.517	0.497	20.741	0.724	0.243	0.489	0.528	0.093
DepthSplat	○	○	23.940	0.876	0.128	0.799	0.176	23.029	0.835	0.154	0.810	0.154	0.131
VolSplat	○	○	27.404	0.903	0.117	0.798	0.183	24.254	0.760	0.248	0.790	0.174	0.188
PF3plat	○	×	20.961	0.706	0.229	0.823	0.166	17.928	0.514	0.341	0.710	0.227	1.096
SPFSplat	○	×	24.062	0.787	0.141	0.806	0.177	21.362	0.638	0.239	0.754	0.198	0.072
FreeSplatter	×	×	19.644	0.611	0.354	0.760	0.198	19.627	0.538	0.384	0.697	0.249	0.487
DensiTok (AnySplat)	×	×	28.034	0.894	0.108	0.866	0.121	26.795	0.844	0.123	0.830	0.141	0.129
DensiTok (WorldMirror)	×	×	27.644	0.886	0.125	0.894	0.094	27.154	0.860	0.130	0.842	0.124	0.136
DensiTok (DA3)	×	×	27.713	0.886	0.114	0.847	0.132	26.540	0.836	0.130	0.818	0.149	0.145
Sparse-view ($n = 7$)													
MVSplat	○	○	21.244	0.820	0.181	0.530	0.488	20.080	0.717	0.252	0.497	0.508	0.147
DepthSplat	○	○	22.742	0.856	0.145	0.791	0.179	21.905	0.815	0.173	0.796	0.157	0.177
VolSplat	○	○	28.328	0.909	0.113	0.801	0.179	25.007	0.783	0.227	0.799	0.163	0.256
PF3plat	○	×	20.888	0.699	0.232	0.821	0.167	17.834	0.510	0.344	0.708	0.232	1.899
SPFSplat	○	×	24.478	0.798	0.132	0.807	0.176	21.515	0.642	0.233	0.755	0.200	0.096
FreeSplatter	×	×	19.203	0.596	0.378	0.748	0.210	19.173	0.521	0.408	0.680	0.257	0.760
DensiTok (AnySplat)	×	×	28.522	0.903	0.101	0.865	0.120	27.627	0.865	0.109	0.835	0.139	0.172
DensiTok (WorldMirror)	×	×	28.087	0.896	0.117	0.898	0.092	27.693	0.876	0.120	0.850	0.121	0.186
DensiTok (DA3)	×	×	28.225	0.895	0.108	0.851	0.129	27.463	0.861	0.114	0.821	0.147	0.193

Comparison results. Most sparse-view methods report results using only two or three input views, while our protocol keeps the sampled evaluation views fixed across different input view counts. This difference in evaluation sampling can lead to slight discrepancies with the results reported in the original papers.

Table 8: Comparison with diffusion-based novel-view synthesis methods on RE10K and DL3DV at 512×512 with $n \in \{2, 3, 5, 7\}$. Runtime is reported in seconds in a shared column. Orange and yellow mark the best and second-best values for each metric at the same n, respectively. Rankings are determined before rounding, with ties sharing the same color.

Method	RE10K			DL3DV			
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Runtime (s)↓
Sparse-view ($n = 2$)							
NVComposer	14.897	0.528	0.508	12.078	0.359	0.610	48.502
Matrix3D	20.934	0.700	0.284	18.149	0.523	0.376	25.592
GLD	23.608	0.768	0.219	21.440	0.621	0.270	76.824
DensiTok (AnySplat)	24.435	0.825	0.208	21.500	0.685	0.270	0.510
DensiTok (WorldMirror)	24.882	0.834	0.213	23.448	0.744	0.275	0.515
DensiTok (DA3)	24.598	0.827	0.226	22.503	0.709	0.285	0.579
Sparse-view ($n = 3$)							
NVComposer	15.518	0.545	0.482	12.254	0.362	0.600	48.486
Matrix3D	21.996	0.725	0.259	18.954	0.545	0.347	29.415
GLD	25.130	0.796	0.192	22.771	0.656	0.233	90.549
DensiTok (AnySplat)	26.826	0.867	0.170	24.212	0.764	0.206	0.531
DensiTok (WorldMirror)	26.720	0.863	0.183	25.276	0.793	0.211	0.553
DensiTok (DA3)	26.985	0.872	0.180	24.452	0.776	0.222	0.605
Sparse-view ($n = 5$)							
Matrix3D	22.684	0.738	0.246	19.480	0.557	0.333	51.567
GLD	26.100	0.813	0.176	23.446	0.677	0.214	118.995
DensiTok (AnySplat)	27.995	0.888	0.152	26.041	0.807	0.172	0.593
DensiTok (WorldMirror)	27.610	0.880	0.168	26.396	0.822	0.179	0.627
DensiTok (DA3)	28.133	0.891	0.163	25.717	0.812	0.192	0.668
Sparse-view ($n = 7$)							
Matrix3D	22.993	0.750	0.236	19.854	0.569	0.318	110.844
GLD	26.918	0.834	0.156	23.919	0.689	0.203	150.433
DensiTok (AnySplat)	28.527	0.897	0.143	26.841	0.826	0.157	0.656
DensiTok (WorldMirror)	28.087	0.890	0.153	26.903	0.837	0.168	0.713
DensiTok (DA3)	28.762	0.905	0.149	26.544	0.833	0.174	0.740

Table 7 shows that all three DensiTok variants outperform FreeSplatter in PSNR, SSIM, and LPIPS on both datasets at all evaluated input counts, except for the DA3 variant in PSNR on DL3DV at $n = 2$. None of the DensiTok variants requires input camera intrinsics or extrinsics. At $n = 3$, all three variants also outperform NoPoSplat in every reported appearance and depth metric on both datasets. Among all evaluated methods, DepthSplat achieves the highest PSNR and lowest LPIPS on both datasets at $n = 2$. DensiTok with AnySplat achieves the highest PSNR and lowest LPIPS on RE10K at $n \in \{5, 7\}$, while DensiTok with WorldMirror achieves the highest PSNR and SSIM on DL3DV at $n \in \{3, 5, 7\}$. The DA3 variant achieves the second-highest PSNR on RE10K at $n = 5$ and the second-lowest LPIPS on RE10K at $n \in \{5, 7\}$ and on DL3DV at $n = 7$.

DensiTok with WorldMirror also achieves the best values for both depth metrics on RE10K at every evaluated input count and on DL3DV at $n \in \{3, 5, 7\}$. Splatt3R remains best on both DL3DV depth metrics at $n = 2$. All three DensiTok variants outperform FreeSplatter in both depth metrics on both datasets at $n \in \{2, 3, 5, 7\}$.

The qualitative comparison in Figure 8 further illustrates the reconstruction quality of DensiTok with the AnySplat backbone. Among methods that require neither input camera intrinsics nor extrinsics,

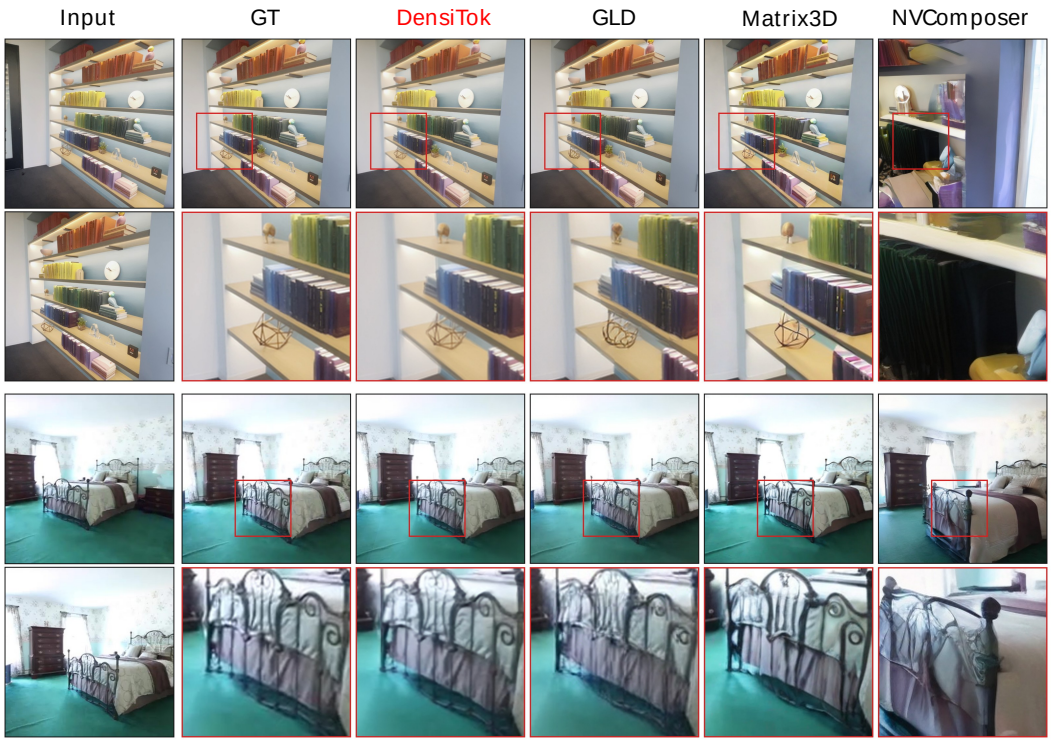

Figure 9: Qualitative novel-view RGB comparisons of DensiTok with GLD, Matrix3D, and NVComposer using $n = 2$ input views. The leftmost column shows the input views for each scene, and GT denotes the ground-truth target image. Red boxes mark regions enlarged below each rendering to compare the preservation of scene content and local geometry.

DensiTok produces clearer object boundaries and fewer structural artifacts than FreeSplatter and Splatt3R in the illustrated examples. DensiTok also achieves better or comparable visual quality relative to the methods that require input intrinsics, with or without extrinsics. These examples show that DensiTok can preserve fine scene details without relying on externally supplied input camera information.

C.5 Comparison with Diffusion-Based Novel View Synthesis Models

Evaluation protocol. We compare DensiTok with NVComposer (Li et al., 2025), Matrix3D (Lu et al., 2025), and GLD (Jang et al., 2026) on RE10K and DL3DV. These methods use diffusion for novel-view image generation, with GLD operating in geometric feature space. All methods are evaluated at 512×512 resolution. GLD and DensiTok require input image dimensions divisible by 14, so we perform inference at 504×504 and resize the predicted images to 512×512 before evaluation, following the resizing procedure in the preceding comparison. Table 8 reports PSNR, SSIM, LPIPS, and runtime for the evaluated method and input-count combinations with $n \in \{2, 3, 5, 7\}$.

Comparison results. Most diffusion-based NVS methods aim to synthesize plausible content in unobserved regions and are typically evaluated with widely spaced input views. Our reconstruction task requires a minimum degree of overlap between input views, so we use more closely spaced inputs. This difference in view sampling can lead to modest discrepancies with the results reported in the original papers.

Table 8 shows that all three DensiTok variants achieve higher PSNR and SSIM than the evaluated diffusion baselines on both datasets at every matching input count. DensiTok with DA3 achieves the highest PSNR and SSIM on RE10K for $n \in \{3, 5, 7\}$, while the WorldMirror variant leads these metrics on DL3DV at every evaluated input count. The AnySplat variant achieves the lowest LPIPS on both datasets across all input counts. All three variants also outperform the diffusion

Table 9: Generalization to ETH3D, ScanNet, and Mip-NeRF 360 with the AnySplat backbone and $n \in \{2, 3, 5, 7\}$ observed views. Camera AUC is reported in percent. Depth is evaluated against official ground truth on ETH3D and ScanNet and MoGe-2 predictions on Mip-NeRF 360.

n	Methods	Appearance			Camera		Depth	
		PSNR↑	SSIM↑	LPIPS↓	AUC@3° ↑	AUC@30° ↑	$\delta_{1.25}$ ↑	AbsRel↓
ETH3D (Schops et al., 2017)								
2	AnySplat	15.99	0.462	0.489	27.3	89.4	0.767	0.148
	+ DensiTok	**17.01**	**0.531**	**0.466**	**33.3**	**90.3**	**0.771**	**0.135**
3	AnySplat	16.79	0.487	0.461	29.3	88.0	0.831	0.119
	+ DensiTok	**17.92**	**0.554**	**0.447**	**31.3**	**89.2**	**0.837**	**0.116**
5	AnySplat	16.53	0.481	0.473	41.2	90.6	0.859	0.114
	+ DensiTok	**17.99**	**0.565**	**0.437**	**41.8**	**90.8**	**0.869**	**0.099**
7	AnySplat	16.90	0.487	0.467	39.5	89.7	0.874	0.101
	+ DensiTok	**18.09**	**0.580**	**0.415**	**41.6**	**91.1**	**0.889**	**0.092**
ScanNet (Dai et al., 2017)								
2	AnySplat	19.60	0.668	0.374	13.8	77.5	0.952	0.070
	+ DensiTok	**22.51**	**0.761**	**0.319**	**14.7**	**77.6**	**0.972**	**0.052**
3	AnySplat	21.63	0.714	0.332	13.9	76.6	0.973	0.057
	+ DensiTok	**24.18**	**0.803**	**0.278**	**16.3**	**78.2**	**0.981**	**0.048**
5	AnySplat	22.68	0.740	0.314	15.4	75.6	0.977	0.053
	+ DensiTok	**24.76**	**0.822**	**0.264**	**17.1**	**76.4**	**0.981**	**0.047**
7	AnySplat	23.25	0.755	0.304	14.5	74.3	0.979	0.052
	+ DensiTok	**25.20**	**0.833**	**0.252**	**15.5**	**74.9**	**0.982**	**0.046**
Mip-NeRF 360 (Barron et al., 2022)								
2	AnySplat	17.58	0.428	0.395	**14.8**	73.7	0.779	0.168
	+ DensiTok	**19.27**	**0.490**	**0.387**	**14.8**	**82.6**	**0.814**	**0.155**
3	AnySplat	18.05	0.450	**0.383**	11.1	78.6	0.796	0.168
	+ DensiTok	**19.24**	**0.469**	0.400	**12.3**	**83.6**	**0.812**	**0.159**
5	AnySplat	18.73	0.470	0.383	11.5	82.1	0.794	0.166
	+ DensiTok	**20.02**	**0.498**	**0.371**	**13.0**	**83.6**	**0.829**	**0.153**
7	AnySplat	19.76	0.501	0.358	15.5	84.8	0.809	0.160
	+ DensiTok	**20.48**	**0.521**	**0.349**	**16.0**	**86.3**	**0.834**	**0.151**

baselines in LPIPS for $n \in \{3, 5, 7\}$. At $n = 2$, GLD retains lower LPIPS than the DA3 variant on both datasets and the WorldMirror variant on DL3DV. The qualitative comparison in Figure 9 shows that the evaluated diffusion baselines can alter scene context and exhibit weaker geometric consistency with the observed scene. NVComposer changes the bookshelf composition and bed appearance, producing plausible content that departs from the target scene. GLD and Matrix3D better retain the overall layout, but the enlarged shelf ornament and bed-frame regions reveal local shape distortions. In these examples, DensiTok more closely preserves the scene content, object arrangement, and geometric structure of the ground-truth targets. DensiTok completes inference in under a second, while the evaluated diffusion baselines require tens of seconds to a few minutes. This yields speedups of tens to hundreds of times at matching input counts. At $n = 2$, DensiTok requires 0.510 to 0.579 seconds compared with 76.824 seconds for GLD, yielding approximately $133\times$ to $151\times$ speedups across the three backbones. DensiTok therefore combines stronger image reconstruction metrics with substantially lower latency, making it more practical for fast novel-view synthesis.

Table 10: Effect of one-step finetuning with AnySplat, $d_z = 4$, $n = 2$, and $N = 16$. FM denotes flow matching and NFE denotes the number of DiT forward evaluations. Runtime is reported in a shared column.

		RE10K			DL3DV			Efficiency
Method	NFE ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Runtime (s)↓
Deterministic regression	1	20.86	0.712	0.246	19.97	0.631	0.288	0.258
One-step training from scratch	1	21.67	0.746	0.240	21.28	0.670	0.285	0.258
FM pretraining only	1	21.56	0.743	0.243	21.09	0.665	0.289	0.258
	8	21.84	0.755	0.236	21.59	0.676	0.279	0.990
	50	22.86	0.773	0.217	22.81	0.709	0.246	4.848
FM + one-step finetuning	1	22.56	0.766	0.223	22.40	0.699	0.255	0.258

D Additional Ablation Studies

In this section, we use AnySplat as a common reconstruction backbone for all experiments, following the ablation studies in the main paper.

D.1 Generalization to Unseen Datasets

Evaluation protocol. We evaluate generalization on ETH3D (Schops et al., 2017), ScanNet (Dai et al., 2017), and Mip-NeRF 360 (Barron et al., 2022), none of which is used to train DensiTok. We compare AnySplat with and without DensiTok using $n \in \{2, 3, 5, 7\}$ observed views. We use the official evaluation sequences for each dataset, comprising 11 sequences for ETH3D, 75 for ScanNet, and 9 for Mip-NeRF 360. We evaluate four novel views per sequence on ScanNet. For ETH3D and Mip-NeRF 360, we reduce the number of evaluation views to one per sequence because fewer views are available in each sequence.

For depth evaluation, we use the official ground-truth depth provided by ETH3D and ScanNet. Since Mip-NeRF 360 does not provide ground-truth depth, we use MoGe-2 (Wang et al., 2025d) predictions as pseudo-labels, following the main evaluation.

Generalization results. Table 9 shows consistent improvements across the three datasets, with PSNR typically increasing by around one to three decibels. PSNR gains are largest on ScanNet, while ETH3D and Mip-NeRF 360 also benefit across input counts. Camera and depth accuracy improve throughout, indicating that the gains extend beyond image reconstruction. These results support the ability of geometry token completion to improve reconstruction on datasets excluded from DensiTok training.

D.2 Effect of flow-matching pretraining and one-step finetuning.

We evaluate whether one-step finetuning preserves the reconstruction quality of iterative sampling while reducing inference cost. We also compare against deterministic regression and one-step training from scratch to examine the benefit of flow-matching pretraining. Table 10 reports these results with the AnySplat backbone, $d_z = 4$, $n = 2$, and $N = 16$, including the Stage 2 checkpoint evaluated at different NFEs and its finetuned counterpart.

Both baselines retain the camera-conditioned DiT architecture, pretrained backbone, and Stage 1 LAM-VAE, but train the DiT from random initialization without flow-matching pretraining. Let c denote the observed latents $\mathbf{Z}^{\text{obs}}$, view mask $\mathbf{m}$, and cameras $\{\mathbf{g}_i\}_{i=1}^N$. *Deterministic regression* uses zero inputs for unobserved views and directly predicts $\tilde{\mathbf{z}}_i = f_{\theta,i}(\mathbf{0}, c)$. *One-step training from scratch* instead fixes the time of the unobserved views at $t = 1$ and predicts $\tilde{\mathbf{z}}_i = \boldsymbol{\epsilon}_i - \widehat{\mathbf{v}}_{\theta,i}$ with $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, following the Stage 3 parameterization. Both baselines use the same loss terms as in Stage 3.

Table 11: Effect of latent channel count with AnySplat, $N = 16$, and one-step inference.

		RE10K							DL3DV						
		Appearance			Camera		Depth		Appearance			Camera		Depth	
n	d_z	PSNR↑	SSIM↑	LPIPS↓	AUC@3° ↑	AUC@30° ↑	$\delta_{1.25}$ ↑	AbsRel↓	PSNR↑	SSIM↑	LPIPS↓	AUC@3° ↑	AUC@30° ↑	$\delta_{1.25}$ ↑	AbsRel↓
2	4	22.56	0.766	0.223	26.7	78.9	0.846	0.153	**22.40**	0.699	0.255	30.4	88.9	0.801	0.170
	16	**22.64**	**0.770**	**0.215**	**27.5**	**79.0**	**0.854**	**0.151**	22.24	**0.700**	**0.247**	**39.6**	**90.8**	**0.804**	**0.164**
3	4	24.28	0.813	0.184	25.8	**79.7**	0.853	0.146	24.28	0.755	0.205	34.4	89.6	0.809	0.158
	16	**24.57**	**0.822**	**0.171**	**27.1**	79.5	**0.862**	**0.145**	**24.29**	**0.760**	**0.194**	**38.5**	**90.5**	**0.812**	**0.155**
5	4	25.35	0.838	0.162	25.2	78.8	0.852	0.141	25.33	0.782	0.182	33.5	87.7	**0.816**	0.153
	16	**25.88**	**0.849**	**0.147**	**25.9**	**79.1**	**0.860**	**0.132**	**25.56**	**0.794**	**0.168**	**40.0**	**89.6**	**0.816**	**0.151**
7	4	25.89	0.851	0.150	24.5	78.5	0.853	0.137	25.71	0.796	0.169	34.0	86.7	**0.817**	0.151
	16	**26.45**	**0.862**	**0.138**	**25.6**	**79.1**	**0.857**	**0.130**	**26.04**	**0.810**	**0.158**	**39.0**	**88.8**	0.816	**0.150**

Table 12: Effect of Gaussian head finetuning with AnySplat, $d_z = 4$, $N = 16$, and one-step inference. Image embedding only denotes finetuning h_{img} while keeping the remaining pretrained Gaussian head weights frozen.

		RE10K							DL3DV						
		Appearance			Camera		Depth		Appearance			Camera		Depth	
n	Gaussian head	PSNR↑	SSIM↑	LPIPS↓	AUC@3° ↑	AUC@30° ↑	$\delta_{1.25}$ ↑	AbsRel↓	PSNR↑	SSIM↑	LPIPS↓	AUC@3° ↑	AUC@30° ↑	$\delta_{1.25}$ ↑	AbsRel↓
2	Fully frozen	22.14	0.754	0.233	26.4	78.8	0.833	0.161	22.01	0.685	0.267	30.6	88.8	0.790	0.179
	Image embedding only	22.56	0.766	0.223	26.7	78.9	0.846	0.153	22.40	0.699	0.255	30.4	88.9	0.801	0.170
	Full finetuning	22.88	0.775	0.216	26.8	78.8	0.854	0.148	22.69	0.708	0.248	30.2	88.8	0.807	0.166
3	Fully frozen	24.07	0.806	0.190	25.6	78.8	0.846	0.150	24.05	0.747	0.212	34.1	88.8	0.803	0.163
	Image embedding only	24.28	0.813	0.184	25.8	79.7	0.853	0.146	24.28	0.755	0.205	34.4	89.6	0.809	0.158
	Full finetuning	24.47	0.818	0.180	26.0	80.0	0.857	0.143	24.50	0.761	0.200	34.6	90.0	0.814	0.155
5	Fully frozen	25.24	0.834	0.165	25.0	78.8	0.849	0.143	25.21	0.779	0.184	33.6	87.5	0.814	0.155
	Image embedding only	25.35	0.838	0.162	25.2	78.8	0.852	0.141	25.33	0.782	0.182	33.5	87.7	0.816	0.153
	Full finetuning	25.47	0.842	0.159	25.3	78.8	0.855	0.139	25.44	0.785	0.180	33.4	87.5	0.818	0.151
7	Fully frozen	25.86	0.850	0.151	24.7	78.8	0.852	0.138	25.68	0.795	0.170	33.9	86.3	0.816	0.152
	Image embedding only	25.89	0.851	0.150	24.5	78.5	0.853	0.137	25.71	0.796	0.169	34.0	86.7	0.817	0.151
	Full finetuning	25.96	0.853	0.148	24.5	78.8	0.855	0.135	25.76	0.798	0.168	34.1	87.5	0.818	0.150

One-step finetuning retains most of the rendering quality of the 50-evaluation reference while achieving a speedup of approximately $19\times$. It also outperforms the unfinetuned model at one or eight evaluations. At the same single-evaluation cost, it achieves better PSNR, SSIM, and LPIPS than both deterministic regression and one-step training from scratch on RE10K and DL3DV. These results support combining flow-matching pretraining with one-step finetuning for efficient latent completion.

D.3 Effect of Latent Channel Count

Table 11 compares DensiTok with $d_z = 4$ and $d_z = 16$ latent channels on RE10K and DL3DV after one-step finetuning. Both variants use the same evaluation views, respectively.

Increasing the latent channel count from 4 to 16 expands the representational capacity of the latent space and relaxes the compression bottleneck. The 16-channel variant improves most appearance, camera, and depth metrics across both datasets and input counts. These results suggest that a wider latent representation can better preserve appearance and geometric information, although improvements are not uniform across all metrics.

D.4 Effect of Gaussian Head Finetuning

Table 12 compares three Gaussian head configurations with AnySplat on RE10K and DL3DV. *Fully frozen* keeps all pretrained Gaussian head weights fixed. *Image embedding only* finetunes only a few image input layers in the Gaussian head, as described in Appendix A.2. *Full finetuning* updates the entire Gaussian head using the same training hyperparameters as *Image embedding only*.

Keeping the Gaussian head fully frozen yields performance close to image embedding finetuning on both datasets. The gap in appearance and depth quality becomes negligible as the number of observed views n increases. We attribute this trend to the smaller spacing between input views

along the same camera trajectory. Closer input views provide better coverage in the warped color prior, reducing the impact of holes and the need for appearance adaptation.

Finetuning the entire Gaussian head further improves appearance and depth quality, while camera accuracy remains broadly similar across the three configurations. These gains are modest, and both the frozen head and image embedding finetuning retain comparable performance.