



# EcoSplat: Efficiency-controllable Feed-forward 3D Gaussian Splatting from Multi-view Images

## Supplementary Material

### A. Implementation Details

EcoSplat is implemented in PyTorch [30] using the gsplat rasterization engine [54]. Our training is conducted on 4 NVIDIA A100 GPUs (40GB each). Following prior works [24, 53], the ViT encoder/decoder and  $F_\mu$  are initialized with pretrained weights from MAST3R [32]. For both PGT and IGF stages, we set the number of iterations to 200K. The number of decoder blocks  $m$  is set to 4. We set  $\lambda_{io}$  to 0.1. For PLGC, we set  $\lambda_{decay}$  and  $S$  to 0.05 and 1000, respectively. For Eq. 12 and Eq. 13, we set  $s$  and  $T$  to 64 and 0.2, respectively.

### B. Demo Video

We provide a demo video, named ‘demo\_EcoSplat.mp4’, that showcases more abundant visualization results. The demo consists of two parts: (i) efficiency-controllable NVS results on the RE10K test dataset [61], and (ii) efficiency-controllable NVS results on the ACID test dataset [34] with cross-dataset generalization. For both parts, we train each model using the RE10K training dataset [61]. We use 24 input views for inference. For each video, we first reduce the number of primitives for each method from 100% of the pixel-aligned Gaussians to 5%, and then present the NVS results for all intermediate views between the first and last indices of the input images using only 5% of the primitives. Note that GGN [59] cannot reduce the number of primitives below 15% due to its architectural limitations for pruning. The demo video demonstrates that EcoSplat faithfully controls the number of primitives while preserving perceptual quality, whereas the other baselines exhibit a substantial drop in visual quality.

### C. Additional Experimental Results

#### C.1. Efficiency Comparisons

In Table 5, we present a detailed efficiency comparison of all feed-forward 3DGS methods evaluated in Table 2. The reported metrics include ‘Recon. Latency (s)’, ‘Render FPS’, ‘# GS’, and ‘Storage (MB)’, which respectively correspond to the time required to generate 3D Gaussians from input views, the rendering speed obtained from rasterizing the predicted 3D Gaussians, the number of output 3D Gaussians, and their memory footprint. As shown, EcoSplat substantially improves both rendering FPS and storage efficiency by predicting far fewer Gaussians. Moreover,

Methods	Recon. Latency (s)↓	Render FPS↑	# GS↓	Storage (MB)↓
MVSplat [11]	1.41	679	1573K	534
DepthSplat [50]	<b>0.56</b>	683	1573K	534
NoPoSplat [53]	2.65	715	1573K	534
SPFSplat [24]	0.61	710	1573K	534
GGN [59]	1.59	<b>987</b>	<b>512K</b>	<b>174</b>
AnySplat [26]	1.63	780	1259K	428
WorldMirror [36]	0.64	830	1020K	346
Ours <sub>5%</sub>	<b>0.52</b>	<b>1042</b>	<b>78K</b>	<b>27</b>
Ours <sub>40%</sub>	<b>0.52</b>	977	629K	214

Table 5. **Efficiency comparison.** We report the efficiency metrics of the existing methods under the 24-view setting. All evaluations are conducted on a single NVIDIA A100 GPU. To ensure a fair comparison of rendering FPS, we first generate and store the 3D Gaussian primitives for every method, and then render the target views using the same rasterization backend across all methods.

EcoSplat achieves the fastest reconstruction time among all feed-forward methods. Although NoPoSplat [53] uses a similar MAST3R [32] backbone, its architecture requires performing cross-attention between the first input view and each remaining view independently, resulting in substantial computational overhead. Following SPFSplat [24], EcoSplat adopts a global attention mechanism that aggregates feature information from all views. Furthermore, whereas SPFSplat predicts Gaussian parameters sequentially across views, EcoSplat predicts all per-view Gaussian primitives in parallel. This parallel implementation significantly reduces reconstruction latency, making EcoSplat the most efficient method overall.

#### C.2. Additional NVS Results under Controlled Number of Primitives

We further evaluate all methods under a more challenging test setup. Instead of randomly sampling input and target views, we require each target view to be at least 20 frames away from its nearest input view, which forces the models to synthesize substantially more extreme novel viewpoints. We additionally include per-scene optimization methods that support controllable primitive counts, such as 3DGS-MCMC [29], PUP 3D-GS [20], and Lightgaussian [15], to compare them with feed-forward approaches. For 3DGS-MCMC [29], we train the model for 30K iterations. For PUP 3D-GS [20] and Lightgaussian [15], we first optimize the 3DGS representation for 10K iterations,

Methods	5% Gaussians			10% Gaussians			40% Gaussians			70% Gaussians		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
3DGS-MCMC [29]	20.34	0.698	0.316	20.50	0.710	0.304	20.28	0.715	0.313	19.97	0.710	0.329
PUP 3D-GS [20]	15.72	0.600	0.503	16.69	0.635	0.452	17.01	0.653	0.422	17.30	0.658	0.417
Lightgaussian [15]	15.82	0.605	0.477	16.44	0.631	0.459	17.57	0.663	0.420	17.43	0.658	0.415
SPFSplat [24] + [15] + 1K iter.	22.08	0.774	0.314	22.79	0.799	0.272	23.92	0.830	0.218	24.38	0.838	0.206
SPFSplat [24] + [20] + 1K iter.	22.67	0.803	0.259	23.39	0.821	0.229	<u>24.13</u>	<u>0.835</u>	<u>0.210</u>	<u>24.50</u>	<u>0.840</u>	0.205
SPFSplat [24] + [15] + 10K iter.	22.08	0.774	0.314	22.79	0.799	0.272	24.07	0.807	0.232	24.06	0.804	0.244
SPFSplat [24] + [20] + 10K iter.	<u>23.73</u>	<u>0.819</u>	<u>0.224</u>	<u>23.99</u>	<u>0.822</u>	<u>0.219</u>	24.05	0.807	0.239	24.07	0.802	0.245
AnySplat [26]	7.92	0.268	0.574	10.97	0.440	0.469	18.73	0.676	0.253	20.13	0.714	<u>0.189</u>
WorldMirror [36]	7.31	0.222	0.627	9.74	0.381	0.526	18.37	0.683	0.287	20.55	0.736	0.203
MVSplat [11] + [15]	4.75	0.053	0.727	5.32	0.098	0.697	9.32	0.368	0.509	10.84	0.430	0.583
MVSplat [11] + [20]	4.86	0.065	0.688	5.65	0.114	0.663	9.89	0.386	0.491	10.84	0.429	0.455
SPFSplat [24] + [15]	6.64	0.243	0.647	7.34	0.285	0.612	10.51	0.519	0.415	13.67	0.669	0.274
SPFSplat [24] + [20]	5.31	0.136	0.610	6.37	0.224	0.564	11.54	0.556	0.367	14.75	0.690	0.252
EcoSplat (Ours)	<b>24.80</b>	<b>0.843</b>	<b>0.157</b>	<b>24.87</b>	<b>0.845</b>	<b>0.156</b>	<b>25.02</b>	<b>0.847</b>	<b>0.151</b>	<b>24.90</b>	<b>0.845</b>	<b>0.151</b>

Table 6. **Addition quantitative comparison of NVS under controlled numbers of primitives on the RE10K dataset [61] with 24 input views.** We evaluate the baselines on more challenging test views than those used in Table 1. Moreover, we further compare EcoSplat against additional baselines, including per-scene optimization methods and post-optimization variants of the cascaded methods. ‘+ 1K iter.’ and ‘+ 10K iter.’ indicate that we further optimize the parameters of the output 3D Gaussians from each cascaded method for an additional 1K and 10K iterations, respectively.

prune the primitives using their respective strategies, and then resume training for an additional 10K iterations. These per-scene optimization methods require approximately 10 minutes of reconstruction time. Moreover, we finetune the 3D Gaussian parameters predicted by the cascaded baselines that combine SPFSplat [24] with post-pruning methods [15, 20], and report results after 1K and 10K finetuning iterations, respectively. The results demonstrate that our feed-forward EcoSplat significantly outperforms all competing methods, including per-scene optimization approaches, by large margins. EcoSplat exhibits significantly less overfitting to the input views and demonstrates greater robustness under large viewpoint shifts than the per-scene optimization approaches [15, 20, 29]. Although the cascade baselines (SPFSplat [24] + Lightgaussian [15], SPFSplat [24] + PUP 3D-GS [20]) show improvements after per-scene post-optimization, they still fall behind our EcoSplat without any further Gaussian finetuning. Note that per-scene post-optimization for these cascade baselines requires up to 1 minute of finetuning, adding substantial overhead to the feed-forward pipeline.

### C.3. Ablation Study on Architectural Design

In Table 7, we ablate the injection strategies of our learnable importance embedding  $R_i$  in Eq. 3, with the corresponding designs illustrated in Fig. 6-(a). In the ‘Cross-attention’ variant, each  $R_i$  interacts with its corresponding  $Z_i^{(\ell)}$  through cross-attention. The ‘Deep Add’ variant replaces this cross-attention operation with a simple additive fusion of  $R_i$  and  $Z_i^{(\ell)}$ . The fused features are then passed to a residual convolutional refinement block. Our final design, the ‘Shallow Add’ variant, fuses  $R_i$  with the output

of the residual convolutional refinement block applied to  $\{Z_i^{(\ell)}\}_{\ell=1}^m$  by simple addition. This refinement block’s architecture is shown in Fig. 6-(b). Among all variants, the ‘Shallow Add’ variant achieves the best balance of effectiveness and stability.

Variants	5% Gaussians			40% Gaussians		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Cross-attention	<u>24.71</u>	<b>0.828</b>	<b>0.179</b>	<u>24.90</u>	<u>0.831</u>	<u>0.168</u>
Deep Add	24.18	0.814	0.196	24.62	0.824	0.178
<b>Ours</b>	<b>24.72</b>	<u>0.822</u>	<u>0.183</u>	<b>25.11</b>	<b>0.835</b>	<b>0.164</b>

Table 7. **Ablation study of the learnable importance embedding injection on RE10K [61] with 24 input views.** The ‘Shallow Add’ injection strategy achieves superior performance compared to the ‘Cross-attention’ and ‘Deep Add’ variants.

### C.4. Additional Cross-Dataset Generalization NVS

We provide extended quantitative results for the 16-view and 20-view settings in Table 8, complementing the evaluations reported in Table 3.

Methods	16 Views				20 Views			
	PSNR↑	SSIM↑	LPIPS↓	# GS↓	PSNR↑	SSIM↑	LPIPS↓	# GS↓
MVSplat [11]	18.19	0.502	0.376	1048K	18.12	0.500	0.379	1310K
DepthSplat [50]	20.41	<u>0.713</u>	0.268	1048K	19.92	0.694	0.286	1310K
NoPoSplat [53]	22.30	0.668	0.286	1048K	22.24	0.666	0.288	1310K
SPFSplat [24]	<b>24.58</b>	<b>0.725</b>	<b>0.218</b>	1048K	<b>24.49</b>	<b>0.722</b>	<b>0.221</b>	1310K
GGN [59]	18.46	0.564	0.380	<u>391K</u>	18.32	0.560	0.384	<u>436K</u>
AnySplat [26]	21.89	0.615	0.262	861K	22.05	0.622	0.256	1050K
WorldMirror [36]	22.15	0.646	0.275	806K	22.20	0.650	0.275	978K
Ours <sub>5%</sub>	23.92	0.689	0.282	<b>52K</b>	23.95	0.690	0.282	<b>78K</b>
Ours <sub>40%</sub>	<u>24.17</u>	0.701	<u>0.250</u>	419K	<u>24.12</u>	<u>0.700</u>	<u>0.252</u>	524K

Table 8. **Cross-dataset generalization with 16 and 20 input views.** We train all methods on the RE10K dataset [61] and evaluate their zero-shot performance on the ACID dataset [34].

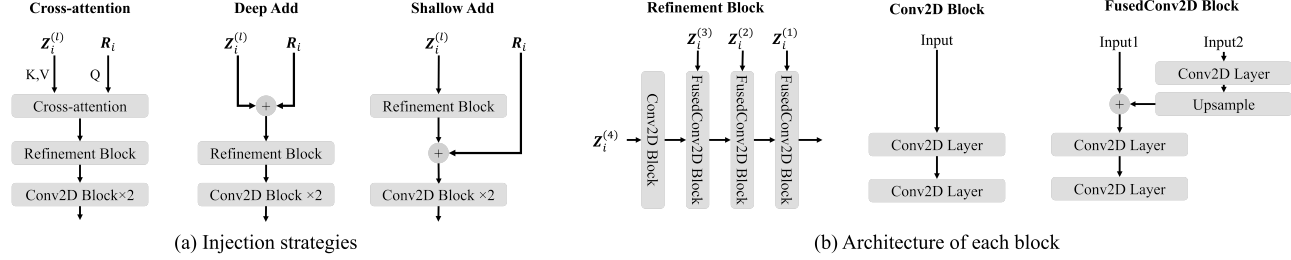


Figure 6. Importance embedding injection strategies.

## C.5. Long-LRM Results

We acknowledge the concurrent work Long-LRM [62], as discussed in the Related Works section (Sec. 2). Despite our extensive efforts to include this model in our comparison, as of 20 November 2025, no official training configuration or released code for the RE10K dataset has been made publicly available. We attempted to train Long-LRM by reproducing all available details and using the checkpoint provided by the authors. However, the absence of an official RE10K configuration and incomplete training specifications led to repeated training failures, and the resulting outputs were not reliable enough for a fair comparison. Therefore, although we made every reasonable effort to reproduce Long-LRM with the currently available public resources, we were unable to obtain valid experimental results. Once the official configuration becomes available, we will gladly include a full experimental comparison.

## C.6. Comparisons with FCGS

FCGS [12] encodes Gaussian parameters while keeping the primitive count unchanged, whereas EcoSplat reduces the count via pruning, making our approach orthogonal to FCGS. Following the taxonomy in the 3DGS.zip survey [1], these strategies correspond to “attribute compression” and “compaction,” respectively. We compare our 3% variant (Ours<sub>3%</sub>) with FCGS [12] applied to SPFSplat in Tab. 9. With a similar storage footprint, EcoSplat achieves superior NVS quality and significantly faster reconstruction. Furthermore, it should be noted that attribute compression only improves storage memory and requires a decompression operation during the rendering stage. Because the decompressed 3D Gaussian primitives match the dense count of the original pixel-aligned method, this results in lower rendering FPS, as shown in Tab. 5. Finally, because these methods are orthogonal, a complete efficiency pipeline could apply FCGS on top of EcoSplat for compounded memory compression.

## C.7. Comparisons with Zpressor

For Zpressor [47] comparisons, we present the results Table. 10. Due to its multi-view clustering, Zpressor cannot achieve arbitrary compaction ratios (e.g., 5%, 10%), but is

Methods	PSNR/SSIM/LPIPS	Storage	Recon. Latency
SPFSplat + FCGS	23.61/0.813/0.178	16 MB	0.61 + 30 sec
Ours <sub>3%</sub>	<b>24.61/0.839/0.167</b>	16 MB	<b>0.52 sec</b>

Table 9. Comparisons with FCGS.

restricted to a ratio set by the number of input images, such as  $\frac{2}{24} \approx 8.33\%$  or  $\frac{3}{24} = 12.5\%$ . With less Gaussians, Zpressor shows large performance drops due to the lack of adaptive controllability.

## D. Limitation and Future Work

Similar to prior feed-forward 3DGS frameworks [8, 11, 24, 26, 36, 50, 53], EcoSplat is currently designed for NVS of *static* scenes from multi-view images. Therefore, our model does not yet handle object deformation or dynamic scene reconstruction. Nevertheless, recent progress in *feed-forward* dynamic scene reconstruction [44, 49] have proposed a promising extension that finetunes the global cross-attention layers to learn temporal consistency in dynamic scenes. Inspired by this, our framework could be extended to handle dynamic scenes and jointly predict global static primitives and per-frame dynamic primitives. Moreover, like most feed-forward methods, we reconstruct only visible regions, leaving holes in uncaptured areas, as in Fig. 7. Video diffusion models can be used to inpaint the occlusions in the final renderings.

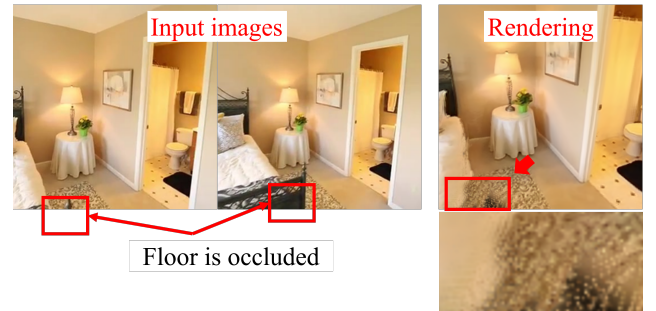


Figure 7. Limitation of rendering occluded regions.

	Methods	8.33% Gaussians	12.5% Gaussians	66.67% Gaussians	100% Gaussians
Tab. 1 Setting	ZPressor	13.03/0.510/0.438	21.62/0.791/0.211	<b>25.50/0.876/0.134</b>	24.25/ <b>0.852/0.165</b>
	EcoSplat (Ours)	<b>24.91/0.828/0.174</b>	<b>25.06/0.833/0.169</b>	25.00/0.833/0.165	<b>24.86/0.829/0.172</b>
Tab. 6 Setting	ZPressor	10.70/0.407/0.524	18.67/0.766/0.240	23.32/ <b>0.844/0.165</b>	22.31/0.823/0.187
	EcoSplat (Ours)	<b>24.58/0.837/0.167</b>	<b>24.69/0.840/0.166</b>	<b>24.83/0.843/0.152</b>	<b>24.60/0.838/0.167</b>

Table 10. Comparisons with Zpressor.