tttLRM: Test-Time Training for Long Context and Autoregressive 3D Reconstruction

Supplementary Material

Contents

A. Further Discussions

Effect of Scene Complexity on Fast Weights The memory of fast-weights has a fixed capacity and is bounded, especially in the autoregressive setting. Our empirical analysis on DL3DV scene labels indicates that higher scene complexity leads to degraded performance, as observed in outdoor vs. indoor scenes (PSNR: 24.45 vs. 24.96) and high- vs. low-frequency scenes (PSNR: 24.20 vs. 25.97). The memory capacity is also influenced by sequence length, where earlier inputs may be gradually forgotten as more tokens are processed.

Selective Update of Fast Weights in AR Setting Instead of updating the fast weights only according to current inputs, we can further use history states for selective update to mitigate drifiting. Inspired by [2], we explore a mechanism to prevent weight drift. Specifically, we approximate the diagonal of the Fisher information using an exponential moving average of squared gradients, as an estimate of parameter importance. Meanwhile, we maintain a sliding anchor via EMA to track the historical trajectory of the fast weights. After each gradient update, we apply elastic regularization based on parameter importance. Specifically, we leverage Fisher information for selective update, where parameters with high Fisher values that are important for the current input, are left parameters with high Fisher values unconstrained, while parameters with low Fisher values are pulled back toward the anchor. This encourages adaptation to the current input and suppresses drift in unimportant parameters. This training-free strategy can further improve our autoregressive model, and we envision it to be more effective by incorporating it into training for future work.

Table 1. Training-free selective update considering history fast weights can further enhance our AR model.

	PSNR↑	SSIM↑	LPIPS↓
w/o selective	24.81	0.814	0.225
w selective	**24.95**	**0.818**	**0.223**

Scaling to More Input Views With distributed training, tttLRM can be further scaled to hundreds of views given enough compute. For example, by finetuning our full model with more iterations on 128 input views (more than 1M tokens), it can achieve 26.80 PSNR.

Possible Usage of Attention Layers We deliberately avoid attention blocks in our model since it has quadratic complexity $O(N^2d)$ compared to our linear FLOPS $O(Nd^2)$ of LaCT blocks (N is the number of tokens and d is hidden dimension). Therefore, attention will bottleneck the computation with growing number of tokens and be very slow in our million-level token setting. As shown in Figure 1, even a 3-layer attention only will be slower than our 24-layer LaCT blocks from 2M tokens (256 views). With more compute, our model can easily scale to longer sequence and remain linear complexity.

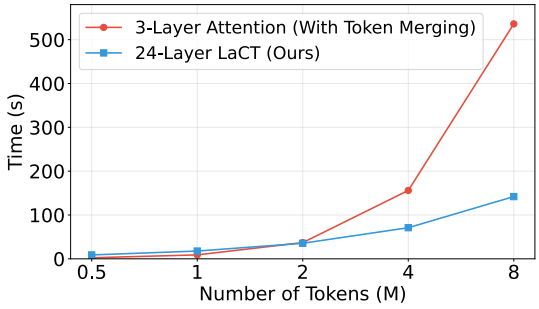

Figure 1. Time comparison of 3 Attention layers vs 24 layers of LaCT blocks under different numbers of tokens.

B. Experiment Details

B.1. Scene-level Training

We adopt a curriculum training strategy that progresses from low to high resolution, motivated by two main reasons. First, fast low-res pretraining enables the model to train with a large batch size with faster iteration time. Second, we found that even with pretrained TTT-LVSM [5] checkpoints at high-resolution (*i.e.* 960×540), the model cannot predict reasonable Gaussians at the beginning iterations, leading to excessive GPU memory usage due to the rendering of a large number of Gaussians.

For scene-level training, We train our model on three stages with 144×256, 288×512 and 540×960 resolution. For each stage, we resize the images to the target resolution, which all have the same aspect ratio as the original dataset. For all stages, we first determine a continuous range based on the start and end frames from the entire video sequence

from the dataset. The range is randomly sampled from 128 to 512 for each sample to ensure enough coverage of the scene. Then, we randomly sample 124 frames from this range, from which both input and target views will be further sampled. They are ensured to have overlap frames for stable training. We train the model across 16 to 64 input views. For training, we use the input views as the virtual views and found that provides the best results.

For the first stage, we train the model with a peak learning rate of $3e{-}4$ with 2K warmup steps and cosine decay. We use AdamW optimizer with betas $(0.9, 0.95)$ and weight decay 0.05. We train the model using a batch size of 128 for 80K steps, which is around 0.3T tokens. For the second stage, we finetune the model at the resolution of 288×512 with a peak learning rate of $5e{-}5$. We use a batch size of 64 to train 6K steps. For the final stage, we enable depth loss and opacity loss, training the model with 32 input views with 5K steps with peak learning rate $1e{-}5$ and batch size of 64. Finally, we train the model with 16 to 64 input views for another 1K steps. We prune 70% Gaussians with the smallest opacity for 64 views and 60% otherwise.

For autoregressive model training, we finetune our model on the final stage checkpoints for around another 3K iterations with peak learning rate $1e{-}4$ and batch size of 64. We train the model on input views from 8 to 64. Our models are trained on 64 Nvidia A100 80GB GPUs.

Besides, we use `gsplat` Python library for efficient Gaussian training. We enable `torch.compile` to accelerate computation, achieving roughly a 30% per-iteration speedup. To further optimize memory and stability, we implement gradient checkpointing [1] and mixed-precision training [3] with the `BFloat16` format. For Gaussian rendering, we utilize deferred backpropagation [4] to reduce GPU memory consumption. In addition, iterations with a gradient norm that exceeds 5.0 are skipped to improve training stability.

B.2. Object-level training

For the GS-based model, we use 8 views as input and another 8 views as supervision and use a patch size of 16×16. We firstly sample a set of 15 images (from 32 renderings) as a data point, from which we randomly select 8 input views and 8 supervision views independently. This sampling strategy encourages more overlap between input views and rendering views than directly sampling from 32 rendering views. We train on the resolution of 256×256 with a batch size of 512 for 80K iterations with a peak learning rate of $4e{-}4$. We then finetune on 512×512 with a batch size of 128 for another 10K iterations with a peak learning rate of $5e{-}5$. We further finetune on 1024×1024 with a batch size of 64 for another 4K iterations with a peak learning rate of $5e{-}5$.

For the triplane-based model, we use 4 views as input and another 4 views as supervision and use a patch size of 16×16. We train on the resolution of 256×256 with a batch size of 256 for 60K iterations and finetune on 512×512 with a batch size of 64 for another 20K iterations.

C. More results and Comparison

In Table 2, we show results where we combine our method with a few additional optimization steps. It demonstrates that the reconstructed model can be further improved with minimal optimization cost, surpassing both purely optimization-based methods and the previous state-of-the-art feed-forward method, Long-LRM, under the same post-optimization setup.

Notably, the quality of Long-LRM with 3-step post-optimization is still lower than our model *without* post-optimization, even though it requires more time to perform the optimization than our feedforward inference.

Table 2. More quantitative comparison on both DL3DV-140 and Tanks&Temples datasets under 32/64 input views. Our method surpasses previous feedforward methods and can further surpass optimization-based methods with a few steps post-optimization. **Note that Long-LRM trains a separate model for each input view, while we are a single model across all input views.** Our model can be **linearly accelerated with multiple GPUs**, here we report time on a single Nvidia A100 80GB GPU.

Views	Method	Time↓	DL3DV-140			Tanks&Temples		
			PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
32	3D GS_{30k}	13m	23.60	0.779	0.213	18.10	0.688	0.269
	Mip-$Splatting_{30k}$	13m	23.32	0.784	0.217	18.39	**0.700**	**0.262**
	Scaffold-GS_{30k}	16m	**24.77**	**0.805**	**0.205**	**18.41**	0.691	0.290
	Long-LRM (32v model)	**1s**	24.10	0.783	0.254	18.38	0.601	0.363
	Long-LRM (w/ 3-step optim)	12s	24.99	0.809	0.243	18.69	0.623	0.360
	Long-LRM (w/ 10-step optim)	37s	25.60	0.826	0.233	18.90	0.642	0.350
	Ours	7.2s	25.07	0.822	0.215	19.22	0.662	0.305
	Ours (w/ 3-step optim)	18s	25.86	0.842	0.208	19.57	0.687	0.300
	Ours (w/ 10-step optim)	42s	**26.37**	**0.854**	**0.201**	**19.78**	**0.704**	**0.291**
64	3D GS_{30k}	13m	26.55	0.852	**0.164**	20.78	**0.778**	**0.205**
	Mip-$Splatting_{30k}$	13m	26.29	0.850	0.166	20.08	0.759	0.220
	Scaffold-GS_{30k}	16m	**27.07**	**0.857**	0.173	**20.96**	0.768	0.240
	Long-LRM (64v model)	**3.7s**	24.63	0.799	0.243	19.11	0.627	0.346
	Long-LRM (w/ 3-step optim)	38.9s	25.74	0.833	0.225	19.69	0.659	0.333
	Long-LRM (w/ 10-step optim)	114s	26.72	0.852	0.212	20.03	0.681	0.320
	Ours	14.8s	25.95	0.844	0.195	20.31	0.700	0.274
	Ours (w/ 3-step optim)	47s	26.97	0.866	0.185	20.76	0.724	0.269
	Ours (w/ 10-step optim)	124s	**27.65**	**0.880**	**0.177**	**21.07**	**0.743**	**0.260**

References

[1] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 2

[2] Ziqiao Ma, Xueyang Yu, Haoyu Zhen, Yuncong Yang, Joyce Chai, and Chuang Gan. Fast spatial memory with scalable elastic test-time training. https://mars-tin.github.io/blogs/posts/elastic_ttt.html, 2025. Blog post. 1

[3] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael

Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. 2

[4] Kai Zhang, Nick Kolkin, Sai Bi, Fujun Luan, Zexiang Xu, Eli Shechtman, and Noah Snavely. Arf: Artistic radiance fields. In *European Conference on Computer Vision*, pages 717–733. Springer, 2022. 2

[5] Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T Freeman, and Hao Tan. Test-time training done right. *arXiv preprint arXiv:2505.23884*, 2025. 1