

# Appendix for Paper 12796:

## PAROAttention: Pattern-Aware ReOrdering for Efficient Sparse and Quantized Attention in Visual Generation Models

### 1 Experimental Results for Wan 2.1 Model

We present the results of applying PAROAttn and baseline sparsification and quantization methods to the Wan-2.1[4] 14B T2V model in Tab. 1, along with qualitative results in Fig. 1. Notably, when applying SpargeAttention[7], we observed numerical instability leading to NaN outputs; hence, its results are omitted from the table. These findings are consistent with those presented for the CogVideoX model in Table 1 of the main paper. We summarize our key observations as follows:

- (1) **PAROAttn consistently outperforms baseline sparsification methods across varying density.** As shown in Tab. 1, PAROAttn significantly outperforms the baseline method SparseVideoGen across different metrics and settings. Remarkably, PAROAttn with a 0.3 density still surpasses SparseVideoGen at 0.5 density.
- (2) **PAROAttn preserves both visual quality and content even under more aggressive sparsity.** As illustrated in Fig. 1, PAROAttn at 0.3 density produces frames nearly identical to the dense baseline. In contrast, PAROAttn at 0.5 density introduces noticeable degradation with changes in both content and style.
- (3) **PAROAttn achieves comparable performance to baseline quantization methods with higher speedups.** As demonstrated in Tab. 1 and Fig. 1, PAROAttn’s quantization scheme further compresses the *PV* computation to INT8/INT4 on top of the SageAttn baseline, maintaining performance while delivering greater speedup.

Table 1: **Performance of PAROAttention Wan 2.1 text-to-video generation on VBench prompts.** Baselines are evaluated using their official codebases. For fair comparison, we configure SparseVideoGen without skipping sparsification during the first 30% of timesteps.

Type	Method	Efficiency		Quality					
		Dense Rate / Bitwidth	Video Quality Metrics			FP Diff. Metrics			
			CLIPSIM↑	VQA↑	ΔFScore↓	FVD-FP16↓	PSNR↑	SSIM↑	CosSim↑
Sparse	FP16 Full Attn.	100.0%	0.215	93.49	0.000	0.000	∞	1.000	1.000
	SparseVideoGen (0.5)	50.0%	0.199	91.56	0.468	0.476	15.32	0.613	0.900
	PAROAttn (0.5)	50.0%	0.213	92.85	0.114	0.251	22.02	0.806	0.978
	SparseVideoGen (0.3)	30.0%	0.196	90.13	0.612	0.679	13.17	0.475	0.839
	PAROAttn (0.3)	30.0%	0.208	91.97	0.153	0.278	21.73	0.786	0.978
	SageAttn	QK (INT8), PV (FP16)	0.201	92.24	0.126	0.209	20.43	0.720	0.970
Quant	SageAttnV2	QK (INT4), PV (FP8)	0.200	88.53	1.260	0.749	17.86	0.678	0.954
	PAROAttn (INT8)	QK (INT8), PV (INT8)	0.213	92.89	0.128	0.362	20.13	0.706	0.967
	PAROAttn (INT4)	QK (INT4), PV (INT4)	0.206	89.77	0.896	0.412	19.30	0.741	0.965

### 2 Additional Qualitative Results and Analysis of Metrics Selection:

**Analysis of Additional Qualitative Results:** We present additional qualitative comparisons of sparsification methods, along with their corresponding metric scores, in Fig. 2. We compare PAROAttn against SpargeAttn and SparseVideoGen on the CogVideoX model under density levels of 50% and 30%. As shown in the figure, PAROAttn produces nearly identical frames at both density levels, whereas SpargeAttn and SparseVideoGen introduce noticeable blurriness and content distortion. In particular, SpargeAttn at 30% density exhibits prominent square-shaped color blocks, and the

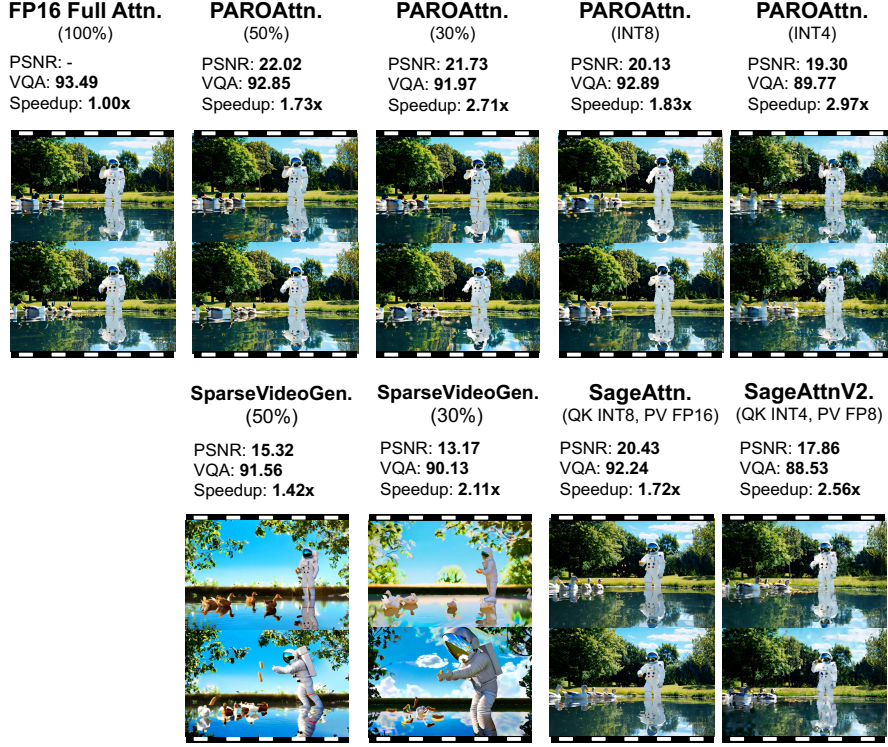


Figure 1: Qualitative results of Wan 2.1 model video generation.

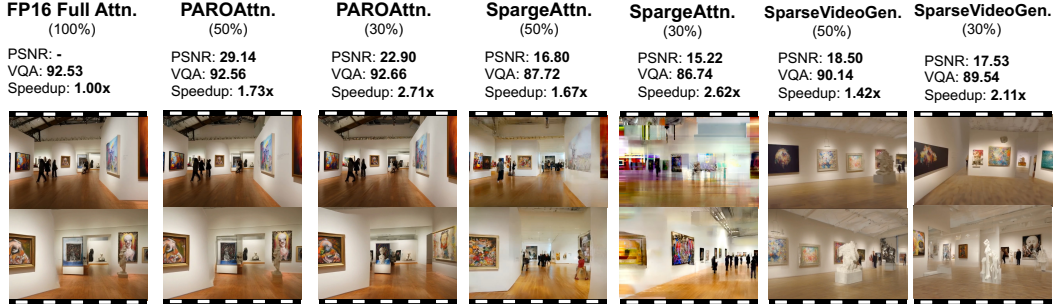


Figure 2: Additional qualitative results of sparsification for CogVideoX.

generated content becomes barely recognizable. We further analyze the corresponding changes in metric scores in the next paragraph.

**Findings and Recommendation of Metrics:** In Fig. 2, we observe that quality-related metric (VQA) and FP difference metric (PSNR) exhibit different trends as the dense rate varies. Specifically, for PAROAttn, the quality-related VQA metric remains stable even with 30% density, while the FP difference-related PSNR metric gradually decays. It reveals that the FP difference-related metrics are more challenging to maintain because they focus on low-level differences and can detect very minor detail changes that quality-related metrics might miss. As a result, they are suitable for scenarios where only minor differences are expected. However, they may not be reliable when comparing samples with significant differences, in which case quality-related metrics are more appropriate.

**Ablation study on skipping scheme in SparseVideoGen.** In the original SparseVideoGen paper and its official code release, sparsification is deliberately omitted for the first two Transformer blocks and the initial 30% of timesteps. In the main paper, for fair comparison, we do not adopt such “skipping” scheme for SparseVideoGen. We conduct an ablation study on the effect of this “skipping” scheme,

with results shown in Sec. 2 and Fig. 4. As observed, removing the skipping strategy leads to a notable degradation in generation quality (PSNR drops from 25.37 to 18.50), significant content distortion, and visibly blurred outputs. In contrast, PAROAttn maintains high generation quality even without skipping early timesteps or transformer blocks.

	FP16 Full Attn. (100%)	PAROAttn. (50%)	SparseVideoGen. (50%, w. Skip)	SparseVideoGen. (50%, w.o. Skip)
PSNR: -	PSNR: -	PSNR: 29.14	PSNR: 25.37	PSNR: 18.50
VQA: 92.53	VQA: 92.56	VQA: 92.56	VQA: 91.89	VQA: 90.14
Speedup: 1.00x	Speedup: 1.73x	Speedup: 1.42x	Speedup: 1.42x	

Method	PSNR $\uparrow$	SSIM $\uparrow$	CosSim $\uparrow$
SparseVideoGen (0.5, w.o. skip)	18.50	0.755	0.960
SparseVideoGen (0.5, w. skip)	25.37	0.871	0.984
PAROAttention (0.5)	29.14	0.936	0.997

Figure 3: Ablation of skipping timestep and transformer blocks for SparseVideoGen.

Figure 4: Qualitative examples from the ablation study on skipping timesteps and Transformer blocks in SparseVideoGen.

### 3 Additional Results for CUDA Kernel Efficiency Improvement

We provide detailed results comparing different CUDA kernel implementations in Sec. 3 and Sec. 3. The reported speedups are measured based on attention computation alone (excluding the QKVO projections), using a token length of 17,750—corresponding to 6-second 720P video generation with CogVideoX. Experiments for sparsification baselines are conducted on an NVIDIA A100 GPU (consistent with the supported hardware in the baseline code release), while quantization results are evaluated on an RTX 4090 GPU to leverage support for INT4 and FP8 quantization.

**Comparison of Latency Speedups:** We detailedly present the experimental results for PAROAttn’s CUDA kernel implementation with baseline sparsification and quantization methods in Sec. 3. We discuss the findings in Sec. 5.2 “Hardware Resource Savings” of the main paper in detail as follows:

- **Baseline Sparsification methods exhibit notable performance degradation.** Both the SparseVideoGen and SpargeAttn achieve PSNR lower than 20, even with a relative high density of 50%, worse than the PAROAttn with both sparsification and quantization applied (0.3+INT8 with PSNR 21.49, 0.5+INT4 with PSNR 24.34).
- **PAROAttn’s sparsification method can generate nearly identical frames, even with lower dense rate.** As seen in Sec. 3 and Fig. 2, the PAROAttn generation result highly resembles the full-precision baseline, while achieving substantial speedups.
- **The PARO token reordering is compatible with dynamic sparsification approaches.** In Sec. 3, the “SpargeAttention (0.3 + PARO)” means adopting the PARO token reordering with SpargeAttn, it notably improves the performance to exceeding the “SpargeAttn (0.5)”, and improve the speedup from 1.67x to 2.11x.
- **PAROAttn introduces minimal overhead.** The overhead of sparsification is presented in Sec. 3, since the PAROAttn adopts static sparse scheme, it avoids the online static mask generation overhead. The remaining overhead is the online permutation, and loading of the sparse mask, which are also diminished with the kernel fusion and prefetch techniques, discussed further in the “overhead of permutation/prefetch” paragraph below.
- **PAROAttn supports quantization of PV to lower-bit formats** Comparing the PAROAttn (INT8/4) with SageAttn (V1/V2), PAROAttn could further quantizes the *PV* computation from FP16/FP8 to INT8/INT4 with similar performance, and notable better speedup (1.72x to 1.83x, and 2.56x to 2.97x).

**Overhead of Permutation:** We present an overhead analysis of integrating permutation within the Rotary Position Embedding (RoPE) operator. As shown, this integration introduces only negligible overhead (0.03%), demonstrating that permutation can be fused with prior operators without performance impact.

**Overhead of Prefetch:** As discussed in Section 4.2 of the main paper, static sparse attention introduces additional memory overhead due to the need to store the sparse mask in GPU memory. Since we adopt timestep-wise and transformer-block-wise sparse masks, the memory cost of storing the binary sparse mask is approximately 1GB. To mitigate this cost, we introduce a prefetch scheme that only loads the sparse mask for the current transformer block and timestep, reducing memory usage to the KB level. Additionally, we employ a double-buffering pipeline technique that allows for simultaneous sparse mask loading and attention computation, minimizing the time spent on sparse mask loading. Overall, the prefetching incurs only 0.33% of the total latency.

Method	PSNR $\uparrow$	Speedup $\uparrow$	Overhead $\downarrow$
FlashAttention	-	1.00x	-
SparseAttention (0.5)	16.80	1.67x	6%
SparseVideoGen (0.5)	18.50	1.42x	10%
PAROAttention (0.5)	29.14	1.73x	0%
SparseAttention (0.3)	15.22	2.62x	9%
SparseAttention (0.3 + PARO)	16.89	2.62x	9%
SparseVideoGen (0.3)	17.53	2.11x	15%
PAROAttention (0.3)	22.90	2.71x	0%

Table 2: **Comparison of latency speedup for sparsification methods on NVIDIA A100.**

Method	PSNR $\uparrow$	Speedup $\uparrow$
FlashAttention	-	1.00x
SageAttnV1	29.58	1.72x
PAROAttn (INT8)	29.01	1.83x
SageAttnV2	24.46	2.56x
PAROAttn (INT4)	24.16	2.97x
PAROAttn (0.3 + INT8)	21.49	5.72x
PAROAttn (0.5 + INT4)	24.34	9.28x

Table 3: **Comparison of latency speedup for quantization methods on NVIDIA RTX4090.**

Table 4: **Overhead of permutation.** The latency comparison of whether adopting permutation to rope operator. The “w.” and “w.o.” stands for with and without

	w.o. permute	w. permute	overhead
Latency (ms)	1.2488	1.2492	0.03%

Table 5: **Overhead of prefetching.** The latency comparison of whether adopting prefetch for attention.

	w.o prefetch	w. prefetch	overhead
Latency (ms)	1296.5	1300.8	0.33%

## 4 Implementation Details and Analysis of Baseline Sparsification Methods

**Implementation details:** We select MInference [3], DiTFastAttn [6], SparseVideoGen [5], and SparseAttention [7] for video generation. We visualize the sparse mask generated by baseline sparse methods in Fig. 5.

- **MInference:** We adapt the sparse attention scheme designed for language models to visual attention masks, making the following modifications: First, since we skip sparsification for the larger text tokens, the "attention sink" phenomenon—where the first few tokens are significantly larger than the rest—is not observed. As a result, the “ $\Delta$ -shaped” pattern degrades to a single diagonal pattern, which can be viewed as a special case of the "vertical-slash" pattern. We select between the remaining "vertical-slash" and "block-wise" patterns based on cosine similarity, following the original implementation. Consistent with the original paper, the “vertical-slash” pattern is determined by selecting the top  $K\%$  of vertical and diagonal lines with the largest summed values, where  $K$  can be tuned to adjust the sparsity rate. For the "block-wise" pattern, we use  $8 \times 8$  blocks and determine whether to retain each block based on its summed value.
- **DiTFastAttn:** Consistent with the original paper, we determine the window length by selecting the smallest window where the sum of values within the window reaches  $K\%$  of the total attention values.



- **SparseVideoGen:** We use the official code implementation, the num-sampled-rows are chosen as the default value 32 and 64 for CogVideo and Wan. Specifically, for fair comparison, we donot adopt skipping sparsification for the first timesteps, and set first-times-fp as 0. We also present the ablation of such skipping scheme in Sec. 3.
- **SparseAttn:** We use the official code implementation to test the performance of the CogVideoX model. When adapting SparseAttn to Wan, a numerical stability issue arises, causing the attention computation to produce NaN values; therefore, we omit these results. The hyperparameters for SparseAttn are tuned using the script provided in the official code. For a density of 50%, we set  $l1 = 0.09$  and  $pv_{l1} = 0.095$ . For a density of 30%, we choose  $l1 = 0.13$  and  $pv_{l1} = 0.135$ .

**Visualization of attention masks:** We present a comparison of sparse masks for PAROAttention and baseline sparse attention methods. The first column shows the post-softmax attention patterns. As can be seen, for the SparseVideoGen method, while it successfully identifies the “diagonal in block” temporal attention pattern, the diagonal selection within the block remains inaccurate even at a relatively high dense rate. In contrast, PAROAttn effectively preserves attention values while exploiting sparsity. For DiTFastAttn, the window-based attention struggles with the multiple diagonal pattern and fails to capture diagonals located far from the center. Similarly, MInference’s diagonal pattern is unable to accurately preserve the naturally block-wise attention pattern.

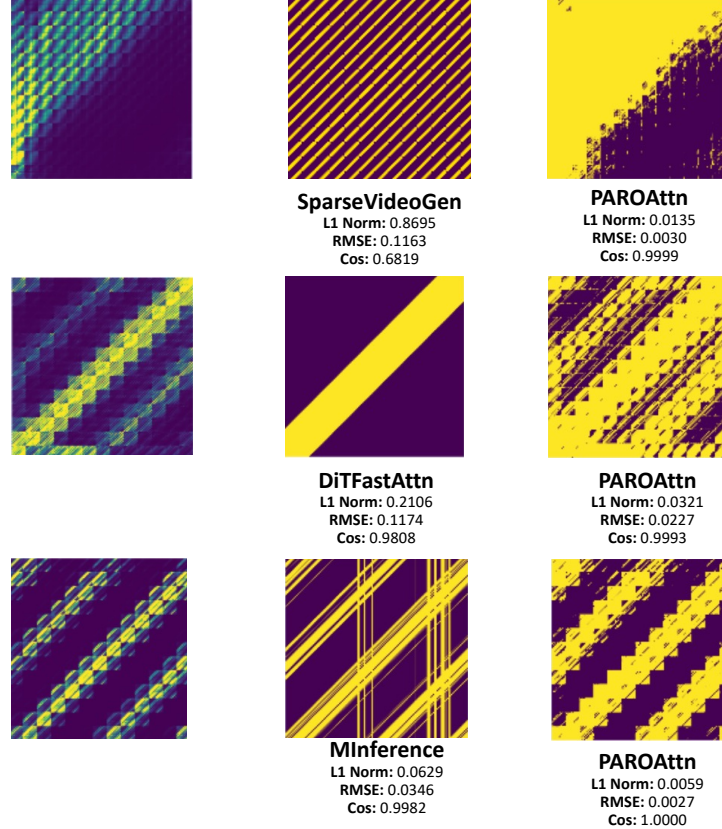


Figure 5: **Comparison of attention masks for PAROAttention and baseline sparse attention methods.** We present the relative difference metrics (L1 Norm, RMSE, CosSim) to measure the difference between the original and masked attention map.

## 5 The Effectiveness of PAROAttention Quantization Technique

**Incoherence Analysis:** To demonstrate the effectiveness of PAROAttention’s quantization technique, we present the data distribution within the quantization group (a  $64 \times 64$  block) in Fig. 6. As shown,

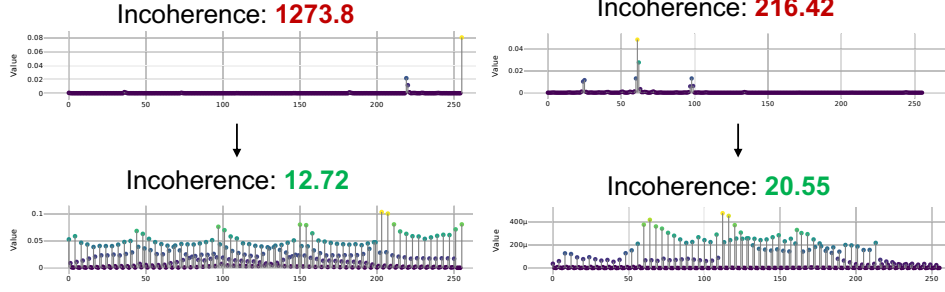


Figure 6: Incoherence for data within the quantization group before and after permutation.

similar values are successfully aggregated into localized blocks, and the outliers present in the original data distribution are significantly reduced. This reduces the incoherence range from 200-1200 to 12-20, and thus significantly reducing the quantization error.

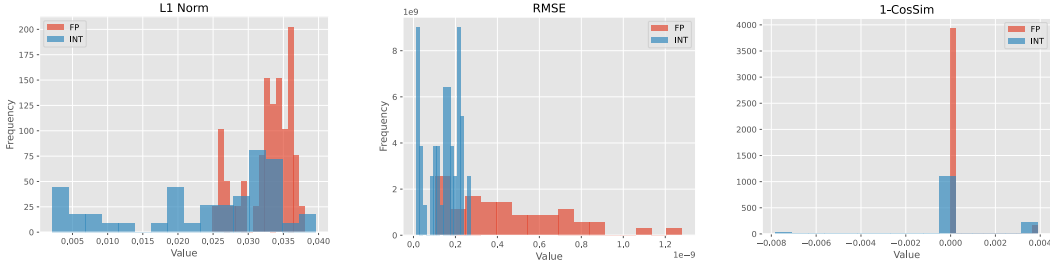


Figure 7: Quantization error with respect to FP for quantization of the attention map  $P$ . The red “FP” stands for the FP8 quantization error.

**Comparison with FP8 quantization:** In SageAttnV2 (Table 6), the authors analyze the cosine similarity between the quantized  $P$  matrix and its original FP counterpart, concluding that INT8 quantization introduces too much error and thus opting for FP8. However, after applying pattern-aware token reordering, the incoherence within the attention map data groups is significantly reduced, leading to a notable decrease in quantization error. As shown in Fig. 7, the INT8 quantized attention map achieves significantly higher FP difference metric scores compared to its FP8 counterpart. This is because INT8 provides more mantissa bits to accurately represent subtle value differences.

**Reasons for exploring interger quantization:** Despite FP8 quantization achieves good performance and valid acceleration with easy deployment. We summarize the reasons for exploring integer quantization as follows:

- **Lower quantization error:** Low-bit floating-point formats consist of both exponent bits and mantissa bits. For example, the E5M2 FP8 format has 5 exponent bits and 2 mantissa bits. The reduced number of mantissa bits limits its ability to represent small value differences, potentially leading to performance degradation. In contrast, with the same bitwidth, integer formats provide more mantissa bits (e.g., equivalently 7 mantissa bits for INT8), enabling them to preserve subtle data differences and achieve lower quantization error. By applying proper preprocessing to remove outliers within data groups, the need for additional exponent bits to handle large dynamic variations is reduced. This advantage becomes even more pronounced at lower bitwidths, such as 4-bit, where FP4 formats have only 1-2 mantissa bits. As presented in Fig. 5 in the main paper, the **all INT4** PAROAttention quantized Flux model could still generate images with high quality. To conclude, integer quantization’s representation power is essential for lower bitwidth quantization.
- **Support non-GPU hardware platforms.** Despite Nvidia TensorCore [1] demonstrate simialr computing power for FP8 and INT8 matrix multiplication. However, for domain-specific accelerator design [2], adopting INT8 matrix multiplication could be more resource-efficient than FP8. Therefore, integer quantization is valuable for AI accelerator hardware design beyond GPU.

## 6 Generalization of Sparse Attention Mask

We present a visualization of the post-softmax attention patterns across different timesteps, prompts, and classifier-free guidance (CFG) settings in Fig. 8. The relative metric scores (L1 Norm, RMSE, cosine similarity) are calculated based on the attention pattern at timestep 5, as indicated by the red text. As shown, the type of attention pattern remains consistent across timesteps, prompts, and CFG. However, the detailed attention pattern may vary over timesteps, gradually stabilizing in later timesteps. To address this, we design timestep-wise sparse masks and share the sparse mask for later timesteps.

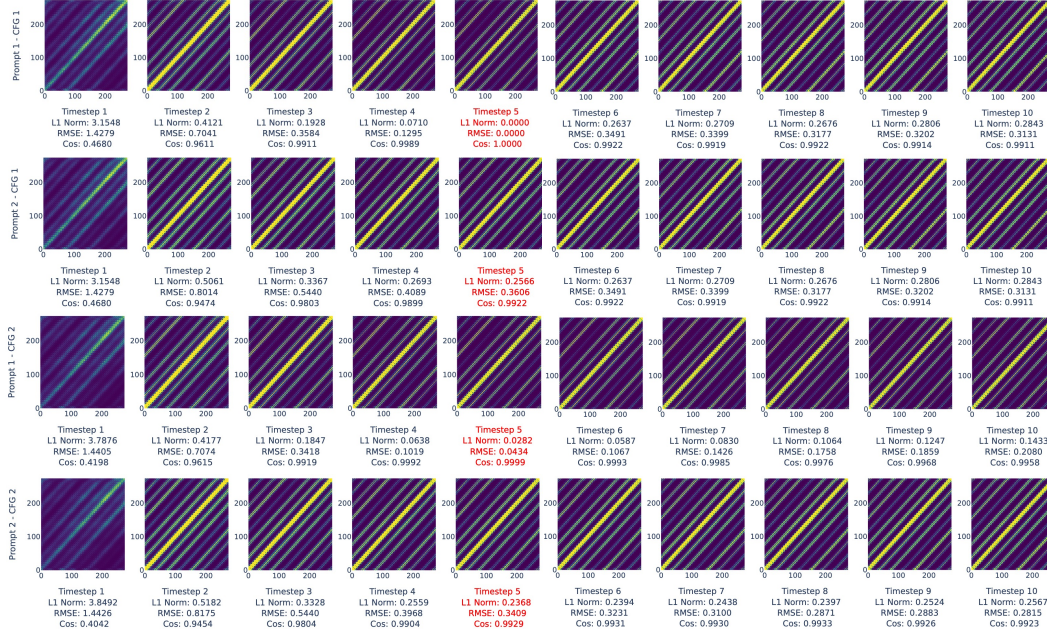


Figure 8: Visualization of post softmax attention pattern for different timesteps, prompts, and classifier-free-guidance (CFG). The metric scores are calculated relative to the attention pattern with red text.

## 7 Additional Visualization of Permutation for Flux

We present the attention pattern for flux under different permutations. The permutation also effectively produces concentrated and regular block-wise pattern.

## 8 Discussion of application of PAROAttn

As discussed in the "Discussion of Adaptability" section of the main paper, we introduce pattern-aware token reordering (permutation) as a universal and efficient preprocessing step for attention patterns. Its effectiveness stems from the unique properties of vision transformers, where 3D (or 2D) spatial information is flattened into a 1D token sequence, disrupting local adjacency. By concentrating attention patterns into more regular and block-wise structures, it benefits a wide range of application scenarios.

**For Compression Techniques:** The advantages of this approach extend beyond the specific design of PAROAttention and are applicable to various compression techniques. For dynamic sparse attention methods, such as SpargeAttn [7], the relative importance of blocks becomes more apparent, simplifying the task of generating sparse masks from QK embeddings. Additionally, the concentrated patterns can improve caching strategies for feature reuse.

**For Model Training:** PAROAttention’s permutation design also sheds light on the meaning of attention patterns, which could inspire future improvements in model training. For instance, it could

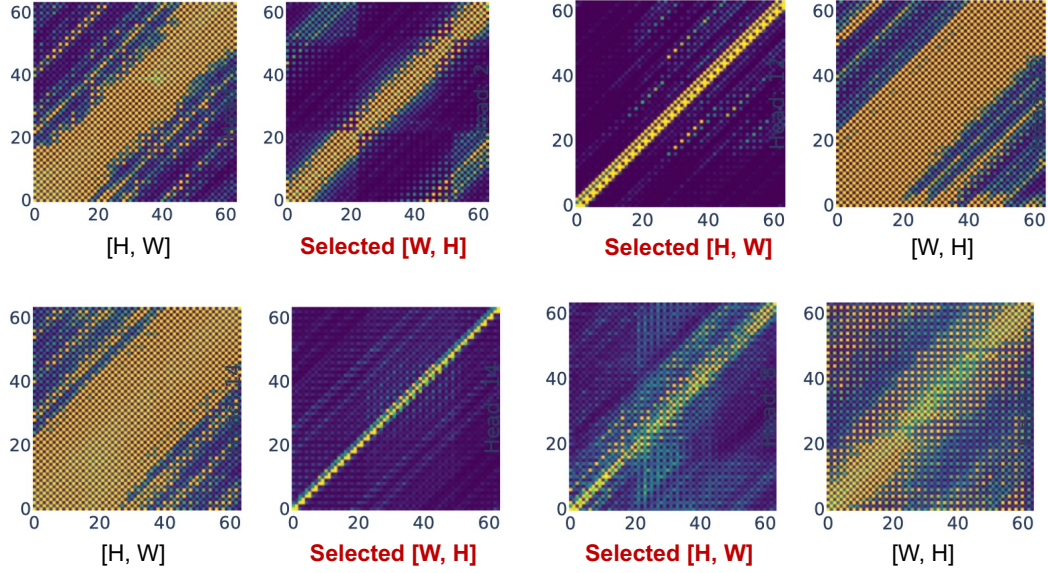


Figure 9: **Visualization of the attention pattern for flux under different permutation.**

encourage different attention heads to focus on aggregating information along different dimensions, leading to more specialized and efficient learning.

**Beyond Visual Generative Models:** The effectiveness of permutation arises from the unique properties of vision transformers, but its applicability is not limited to visual generative models. It could potentially be extended to multi-modal large language models and large vision models for perception tasks, offering similar benefits in these domains.

## 9 Limitations and Broader Impacts

The methodology can be further improved from several perspectives. Permutation represents a constrained subset of possible token reordering, and we adopt simple block sum as sparse metric, exploring more advanced reordering techniques or sparse metric could further enhance performance. PAROAttn introduces a novel direction by leveraging token reordering to reorganize attention patterns. The idea is not limited to post-training compression, and could be extended to broader applications, such as enabling native sparse attention or training acceleration.



## References

- [1] Tensor core. <https://resources.nvidia.com/en-us-tensor-core>,. 6
- [2] Xilinx dsp. <https://docs.amd.com/r/2021.2-English/ug1483-model-composer-sys-gen-user-guide/DSP48E>,. 6
- [3] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention, 2024. 4
- [4] Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwei Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wenten Wang, Wenting Shen, Wenyan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. 1
- [5] Haocheng Xi, Shuo Yang, Yilong Zhao, Chenfeng Xu, Muyang Li, Xiuyu Li, Yujun Lin, Han Cai, Jintao Zhang, Dacheng Li, et al. Sparse videogen: Accelerating video diffusion transformers with spatial-temporal sparsity. *arXiv preprint arXiv:2502.01776*, 2025. 4
- [6] Zhihang Yuan, Hanling Zhang, Lu Pu, Xuefei Ning, Linfeng Zhang, Tianchen Zhao, Shengen Yan, Guohao Dai, and Yu Wang. DiTFastattn: Attention compression for diffusion transformer models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. 4
- [7] Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattn: Accurate sparse attention accelerating any model inference, 2025. 1, 4, 7