FLASHLVSM: A CONSTANT-TIME VIEW-SYNTHESIS TRANSFORMER

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent scene representations have emerged as a promising alternative for novel view synthesis, encoding multi-view observations into learned features without relying on predefined geometric primitives or rendering operations. Despite their promise, realizing them with modern neural architectures, such as transformers, often incurs substantial rendering cost. As the number and resolution of source views increase, every target token repeatedly attends to a growing set of source tokens through a deep neural renderer, causing the rendering cost to grow rapidly. We present *FlashLVSM*, a constant-time view-synthesis model designed for scalable, highly efficient rendering from latent scene representations, which is carefully co-designed around four complementary principles, each addressing a distinct source of rendering cost. A richer, more 3D-expressive latent representation enables us to use a lightweight decoder without sacrificing rendering quality. Target-aware view selection restricts computation to source views geometrically relevant to the target, while target-aware token pruning further retrieves only the informative tokens within them, making rendering cost independent of the total number of input views. Finally, mixed-precision inference accelerates the remaining decoder computation. Together, *FlashLVSM* renders at 142 FPS with 32 input views on a single consumer-grade GPU, and maintains roughly 140 FPS as the number of views increases to 256 while preserving rendering quality. This corresponds to speedups of nearly 12× and 95×, respectively, with 204× fewer FLOPs and 25× lower peak memory at 256 views. Despite this reduction in computation, *FlashLVSM* remains competitive in rendering quality, matching or exceeding the baseline from 64 input views onward. These results establish efficient latent scene representations as a compelling alternative to conventional explicit representations for efficient novel view synthesis.

1 INTRODUCTION

Recent advances in novel view synthesis have shown that multi-view observations can be encoded into learned latent scene representations and directly rendered into novel views with minimal 3D inductive bias (Sajjadi et al., 2022; Jin et al., 2025). Unlike Neural Radiance Fields (Mildenhall et al., 2020) and 3D Gaussian Splatting (Kerbl et al., 2023), which rely on predefined primitives and volumetric rendering operations, latent representations learn how to organize scene information directly from data, offering a promising alternative for feed-forward novel view synthesis. Modern view-synthesis network architecture, such as transformers, realize this idea by representing the scene as a set of source tokens and rendering each novel view through target-conditioned attention. Specifically, each target token repeatedly attends to the source tokens across decoder layers, allowing it to progressively retrieve and refine the scene information most relevant to the target view. With minimal inductive bias, this formulation scales naturally with large-scale training data and generalizes well across diverse scenes, making it a strong foundation for feed-forward view synthesis.

Despite this promise, rendering speed falls far below real-time rates as the number and resolution of input views increase. Recent view-synthesis transformers (Szymanowicz et al., 2026) pair a powerful but heavy 3D foundation model (Wang et al., 2025) as the encoder with a comparatively lightweight decoder, so that rendering stays fast once the scene has been encoded. The decoder, however, still performs dense cross-attention between the target tokens and every source token, and rendering cost quickly becomes prohibitive. At 256 views of 1024×1024 images with a patch size of

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.

14, the source token sequence exceeds one million tokens, and the decoder repeatedly attends over it for every view it renders, at a cost well beyond what a real-time application can afford. Moreover, the decoder itself is not negligible, typically consisting of multiple transformer layers that repeatedly perform attention and feed-forward computation for each novel view. This is considerably more expensive than conventional rendering pipelines based on explicit scene representations, such as meshes or 3D Gaussians, where novel views can be rendered efficiently without repeatedly evaluating a deep neural decoder. Long source token sequences and substantial decoder depth therefore compound the per-view rendering cost, posing a major obstacle to scalable, real-time view synthesis.

First, we ask whether all source tokens are needed to render a novel view, and argue that they are not. In a dense capture of a large scene, each camera observes only part of that scene, and many source frustums overlap the target frustum only partially or not at all, making the tokens they carry irrelevant or only weakly informative for the target view. Moreover, the source views that overlap the target also overlap heavily with one another, so a large fraction of tokens is redundant and may contribute little to synthesizing the target view. Dense cross-attention thus spends much of its computation on tokens that are either irrelevant to the target or redundant with other observations. Selecting nearby source views and pruning redundant tokens within them can reduce decoder computation while preserving target-relevant information.

A straightforward way to remove such redundancy is to compress the source tokens into a compact set of learnable scene tokens and render novel views from those alone, which forces the encoder to gather into a fixed budget the information required to represent the scene. Such compression, however, is lossy by construction, and the detail it discards cannot be recovered at render time. It is also target-agnostic. Different target tokens require different source tokens, yet the encoder receives no target information and must commit to a single summary shared by all views. This introduces a trade-off in which greater compression improves efficiency but risks discarding target-specific details. In practice, compressive scene tokens have accordingly been shown to render at lower quality than the full source token set (Jin et al., 2025; Szymanowicz et al., 2026).

In this work, we propose *FlashLVSM*, a constant-time view-synthesis transformer. Our key insight is that rendering a single target view does not require all source tokens, but only a small subset relevant to the target. To exploit this, we introduce target-aware token pruning: we first use camera geometry to select a small set of nearby source views, and then retrieve the tokens most relevant to the target within this restricted source pool. This avoids exhaustively comparing target tokens against all source tokens while preserving target-relevant information. By preserving all source tokens while selectively reading a fixed number of target-relevant tokens, *FlashLVSM* minimizes information loss while enabling constant-time rendering independent of the number of input views.

We also revisit the depth of the neural renderer itself. Rendering inherently involves a trade-off between the expressiveness of the scene representation and the renderer's complexity. When the scene is represented explicitly and richly, as with meshes or 3D Gaussians, it is already organized into a rendering-ready scene representation, allowing novel views to be produced with a lightweight rendering operation. In contrast, a compact or weakly structured scene representation places a greater burden on the renderer, which must recover the missing geometric and appearance information through multiple layers of neural computation. This suggests that a deep decoder is not necessarily intrinsic to transformer-based view synthesis, but may instead compensate for limitations of the underlying scene representation. Indeed, we empirically find that replacing the baseline representation with a more expressive 3D representation from a more recent 3D foundation model lets us cut decoder depth in half while maintaining rendering quality. Strengthening the scene representation therefore offers a complementary path to efficient rendering, reducing the amount of neural rendering required in the first place.

FlashLVSM maintains approximately 140 FPS as the number of input views increases from 32 to 256, achieving speedups ranging from nearly 12$\times$ to 95$\times$ over the dense baseline. At 256 input views, it requires only 1/204 of the renderer FLOPs and 1/25 of the renderer peak memory. Despite these substantial efficiency gains, our model preserves competitive rendering quality across multi-view settings. In contrast, directly applying the same view selection, token pruning, and quantization to the baseline leads to a substantially larger quality drop, including a PSNR loss of nearly 4 dB at 32 input views. These results demonstrate that our framework combines high rendering quality with an increasingly large speed advantage as more source views become available.

2 Related Work

Generalizable novel view synthesis. Generalizable novel view synthesis methods train a network across many scenes to infer a renderable scene representation in a single forward pass. Geometry-aware methods predict a radiance field (Yu et al., 2021; Wang et al., 2021; Chen et al., 2021; Johari et al., 2022; Xu et al., 2024; Hong et al., 2024) or a set of 3D Gaussians (Charatan et al., 2024; Chen et al., 2024; Xu et al., 2025; Zhang et al., 2024; Ye et al., 2024; Jiang et al., 2025b; Ye et al., 2026; Hong et al., 2025; Kang et al., 2026; Ziwen et al., 2025; Zhao et al., 2026) and render it through a fixed, hand-designed model such as volume rendering or rasterization. This representation provides a strong inductive bias but bounds rendering quality by the accuracy of the geometry it encodes. Geometry-free methods instead encode appearance, and optionally camera information, into source tokens without assuming any 3D structure (Sajjadi et al., 2022; 2023; Jin et al., 2025; Jiang et al., 2025a). A transformer then renders each target view by attending from the target rays to the source tokens, learning both the retrieval of relevant content and its blending into pixels. With minimal 3D inductive bias, these models scale favorably with data and compute (Jin et al., 2025; Kim et al., 2026; Szymanowicz et al., 2026). However, since the network itself is the renderer, every target view requires a transformer pass, and rendering remains slow.

Efficient view-synthesis transformers. View-synthesis transformers differ in how they represent the source views at render time. Decoder-only models process source and target tokens jointly at every layer and achieve the highest fidelity (Jin et al., 2025). However, they reprocess all source tokens for every rendered view, and the cost grows quadratically with the number of input views. Compressive models instead encode the source views once into a fixed-size latent representation, such as learnable scene tokens (Sajjadi et al., 2022; Jin et al., 2025) or fast weights (Zhang et al., 2026). They render quickly, but the compression is lossy, and quality falls short of decoder-only models. Recent works bridge the two designs by encoding the source views once without compression and reusing them across all target views (Jia et al., 2026; Kim et al., 2026; Szymanowicz et al., 2026). Their decoders, however, still attend to every source token, and rendering cost grows linearly with the number of input views. CLiFT (Wang et al., 2026c) retrieves a subset of tokens for each target view, but selects them from a representation already compressed. We instead keep all source tokens and let the decoder read a fixed number of target-relevant tokens, making rendering cost independent of sequence length.

Token reduction for vision transformers. Reducing the number of tokens is a well-studied route to efficient transformers. Vision transformers prune tokens by learned or attention-based importance (Rao et al., 2021; Liang et al., 2022) or merge them by similarity (Bolya et al., 2023). Similar strategies have recently accelerated multi-view geometry transformers, where global attention over hundreds of frames dominates runtime, through token merging guided by similarity, geometry, or confidence (Shen et al., 2025; Wang et al., 2026b; Shu et al., 2026; Chen et al., 2026) and block-sparse global attention (Wang et al., 2026a). These methods, however, reduce tokens independently of any downstream query, so all outputs share a single reduced set. In novel view synthesis, the source tokens that matter depend on the target view itself. We therefore leave the stored source tokens intact and let the decoder read only a target-relevant subset of them, selected independently for each target view. We first restrict the candidates to nearby source views using camera geometry, then retrieve the tokens most relevant to the target within that pool.

3 Method

3.1 Preliminary

We build on a foundation-model encoder decoder framework for novel view synthesis. Given source images $\mathcal{I}_s$ and their camera parameters $\mathcal{C}_s$, a pretrained multi-view encoder E jointly produces source tokens, while a ray embedding ϕ maps the target camera's Plücker ray grid $\mathcal{R}_t$ to the target tokens:

$$S = E(\mathcal{I}_s, \mathcal{C}_s), \qquad X_t = \phi(\mathcal{R}_t). \tag{1}$$

Cross-view interactions allow each source view's token to incorporate information from other views. These target-independent features are encoded once and cached for subsequent rendering. A transformer decoder R, referred to as the renderer, processes source and target tokens using bidirec-

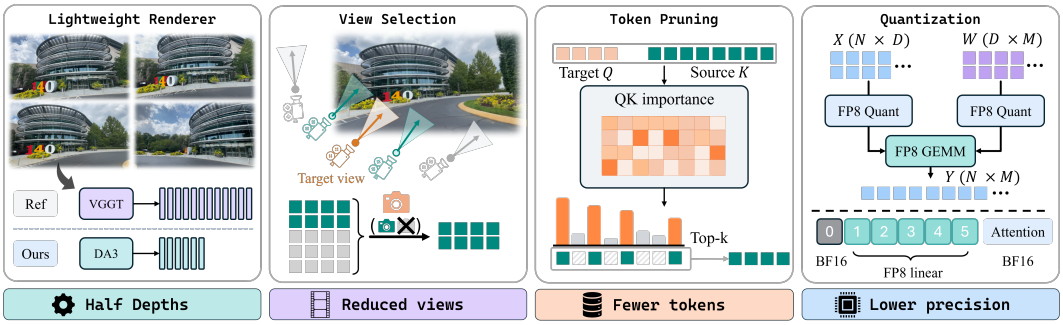

Figure 1: Overview of our rendering acceleration framework. From left to right: (1) a DA3 encoder paired with a 6-block renderer; (2) geometry-guided view selection favoring source cameras within nearby positions and aligned viewing directions; (3) target-aware token pruning that retains source tokens with the highest importance scores computed from target queries and source keys; and (4) FP8 quantization of renderer linear operations, while keeping block 0 and attention computation in BF16. These components reduces renderer depth, source-view count, and numerical precision.

tional cross-attention and maps the final target features to RGB patches to produce the target image $\hat{I}_t = R(S, X_t)$. Although source encoding is shared across target views, the renderer is executed for every target, with its computational cost growing with the source token count. This motivates reducing both renderer depth and the source tokens processed for each target.

3.2 Richer 3D Representations, Lighter Decoder

Rendering involves an inherent trade-off between the expressiveness of the scene representation and the complexity of the renderer. Explicit representations such as meshes or 3D Gaussians encode much of the scene geometry and appearance in a rendering-ready form, allowing novel views to be produced with lightweight rendering operations. In contrast, when the scene representation is less structured or expressive, the renderer must recover the missing scene information through neural computation, requiring a deeper rendering network. From this perspective, a deep decoder is not necessarily intrinsic to transformer-based view synthesis; rather, part of its capacity may compensate for limitations of the underlying scene representation.

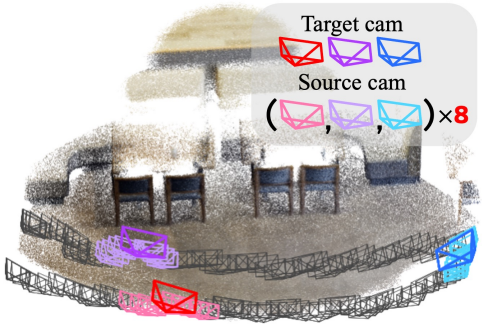

Figure 2: Geometry-guided view selection. Gray frustums denote candidate source views; colored frustums show each target and its top-8 selected source views.

Motivated by this observation, we build our model upon a richer 3D scene representation from a recent powerful 3D foundation model, Depth Anything 3 (Lin et al., 2025) and pair it with a lightweight transformer renderer. Compared with the 12-layer decoder in the baseline method, LagerNVS (Szymanowicz et al., 2026), we find that this richer 3D representation lets us halve the decoder depth (6-layer) while maintaining rendering quality. This trade-off is particularly favorable for efficient novel-view synthesis. The encoder is executed only once per scene, and its representation is cached, whereas the decoder is executed for every target view. Investing more computation in the one-time scene representation therefore enables a substantially lighter recurring rendering stage.

3.3 Geometry-guided View Selection

We select views relevant to the target pose by ranking source views based on camera proximity and viewing-direction agreement. For target t and source s, we compute the view scores $w_{t,s}$ as follows,

$$w_{t,s} = \max\left(0, d_t^\top d_s\right) \exp\left(-\frac{\|C_t - C_s\|_2}{\tau\sigma}\right), \tag{2}$$

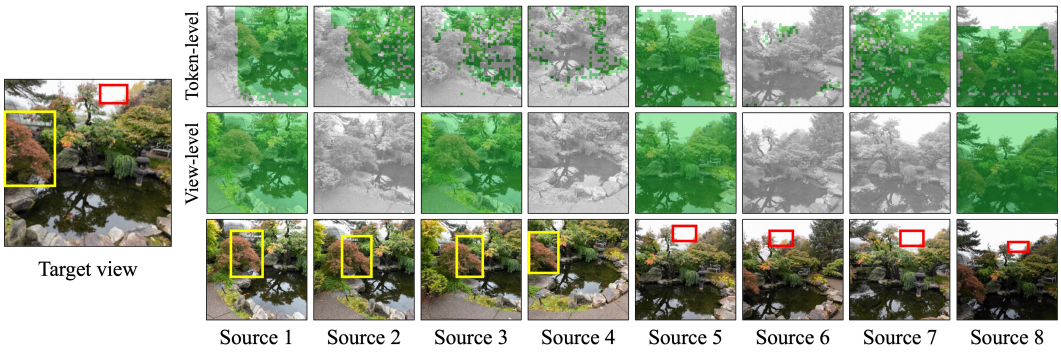

Figure 3: Comparison of the proposed token-level and view-level pruning under the same token budget. Top: token-level pruning; bottom: view-level pruning. Green and gray denote retained and pruned source tokens, respectively. Token-level pruning allocates the token budget to target-relevant regions across multiple source views, whereas view-level pruning retains or discards entire views.

where C_s, C_t denote the camera origins of each source and target view, d_s, d_t represent source and target camera directions, and σ is the mean distance of all pairwise source camera centers, which makes the distance term invariant to scene scale. We set $\tau = 0.5$ throughout the experiments, and retain all tokens from top-k scoring views ($k = 8$, by default). Figure 2 illustrates the resulting target-dependent view selection.

We first jointly encode all available source views, then select a small subset of source views for each target. In this way, all source tokens can incorporate information from views that are not directly passed to the renderer for a specific target. The encoded tokens from all source views remain cached, while each target reads at most $k \cdot$ (the number of tokens in each source view) source tokens during the rendering process. Thus, given the output rendering resolution and the encoder's source token resolution, the computational complexity of rendering for each target view is bounded independently of the available view count.

3.4 Target-view-dependent Source Token Pruning

The previous view selection stage provides only coarse-grained token selection, as it retains or discards entire source views. However, not all tokens within the retained views contribute equally to target rendering. We therefore perform an additional token-level selection at the l-th rendering decoding block. Importantly, our token pruning is target-view dependent: the importance of each source token is computed with respect to the queries of the current target view, allowing a different subset of source tokens to be preserved for each rendered view.

Since the pre-softmax query-key (QK) logit reflects the relevance between a target query and a source token, we use the target-to-source QK relevance to estimate source-token importance. Using the query and key projections of block l, for each target query n and source token i, we compute

$$\ell_{n,i} = \sum_{h=1}^{H} \frac{q_{h,n}^{\top} k_{h,i}}{\sqrt{D}}, \tag{3}$$

where H is the number of attention heads and D is its dimension. Here, $q_{h,n} \in \mathbb{R}^D$ denotes the n-th target query of head h, and $k_{h,i} \in \mathbb{R}^D$ denotes the key of source token i. We normalize the relevance over the source-token dimension for each query and accumulate it across all queries:

$$p_{n,i} = \frac{\exp(\ell_{n,i})}{\sum_{j=1}^{N_s} \exp(\ell_{n,j})}, \qquad a_i = \sum_{n=1}^{N_t} p_{n,i}, \tag{4}$$

where N_t is the number of non-register target tokens, and N_s is the number of source tokens from the retained views only (from the view selection stage). The per-target softmax makes source tokens compete for each target query, and the summation aggregates the relevance each source token receives across all targets. We use the resulting score a_i as the source-token importance for pruning. Based on this importance score, we retain the top-$\lfloor pN_s \rfloor$ source tokens, where $p \in (0, 1]$ denotes

the token-keeping ratio. Blocks preceding block l operate on the source tokens preserved by coarse view selection, while block l and all subsequent blocks operate on the reduced token set.

Figure 3 qualitatively compares token-level and view-level pruning under the same token budget. Token-level pruning retains only target-relevant regions, such as the visible part of the red-leaved tree and the small sky region between trees, while discarding irrelevant regions within the same source views. In contrast, view-level pruning must retain or discard entire views, wasting budget on irrelevant content and potentially removing useful regions, as in Sources 2 and 7. This shows that token-level pruning allocates a limited budget more precisely across complementary source regions.

3.5 Mixed-Precision FP8 Rendering

Token pruning reduces the source sequence length and, consequently, the cost of cross-attention. We further accelerate the remaining renderer computation through mixed-precision inference. Specifically, we execute most linear layers in FP8 while retaining attention operations in BF16. This design targets the matrix multiplications that contribute significantly to decoder computation while preserving higher precision for attention, where numerical stability is more sensitive. Such mixed-precision execution is particularly well suited to modern GPU architectures, whose tensor cores provide dedicated support and higher throughput for low-precision matrix multiplication.

FP8 quantization. We quantize the Linear layers of the renderer using the FP8 E4M3 format, with dynamic per-token scaling for the input activations and static per-output-channel scaling for the weights. For a linear layer $y = xW^\top$, we choose scales that are constant along the shared input-feature dimension, allowing them to be applied outside the dot product. This preserves the standard FP8 matrix-multiplication structure while still allowing independent scaling across activation tokens and weight output channels. Compared with sharing a single scale across the entire tensor, this finer granularity allows each row to better utilize the E4M3 dynamic range. Weight scales are precomputed once from the fixed model weights and reused during inference, whereas activation scales are computed dynamically from each layer's runtime activations.

Sensitivity-aware mixed precision. We observe that the first renderer block is much more sensitive to FP8 quantization than the later blocks. We therefore keep all Linear layers in the first block in BF16 and quantize the Linear layers in the remaining blocks to FP8. The first block directly consumes features from the frozen encoder, whereas subsequent blocks operate on representations that have already been processed by the renderer. An empirical analysis of this sensitivity-based precision choice is provided in section A.1

BF16 attention. We keep the attention operations in BF16. Although FP8 attention slightly degrades the rendering quality in our experiments, it does not improve throughput at our target resolution. We attribute this partly to the overhead of quantizing the source key and value tensors, including amax reduction, scaling, and casting, which is incurred largely independently of the number of target queries, whereas the computational benefit of FP8 attention grows with the amount of target-query computation. We therefore retain BF16 attention in the final model and provide a detailed FP8 attention analysis in section A.2.

4 Experimental Results

4.1 Implementation Details

Architectures and training. We compare LagerNVS (Szymanowicz et al., 2026) with a VGGT (Wang et al., 2025) encoder and a 12-block renderer against our DA3-giant (Lin et al., 2025) model with a 6-block renderer. Both renderers use a hidden dimension of 768 and 12 attention heads. The models at 256×256 and 512×512 use target patch sizes of 8 and 16, respectively. For source inputs, DA3 operates on 504×504 images and produces 1,296 source tokens per view, while VGGT runs on 518×518 images and produces 1,369 source tokens per view. The 256-resolution models are trained for 100K iterations with trainable encoders, followed by 50K iterations of fine-tuning at 512×512 resolution with the encoders frozen.

Data and evaluation. We evaluate on DL3DV (Ling et al., 2024) test set with 32, 64, and 128 source views, following view indices used in Long-LRM (Ziwen et al., 2025), and 256 source views

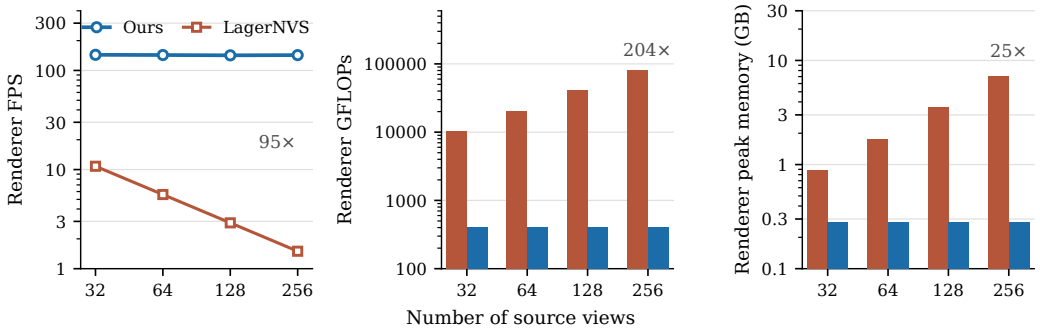

Figure 4: Scaling behavior with increasing numbers of source views. (Left): Renderer FPS. (Middle) Renderer GFLOPs. (Right): Renderer peak memory. FlashLVSM maintains nearly constant renderer FPS, GFLOPs, and peak memory, while LagerNVS becomes increasingly expensive as the input-view count grows. The source-view axis is shown on a logarithmic scale in the FPS plot.

following MVP (Kang et al., 2026) evaluation protocol. Rendering uses BF16 autocast by default; *+PQ* variants add view selection, token pruning, and BF16/FP8 mixed precision. FPS is measured on a single RTX 5090.

Pruning and quantization. View selection retains 8 source views per target pose ($k = 8$), independent of the number of input views N. Token pruning is then applied before the second block ($l = 2$), where a keep ratio of $p = 0.5$ retains half of the remaining tokens. Only the first two blocks therefore process 8×1296 tokens and the remaining blocks half that, a budget independent of N.

4.2 Results

Lightweight decoder. We first evaluate the encoder renderer configuration in table 1. With a DA3 encoder and our model achieves comparable quality to the VGGT-based, 12-block LagerNVS under 6 view protocol: 29.40 versus 29.46 dB PSNR. Under Long-LRM evaluation protocol, our model improves PSNR by 0.62 and 1.47 dB at 32 and 64 views, respectively, with corresponding improvements in SSIM and LPIPS. These results show that the proposed configuration maintains rendering quality with half the renderer depth.

Pruning and quantization. Table 2 compares FlashLVSM with LagerNVS at 256×256. PQ denotes the proposed pruning and quantization modules. FlashLVSM (Ours) incurs a 0.16 dB PSNR decrease at 32 views, but exceeds LagerNVS by 0.86 and 1.41 dB at 64 and 128 views, respectively, with higher SSIM and lower LPIPS. Our model therefore remains competitive in quality, and increasingly favorable as the number of views grows. A qualitative comparison between Ours and Ours w/o PQ in fig. 5 shows no noticeable difference in perceived quality. Results at 512×512 follow the same pattern and are reported in section B.1.

Constant-time rendering. As shown in fig. 4, our fixed token budget largely decouples rendering cost from the number of input views. Renderer FPS, GFLOPs, and GPU peak memory remain nearly constant as the source sequence grows, whereas LagerNVS becomes progressively more expensive, resulting in an increasingly large efficiency gap at higher view counts. As a result, FlashLVSM achieves $95\times$ higher FPS while requiring only $1/204$ of the renderer FLOPs and $1/25$ of the peak memory of LagerNVS at 256 source views.

4.3 Ablation studies

We ablate each component and the design choices of our pruning strategy. Unless noted otherwise, all experiments in this section are performed under the 64-view setting.

Overall component ablation. We evaluate the contribution of each component in table 3. The dense 64-view baseline achieves 26.73 dB at 13 FPS. Applying view selection reduces the decoder's input to eight source views and increases throughput to 92 FPS while retaining 26.49 dB, corresponding to only a 0.24 dB drop. Starting from the view selection configuration, token pruning further reduces the token budget to the equivalent of four views, further improving throughput to 110 FPS with a

Table 1: A strong encoder supports a lightweight renderer. Paired with DA3 (Lin et al., 2025), our 6-layer renderer matches the 12-layer VGGT (Wang et al., 2025)-based LagerNVS (Szymanowicz et al., 2026). Layers denotes renderer depth. †The 6-view results use the LagerNVS test indices; the 32- and 64-view results use the Long-LRM (Ziwen et al., 2025) evaluation indices.

Method	Layers	6† views			32 views			64 views		
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
LagerNVS	12	**29.46**	**0.906**	**0.068**	26.06	0.828	0.105	25.26	0.798	0.108
Ours	6	29.40	0.903	0.070	**26.68**	**0.844**	**0.103**	**26.73**	**0.840**	**0.097**

Table 2: Main comparison against LagerNVS (Szymanowicz et al., 2026) across different numbers of input views at 256×256. FPS is measured using an RTX 5090 GPU.

Method	32 views				64 views				128 views				256 views			
	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑
LagerNVS	26.06	0.828	0.105	12.0	25.26	0.798	0.108	6.0	23.94	0.751	0.127	3.1	22.58	0.701	0.150	1.5
+ PQ	22.06	0.678	0.164	79	21.57	0.652	0.170	79	20.96	0.622	0.185	79	20.27	0.591	0.215	79
Ours w/o PQ	**26.68**	**0.844**	**0.103**	25.3	**26.73**	**0.840**	**0.097**	12.8	**25.82**	**0.806**	0.110	6.5	**23.53**	**0.737**	**0.137**	3.3
Ours	25.90	0.828	0.111	**142**	26.13	0.831	0.101	**143**	25.35	0.802	**0.108**	**141**	**23.11**	0.727	**0.137**	**141**

Table 3: Ablation of the proposed modules. VS, Pr., and Qu. denote view selection, token pruning, and quantization, respectively. View Equiv. denotes the remaining source-token budget expressed as an equivalent number of complete source views (1,296 tokens per view).

VS	Pr.	Qu.	View Equiv.	PSNR	SSIM	LPIPS	FPS
			64	26.73	0.840	0.097	13
✓			8	26.49	0.840	0.094	92
✓	✓		4	26.33	0.836	0.097	110
✓		✓	8	26.28	0.835	0.098	119
✓	✓	✓	4	26.13	0.831	0.101	143

modest decrease to 26.33 dB. Our mixed-precision quantization provides a complementary source of acceleration, reaching 119 FPS at 26.28 dB without token pruning. Since our renderer operates on a fixed source token budget independent of the number of input views, its rendering cost remains constant as the input sequence grows, yielding increasingly larger throughput over dense rendering.

When and how much to prune source tokens. We evaluate the effect of the pruning location in table 4. Pruning before any renderer block ($l = 0$) gives the highest throughput, since all subsequent blocks operate on the reduced token set, but causes a substantial quality drop to 21.48 dB. Allowing the first block to attend over all view-selected source tokens before pruning improves PSNR to 25.78 dB, and extending this to the first two blocks raises it to 26.13 dB. Delaying pruning beyond this point provides little additional benefit, with block 3 achieving a similar 26.12 dB at lower throughput. Token pruning is therefore effective only after the early blocks have established informative target–source interactions, and we prune immediately before block 2. This also explains why geometry-guided view selection is necessary. Token-level pruning allocates the budget more precisely across source views than view-level pruning does (fig. 3), but processing all source views through the early blocks is prohibitively expensive at long input sequences such as 256 views. View selection therefore acts as a coarse pre-filter that bounds the number of tokens reaching them.

We next sweep the source-token keep ratio in table 5. As the keep ratio decreases, rendering throughput increases at the cost of reconstruction quality. Retaining 62.5%, 50%, and 25% of the source tokens yields 26.20, 26.13, and 25.76 dB, respectively, as an expected quality–efficiency trade-off. At the same 50% token budget, random pruning performs worse than our target-aware selection, showing that the benefit comes from identifying relevant source tokens rather than simply shortening the sequence without considering the target rendering view. We use a 50% keep ratio by default as a balanced operating point between rendering quality and speed.

4.4 Test-Time Optimization

Our pretrained model follows the LagerNVS (Szymanowicz et al., 2026) training protocol and is trained with only 2–6 source views. To adapt it to substantially longer sequences, we perform test-time optimization (TTO) at 256×256 resolution with lightweight LoRA adapters (Hu et al., 2021), keeping the encoder, the renderer weights, and the cached source tokens frozen. All source images are encoded once. At each step we hold out one source view as a pseudo-target, exclude its tokens

Table 4: Effect of the pruning location.

Block l	PSNR	SSIM	LPIPS	FPS
0	21.48	0.671	0.199	**166**
1	25.78	0.821	0.107	150
2	**26.13**	0.831	**0.101**	143
3	26.12	**0.832**	**0.101**	134

Table 5: Quality–efficiency trade-off with different source-token keep ratios.

Keep Ratio p	View Equiv.	PSNR	SSIM	LPIPS	FPS
0.625	5	**26.20**	**0.833**	**0.100**	134
0.500	4	26.13	0.831	0.101	143
0.250	2	25.76	0.821	0.109	**165**
0.500 + Random mask	4	24.33	0.774	0.142	147

Table 6: Test-time optimization at 256×256. FlashLVSM adapts under its fixed token budget; LagerNVS adapts over all source tokens and runs out of memory (OOM) at 256 views.

Method	32 views				64 views				128 views				256 views			
	PSNR	SSIM	LPIPS	Time (m)	PSNR	SSIM	LPIPS	Time (m)	PSNR	SSIM	LPIPS	Time (m)	PSNR	SSIM	LPIPS	Time (m)
LagerNVS	27.32	0.865	0.094	4.7	28.81	0.893	0.074	19.6	29.84	0.910	0.063	53.8	OOM			
Ours	26.85	0.855	0.104	2.9	28.10	0.880	0.085	5.9	28.98	0.895	0.074	8.7	29.78	0.910	0.066	11.3

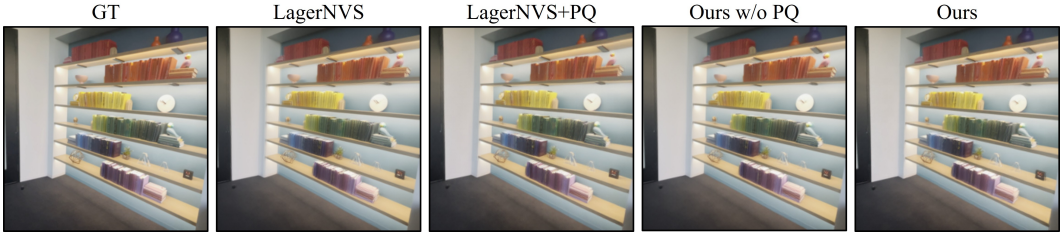

Figure 5: Qualitative comparison with LagerNVS. Ours preserve fine details and overall visual quality with no noticeable degradation, while substantially accelerating rendering.

from the renderer's conditioning, and supervise reconstruction of the observed image. We apply the same view selection, token pruning, and mixed-precision configuration used at evaluation, so that the renderer adapts to its inference-time computation. TTO is inexpensive, requiring roughly 3 minutes for 500 iterations using a single H100. Further details are provided in section B.5.

Table 6 reports the results after test-time optimization. Note that the two rows are not budget-matched: LagerNVS adapts a 12-block renderer over all source tokens, so both its per-iteration cost and memory grow with the number of views, whereas FlashLVSM, the gain grows with the number of source views ($+0.95$, $+1.97$, and $+3.62$ dB at 32, 64, 128 views) indicating that the fixed budget itself is not what limits long-sequence quality; the renderer was pretrained with only 2–6 views, and a lightweight adapter recovers most of the gap. Under its unbounded budget, LagerNVS remains 0.5–0.9 dB higher at 32–128 views, but its adaptation cost and memory scale with the number of source views and it runs out of memory at 256 views. FlashLVSM adapts at constant per-iteration cost, extends to 256 views (29.78 dB), and renders the adapted model at 141 FPS.

5 Limitations and Conclusion

We presented FlashLVSM, a constant-time view-synthesis transformer. Rather than compressing the scene into a fixed latent, we keep all encoded source tokens and let each target view read a fixed budget of relevant ones through two stages of target-aware selection: geometry-guided view selection bounds the candidates to nearby cameras, and similarity-based token pruning retains only the tokens the target actually attends to. We further showed that a more 3D-expressive latent representation, obtained from a stronger foundation encoder, allows decoder depth to be halved without loss of quality, and that FP8 mixed-precision inference accelerates the remaining computation. Together, these components render at roughly 140 FPS regardless of the source sequence length, a $95\times$ speedup over dense rendering at 256 views with $1/204$ of the FLOPs and $1/25$ of the memory, while matching or exceeding the dense baseline in quality from 64 views onward. These results suggest that latent scene representations can be rendered efficiently enough to compete with explicit ones.

Our approach has limitations. Pruning costs non-negligible quality drop at small view counts and slightly increases LPIPS at high resolution. The fixed view budget assumes that a target is covered by a handful of nearby cameras, which need not hold for wide-baseline or sparsely captured scenes, and view selection relies on known camera poses and a hand-designed scoring function. Learning the selection and pruning criteria end-to-end, adapting the token budget to scene coverage, and extending target-aware pruning to dynamic scenes are promising directions for future work.

AI USE STATEMENT

We used generative AI tools to assist with manuscript drafting and language editing, and to support discussions of methodological choices and experimental design. All AI-assisted content was reviewed and revised by the authors. The authors take full responsibility for the final manuscript, including its technical claims, experimental results, and conclusions.

REFERENCES

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *Proceedings of the International Conference on Learning Representations*, 2023.

David Charatan, Sizhe Lester Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelsplat: 3d gaussian splats from image pairs for scalable generalizable 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

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 *Proceedings of the European Conference on Computer Vision*, 2024.

Yutian Chen, Yuheng Qiu, Ruogu Li, Jay Patrikar, and Sebastian Scherer. Co-me: Confidence guided token merging for visual geometric transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

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 *Proceedings of the International Conference on Machine Learning*, 2025.

Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. Lrm: Large reconstruction model for single image to 3d. In *Proceedings of the International Conference on Learning Representations*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Hwasik Jeong, Seungryong Lee, Gyeongjin Kang, Seungkwon Yang, Xiangyu Sun, Seungtae Nam, and Eunbyung Park. 2xplat: Two experts are better than one generalist. *arXiv preprint arXiv:2603.21064*, 2026.

Xiaosong Jia, Yihang Sun, Junqi You, Zichen Zou, Junchi Yan, Zuxuan Wu, Yu-Gang Jiang, et al. Efficient-lvsm: Faster, cheaper, and better large view synthesis model via decoupled co-refinement attention. In *Proceedings of the International Conference on Learning Representations*, 2026.

Hanwen Jiang, Hao Tan, Peng Wang, Haian Jin, Yue Zhao, Sai Bi, Kai Zhang, Fujun Luan, Kalyan Sunkavalli, Qixing Huang, et al. Rayzer: A self-supervised large view synthesis model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025a.

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*, 2025b.

Haian Jin, Hanwen Jiang, Hao Tan, Kai Zhang, Sai Bi, Tianyuan Zhang, Fujun Luan, Noah Snavely, and Zexiang Xu. Lvsm: A large view synthesis model with minimal 3d inductive bias. In *Proceedings of the International Conference on Learning Representations*, 2025.

Mohammad Mahdi Johari, Yann Lepoittevin, and François Fleuret. Geonerf: Generalizing nerf with geometry priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Gyeongjin Kang, Seungkwon Yang, Seungtae Nam, Younggeun Lee, Jungwoo Kim, and Eunbyung Park. Multi-view pyramid transformer: Look coarser to see broader. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

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

Evan Kim, Hyunwoo Ryu, Thomas W Mitchel, and Vincent Sitzmann. Scaling view synthesis transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not all patches are what you need: Expediting vision transformers via token reorganizations. In *Proceedings of the International Conference on Learning Representations*, 2022.

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.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of the European Conference on Computer Vision*, 2020.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in Neural Information Processing Systems*, 2021.

Mehdi SM Sajjadi, Henning Meyer, Etienne Pot, Urs Bergmann, Klaus Greff, Noha Radwan, Suhani Vora, Mario Lučić, Daniel Duckworth, Alexey Dosovitskiy, et al. Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Mehdi SM Sajjadi, Aravindh Mahendran, Thomas Kipf, Etienne Pot, Daniel Duckworth, Mario Lučić, and Klaus Greff. Rust: Latent neural scene representations from unposed imagery. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

You Shen, Zhipeng Zhang, Yansong Qu, Xiawu Zheng, Jiayi Ji, Shengchuan Zhang, and Liujuan Cao. Fastvggt: Training-free acceleration of visual geometry transformer. *arXiv preprint arXiv:2509.02560*, 2025.

Zhijian Shu, Cheng Lin, Tao Xie, Wei Yin, Ben Li, Zhiyuan Pu, Weize Li, Yao Yao, Xun Cao, Xiaoyang Guo, et al. Litevggt: Boosting vanilla vggt via geometry-aware cached token merging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Stanislaw Szymanowicz, Minghao Chen, Jianyuan Wang, Christian Rupprecht, and Andrea Vedaldi. Lagernvs: Latent geometry for fully neural real-time novel view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Philippe Tillet, H. T. Kung, and David D. Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019. URL https://dl.acm.org/doi/10.1145/3315508.3329973.

Chung-Shien Brian Wang, Christian Schmidt, Jens Piekenbrinck, and Bastian Leibe. Block-sparse global attention for efficient multi-view geometry transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026a.

Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Weitian Wang, Lukas Meiner, Rai Shubham, Cecilia De La Parra, and Akash Kumar. Httm: Head-wise temporal token merging for faster vggt. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026b.

Zhengqing Wang, Yuefan Wu, Jiacheng Chen, Fuyang Zhang, and Yasutaka Furukawa. Clift: Compressive light-field tokens for compute efficient and adaptive neural rendering. *Advances in Neural Information Processing Systems*, 2026c.

Haofei Xu, Anpei Chen, Yuedong Chen, Christos Sakaridis, Yulun Zhang, Marc Pollefeys, Andreas Geiger, and Fisher Yu. Murf: Multi-baseline radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Haofei Xu, Songyou Peng, Fangjinhua Wang, Hermann Blum, Daniel Barath, Andreas Geiger, and Marc Pollefeys. Depthsplat: Connecting gaussian splatting and depth. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

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 *Proceedings of the International Conference on Learning Representations*, 2024.

Botao Ye, Boqi Chen, Haofei Xu, Daniel Barath, and Marc Pollefeys. Yonosplat: You only need one model for feedforward 3d gaussian splatting. In *Proceedings of the International Conference on Learning Representations*, volume 2026, 2026.

Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Kai Zhang, Sai Bi, Hao Tan, Yuanbo Xiangli, Nanxuan Zhao, Kalyan Sunkavalli, and Zexiang Xu. Gs-lrm: Large reconstruction model for 3d gaussian splatting. In *Proceedings of the European Conference on Computer Vision*, 2024.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William Freeman, and Hao Tan. Test-time training done right. In *Proceedings of the International Conference on Learning Representations*, 2026.

Qitao Zhao, Hao Tan, Qianqian Wang, Sai Bi, Kai Zhang, Kalyan Sunkavalli, Shubham Tulsiani, and Hanwen Jiang. E-rayzer: Self-supervised 3d reconstruction as spatial visual pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Chen Ziwen, Hao Tan, Kai Zhang, Sai Bi, Fujun Luan, Yicong Hong, Li Fuxin, and Zexiang Xu. Long-lrm: Long-sequence large reconstruction model for wide-coverage gaussian splats. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

A Additional Method Analysis

A.1 Quantization Sensitivity of the First Renderer Block

We keep the Linear layers of the first renderer block in BF16 while quantizing blocks 1–5 to FP8. We analyze the sensitivity underlying this mixed-precision design through numerical perturbation analysis and block-wise FP8 quantization.

Perturbation sensitivity. For each renderer block l, we add a random Gaussian perturbation δx to either the target or source stream, normalized such that $\|\delta x\|_2 = \epsilon \|x\|_2$ and measure the resulting change in the rendered image. We define the perturbation gain as

$$G_l = \frac{\|I_l^{\text{pert}} - I^{\text{clean}}\|_2}{\|I^{\text{clean}}\|_2} \frac{1}{\epsilon}, \tag{5}$$

where I^{clean} and I_l^{pert} denote the outputs without and with perturbation at block l, respectively. A gain larger than one indicates that the injected error is amplified by the remaining renderer, whereas a gain below one indicates attenuation.

As shown in table 7, the first renderer block is substantially more sensitive to numerical perturbations than the later blocks. At $\epsilon = 10^{-2}$, perturbing the target stream at block 0 gives a gain of 3.01, compared with 0.48, 0.48, 0.36, 0.29, and 0.26 for blocks 1–5, respectively. A similar trend is observed in the source-conditioning stream, where block 0 again exhibits the largest sensitivity. This behavior is consistent across both perturbation magnitudes considered in our analysis. These results indicate that numerical errors introduced in the first renderer block propagate much more strongly to the final prediction, whereas perturbations introduced in later blocks are largely attenuated.

Block-wise FP8 quantization. We next verify whether this sensitivity translates to actual FP8 degradation and the corresponding results are provided in table 8. We independently quantize the Linear layers of each renderer block to FP8 while keeping all other blocks in BF16. Quantizing block 1 alone causes a substantial 1.75 dB PSNR drop, whereas individually quantizing blocks 2–6 results in only 0.005–0.12 dB degradation. We further perform the complementary experiment by keeping one block in BF16 while quantizing all remaining blocks. Keeping block 1 in BF16 and quantizing blocks 2–6 limits the PSNR drop to only 0.19 dB. In contrast, keeping any of blocks 2–6 in BF16 while quantizing the remaining blocks still results in a 1.78–1.87 dB drop, close to the 1.87 dB degradation when all blocks are quantized. These results are consistent with the perturbation analysis and identify block 1 as the primary source of FP8 sensitivity. We therefore retain the Linear layers of block 1 in BF16 while executing those of blocks 2–6 in FP8.

A.2 Mixed-Precision FP8 Attention

On the RTX 5090 used for our evaluation, a highly optimized cuDNN BF16 attention path is available, whereas no comparably optimized native FP8 attention path is available. We therefore implement FP8 attention using Triton (Tillet et al., 2019). For a controlled precision comparison, we additionally implement a BF16 counterpart using the same Triton kernel structure, tensor shapes, tiling strategy, and attention computation path. The Triton BF16 and FP8 variants therefore differ

Table 7: Sensitivity of renderer blocks to numerical perturbations at two perturbation magnitudes. Perturbation gain measures the relative output change normalized by ϵ.

	$\epsilon = 10^{-4}$		$\epsilon = 10^{-2}$	
Block	**Target**	**Source**	**Target**	**Source**
0	**1.117 ± 0.437**	**0.641 ± 0.364**	**3.011 ± 1.182**	**0.809 ± 0.374**
1	0.344 ± 0.111	0.266 ± 0.098	0.477 ± 0.140	0.489 ± 0.161
2	0.373 ± 0.123	0.188 ± 0.059	0.476 ± 0.136	0.427 ± 0.135
3	0.277 ± 0.091	0.165 ± 0.037	0.361 ± 0.089	0.345 ± 0.086
4	0.202 ± 0.063	0.037 ± 0.007	0.289 ± 0.056	0.151 ± 0.017
5	0.166 ± 0.039	0.007 ± 0.002	0.258 ± 0.034	0.100 ± 0.012

Table 8: Block-wise sensitivity to FP8 quantization. The BF16 baseline achieves 26.33 dB. #FP8 counts the number of FP8 operations. Left: only the indicated block is quantized to FP8. Right: all renderer blocks except the indicated block are quantized to FP8. ΔPSNR is measured relative to the BF16 baseline.

Only block l quantized				**All except block l quantized**			
Block	**#FP8**	**PSNR**	**ΔPSNR**	**Block**	**#FP8**	**PSNR**	**ΔPSNR**
1	16	**24.58**	**−1.75**	1	75	**26.14**	**−0.19**
2	16	26.21	−0.12	2	75	24.55	−1.78
3	16	26.29	−0.04	3	75	24.47	−1.86
4	16	26.32	−0.01	4	75	24.47	−1.86
5	16	26.33	0.0	5	75	24.46	−1.87
6	10	26.33	0.0	6	81	24.46	−1.87

Table 9: Module-wise sensitivity to FP8 attention. Each row quantizes only the indicated attention module to FP8 while keeping all remaining attention modules in BF16. ΔPSNR is measured relative to the all-BF16 attention baseline (26.13 dB). Bl denotes renderer block l. Cross-Attn$_x$ and Cross-Attn$_{\mathrm{rec}}$ denote the target-from-source and source-from-target directions of our bidirectional cross-attention, respectively.

Attention Module	**PSNR**	**ΔPSNR**
B0 Cross-Attn$_{\mathrm{rec}}$	25.26	−0.87
B2 Cross-Attn$_x$	25.60	−0.53
B3 Cross-Attn$_x$	25.85	−0.28
B0 Self-Attn	26.05	−0.08
B0 Cross-Attn$_x$	26.10	−0.02
B3 Self-Attn	26.11	−0.02
B1 Cross-Attn$_{\mathrm{rec}}$	26.12	−0.01
B2 Self-Attn	26.12	−0.01
Remaining 9 modules	≈26.13	≤0.00

only in the low-precision matrix multiplications and the quantization/scaling operations required for FP8.

Attention-wise sensitivity. We first analyze the sensitivity of each attention module by quantizing only that module to FP8 while keeping all remaining attention operations in BF16. As shown in table 9, the degradation is highly concentrated in three modules. Quantizing the reconstruction cross-attention in block 0 reduces PSNR by 0.87 dB, while the target cross-attention modules in blocks 2 and 3 incur drops of 0.53 and 0.28 dB, respectively. In contrast, every other attention module causes at most 0.08 dB degradation, and nine of them exhibit changes below 0.02 dB. This strong variation motivates retaining only the three most sensitive modules in BF16.

Sensitivity-aware FP8 attention. Based on the sensitivity analysis, we retain the three most sensitive attention modules in BF16 and execute all remaining attention modules in FP8. Table 10 compares this mixed-precision configuration with all-FP8 attention, the Triton BF16 variant, and the optimized cuDNN BF16 baseline. Quantizing all attention modules to FP8 causes a substantial 1.61 dB PSNR drop, whereas retaining only the three sensitive modules in BF16 reduces the degradation to 0.15 dB. The mixed configuration also improves SSIM from 0.775 to 0.827 and LPIPS from 0.127 to 0.103 compared with all-FP8 attention. Within the controlled Triton implementation, mixed-precision FP8 increases throughput from 125 to 129 FPS over BF16 while largely preserving reconstruction quality, demonstrating that FP8 attention is viable when sensitive modules are selectively protected. However, the optimized cuDNN BF16 attention path available on the RTX 5090 achieves a higher throughput of 144 FPS. Therefore, mixed-precision FP8 attention does not provide an end-to-end throughput advantage on our RTX 5090 deployment platform, and we retain cuDNN BF16 attention in the final model. This result reflects the currently available backend rather than an inherent limitation of FP8 attention.

Table 10: Effect of sensitivity-aware mixed-precision FP8 attention. *All FP8* quantizes all attention modules, while *Mixed FP8* retains the three most sensitive modules in BF16. ΔPSNR is measured relative to the cuDNN BF16 baseline. Triton BF16 serves as a control using the same Triton attention implementation without FP8 quantization.

Attention	PSNR	ΔPSNR	SSIM	LPIPS	FPS
cuDNN BF16	26.13	–	0.831	0.101	144
Triton BF16	26.13	0.00	0.831	0.101	125
All FP8	24.51	−1.61	0.775	0.127	128
Mixed FP8	25.98	−0.15	0.827	0.103	129

Table 11: Comparison of low-precision numerical formats. W and A denote the formats used for Linear-layer weights and activations, respectively. ΔPSNR is measured relative to our E4M3/E4M3 configuration. † denotes fake quantization, as E5M2-weight GEMMs are not supported by our native `torch._scaled_mm` inference path.

W	A	Scaling	PSNR	ΔPSNR	SSIM	LPIPS
BF16	BF16	–	26.33	+0.20	0.836	0.097
E4M3	E4M3	Row	**26.14**	–	**0.831**	**0.101**
INT8	INT8	Row	26.05	−0.08	0.828	0.107
E4M3	E5M2	Row	25.71	−0.42	0.819	0.111
E5M2†	E4M3	Row	25.85	−0.29	0.822	0.106
E5M2†	E5M2	Row	25.45	−0.68	0.810	0.116
INT8	INT8	Tensor	19.62	−6.51	0.563	0.223

A.3 Choice of Quantization Format

We compare alternative 8-bit numerical formats in table 11. Our E4M3/E4M3 configuration achieves 26.14 dB, while replacing E4M3 activations with E5M2 reduces PSNR by 0.42 dB. Although E5M2 provides a wider dynamic range through additional exponent bits, its lower mantissa precision results in coarser representation within a given exponent range. Under our fine-grained scaling scheme, where each quantization group spans a relatively limited numerical range, this additional dynamic range provides little benefit and preserving finer local precision is more important. Consistent with this observation, fine-grained INT8 remains competitive at 26.05 dB despite having no floating-point exponent or mantissa, as its uniformly spaced quantization levels can represent values densely within each locally scaled range. In contrast, tensor-wise INT8 drops substantially to 19.62 dB, further demonstrating that the effectiveness of INT8 relies on sufficiently fine-grained scaling. We additionally evaluate E5M2 weights using fake quantization, since E5M2-weight GEMMs are not supported by our native `torch._scaled_mm` inference path. These variants also perform worse than E4M3/E4M3. Overall, the results suggest that fine-grained local precision is more important than a large representable dynamic range for our renderer, motivating our use of E4M3 with per-output-channel weight scaling and per-token activation scaling.

B Additional Experimental results

B.1 Results at Higher Resolution

In this section, we evaluate our method at a higher rendering resolution of 512×512. As shown in table 12, the overall trend remains consistent with the 256×256 results. While the throughput of LagerNVS (Szymanowicz et al., 2026) decreases rapidly as the number of input views increases, our full model maintains nearly constant rendering speed at approximately 140 FPS across 32, 64, and 128 views. At 128 views, our method achieves 143 FPS compared with 1.8 FPS for LagerNVS, while also improving PSNR from 22.51 to 23.16 dB. Without pruning and quantization, our model also consistently outperforms LagerNVS in reconstruction quality, indicating that the architectural improvement is preserved at the higher resolution. These results demonstrate that the efficiency and scalability of our approach extend to high-resolution rendering.

Table 12: A Comparison against LagerNVS (Szymanowicz et al., 2026) across different numbers of input views at 512×512 resolution. FPS is measured using an RTX 5090 GPU.

Method	**32 views**				**64 views**				**128 views**			
	PSNR ↑	**SSIM** ↑	**LPIPS** ↓	**FPS** ↑	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**FPS** ↑	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**FPS** ↑
LagerNVS	24.30	0.755	0.195	6.5	23.59	0.720	0.202	3.5	22.51	0.673	0.229	1.8
+ PQ	21.23	0.637	0.267	78	20.73	0.613	0.283	78	20.10	0.589	0.311	78
Ours w/o PQ	24.70	0.764	0.202	25.2	24.76	0.755	0.200	12.7	23.98	0.717	0.220	6.5
Ours	23.91	0.741	0.218	143	23.92	0.733	0.216	140	23.16	0.699	0.233	143

Table 13: Quantitative comparison on the DL3DV (Ling et al., 2024) dataset at a resolution of 224×224 with 6, 12, and 24 input views. Baseline results are sourced from 2Xplat (Jeong et al., 2026). Under the 6-views-context image setting, *Ours* uses source token pruning and quantization while omitting view selection.

Method	**6 views**			**12 views**			**24 views**		
	PSNR	**SSIM**	**LPIPS**	**PSNR**	**SSIM**	**LPIPS**	**PSNR**	**SSIM**	**LPIPS**
MVSplat Chen et al. (2024)	22.66	0.760	0.173	21.29	0.709	0.224	19.98	0.662	0.269
DepthSplat Xu et al. (2025)	23.42	0.797	0.136	21.91	0.753	0.179	20.09	0.690	0.240
YoNoSplat Ye et al. (2026)	24.72	0.817	0.139	23.29	0.773	0.177	22.66	0.758	0.192
2Xplat Jeong et al. (2026)	25.22	0.827	0.150	24.59	0.813	0.169	24.35	0.811	0.174
Ours w/o PQ	**27.13**	**0.866**	**0.077**	**27.79**	**0.859**	**0.077**	**26.58**	**0.848**	**0.081**
Ours	26.93	0.862	0.081	27.45	0.853	0.083	26.19	0.839	0.089

B.2 Comparison against 3D Gaussian Splatting-based approach

In this section, we additionally evaluate our model with 3D Gaussian Splatting (Kerbl et al., 2023)-based methods on DL3DV at 224×224 resolution, using 6, 12, and 24 input views, following the evaluation protocol of Yonosplat (Ye et al., 2026). This experiment broadens our evaluation beyond Transformer-based renderers by comparing against MVSplat (Chen et al., 2024), Depthsplat (Xu et al., 2025), Yonosplat (Ye et al., 2026), and 2Xsplat (Jeong et al., 2026). For this comparison, we additionally fine-tune our model at 224×224 resolution and evaluate it using 6, 12, and 24 input views.

As shown in table 13, our model achieves higher PSNR and SSIM and lower LPIPS than the evaluated Gaussian splatting baselines across all three input-view settings. The advantage persists after token pruning and post-training quantization: even retaining only 50% of the source tokens. our model outperforms all evaluated baselines on all three metrics.

B.3 Effect of post-encoding view selection

Table 14 evaluates how aggressively the renderer context can be reduced after jointly encoding all 64 source views. Importantly, this setting differs from encoding only the selected views from the outset: although only N views are passed to the renderer, their representations are obtained after joint encoding with the full 64-view context. For our DA3-based model, the resulting representations remain highly robust to post-encoding selection. Retaining 16 views reduces the renderer's source-token count by $4\times$ while improving PSNR from 26.73 to 26.90 dB, and our default budget of eight views reduces the renderer context by $8\times$ with only a 0.24 dB PSNR drop. In contrast, the VGGT-based reference degrades monotonically as fewer jointly encoded views are retained, losing 2.85 dB at 16 views and 3.55 dB at eight views. These results support post-encoding view selection in our model, showing that DA3-based representations remain effective even when only a small subset of the jointly encoded source views is forwarded to the renderer.

B.4 Ablation on the number of selected views

We ablate the number of retained source views in table 15. Retaining more views improves reconstruction quality but reduces rendering throughput: increasing the budget from 4 to 10 views raises PSNR from 25.25 to 26.35 dB while reducing FPS from 209 to 120. We use 8 retained views by default, which provides a good balance between quality and efficiency.

Table 14: Post-encoding view selection on all 140 test scenes at 256×256, without token pruning or TTO. All 64 source views are jointly encoded, and only N selected views are passed to the renderer. R_N denotes the PSNR change (dB) relative to retaining all 64 views.

	VGGT-based reference				DA3-based (Ours)			
N	PSNR	SSIM	LPIPS	R_N	PSNR	SSIM	LPIPS	R_N
64	25.26	0.798	0.108	0.00	26.73	0.840	0.097	0.00
32	23.55	0.730	0.127	−1.71	26.88	0.847	0.093	+0.15
16	22.41	0.685	0.147	−2.85	**26.90**	**0.849**	**0.090**	**+0.17**
8	21.71	0.656	0.164	−3.55	26.49	0.840	0.094	−0.24
4	21.23	0.638	0.179	−4.02	25.73	0.820	0.106	−1.00

Table 15: Ablation of view selection.

# Retained views	PSNR	SSIM	LPIPS	FPS
10	**26.35**	**0.837**	0.098	120
8	26.13	0.831	0.101	143
6	25.80	0.822	0.106	158
4	25.25	0.805	0.117	**209**

B.5 Test-time Optimization Details

Since FP8 quantization is non-differentiable, we simulate the quantized Linear layers with fake quantization and use the straight-through estimator (STE) (Bengio et al., 2013) to propagate gradients through the quantizer:

$$\frac{\partial Q(x)}{\partial x} \approx I, \tag{6}$$

where $Q(\cdot)$ denotes the FP8 quantization operator. At inference, the fake-quantized Linear layers are replaced by the same native FP8 GEMMs used in our main experiments.

We use a LoRA rank of 8, corresponding to approximately 1.5M trainable parameters, and optimize for 500, 1,000, 1,500, and 2,000 iterations at 32, 64, 128, and 256 input views, respectively.

B.6 Encoding and Decoding Time

We separately measure encoding and decoding latency under the 64-view setting. FlashLVSM requires 4,288 ms for encoding, compared with 5,080 ms for LagerNVS, while the difference is substantially larger during decoding: 7.0 ms versus 176.94 ms, respectively. This corresponds to a $25.3\times$ reduction in decoding latency, highlighting that the primary efficiency gain of FlashLVSM comes from its lightweight, fixed-budget rendering stage. Since source features are encoded once and reused for multiple target views, this reduction directly improves novel-view rendering efficiency.

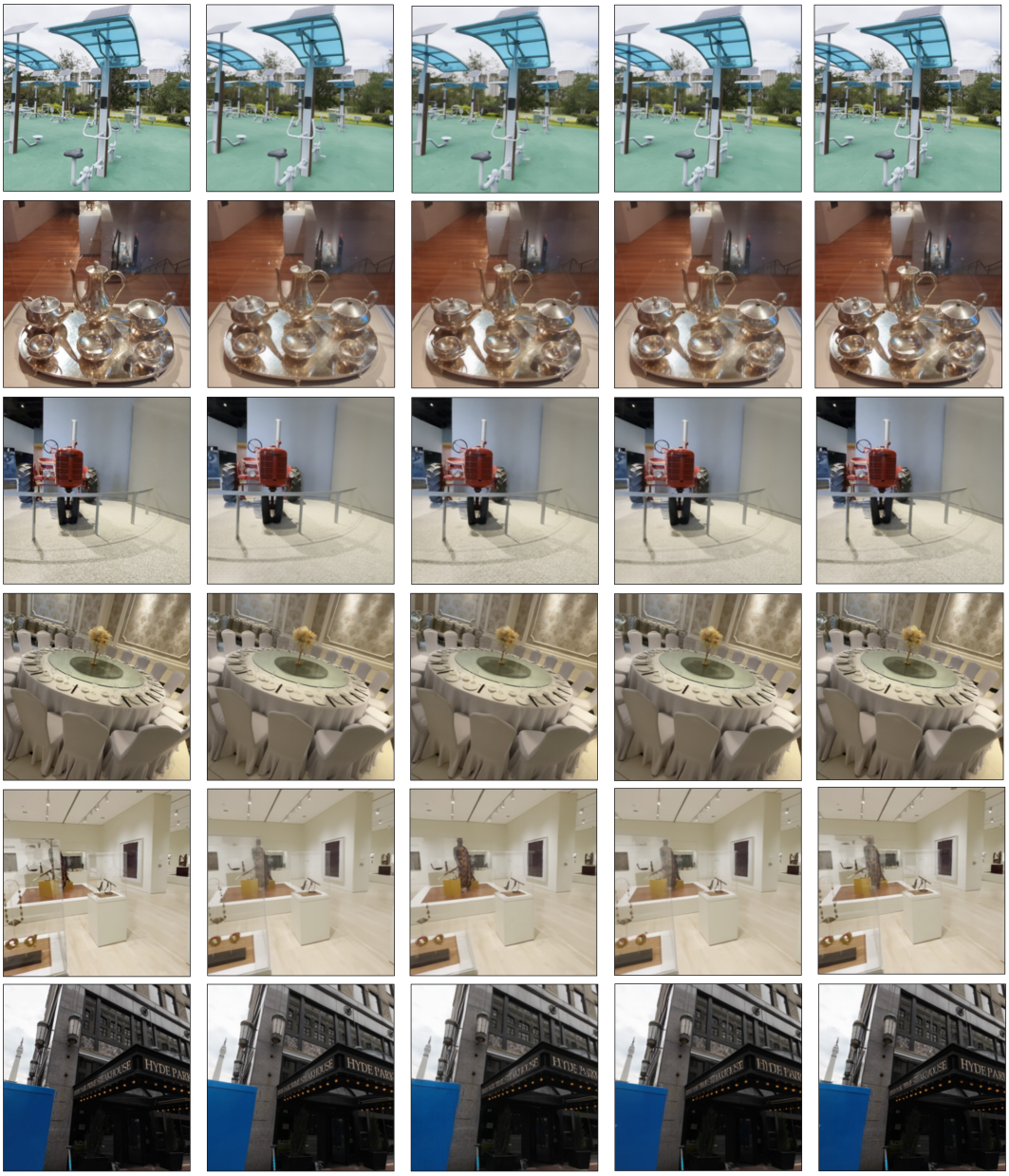

Figure 6: Additional qualitative results.

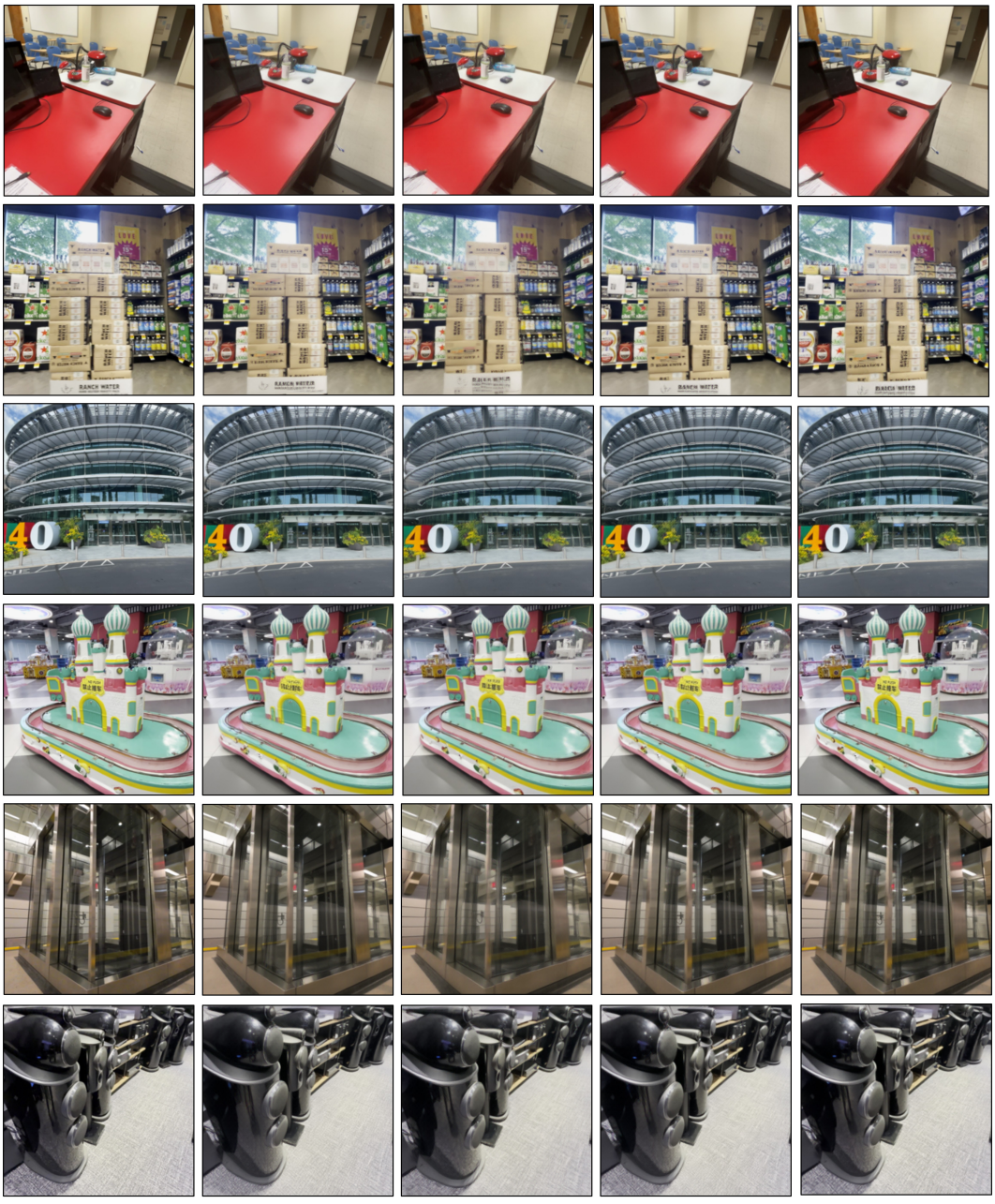

Figure 7: Additional qualitative results.