SAGEATTENTION2: EFFICIENT ATTENTION WITH SMOOTHING Q AND PER-THREAD QUANTIZATION

Jintao Zhang[1],* Haofeng Huang[2],* Pengle Zhang[1], Jia Wei[1], Jun Zhu[1], Jianfei Chen[1]†
[1]Dept. of Comp. Sci. & Tech., Institute for AI, BNRist Center,
Tsinghua-Bosch Joint ML Center, THBI Lab, Tsinghua University
[2]Institute for Interdisciplinary Information Sciences, Tsinghua University
{zhang-jt24@mails., huanghf22@mails., dcszj@, jianfeic@}tsinghua.edu.cn

ABSTRACT

Although quantization for linear layers has been widely used, its application to accelerate the attention process remains limited. To further enhance the efficiency of attention computation compared to SageAttention while maintaining precision, we propose `SageAttention2`, which utilizes significantly faster 4-bit matrix multiplication (Matmul) alongside additional precision-enhancing techniques. First, we propose to quantize matrices (Q, K) to INT4 in a hardware-friendly thread-level granularity and quantize matrices $(\widetilde{P}, V)$ to FP8. Second, we propose a method to smooth Q, enhancing the accuracy of INT4 $QK^\top$. Third, we propose a two-level accumulation strategy for $\widetilde{P}V$ to enhance the accuracy of FP8 $\widetilde{P}V$. The operations per second (OPS) of `SageAttention2` surpass FlashAttention2 and xformers by about **3x** and **4.5x** on RTX4090, respectively. Moreover, `SageAttention2` matches the speed of FlashAttention3(fp8) on the Hopper GPUs, while delivering much higher accuracy. Comprehensive experiments confirm that our approach incurs negligible end-to-end metrics loss across diverse models—including those for language, image, and video generation.

1 INTRODUCTION

Motivation. For the two matrix multiplication (Matmul) operations in attention: $QK^\top$ and $\widetilde{P}V$, SageAttention accelerates them by quantizing the $QK^\top$ to INT8 and uses FP16 Matmul with FP16 accumulators for $\widetilde{P}V$. Moreover, to keep the accuracy of attention, SageAttention proposes smoothing K by eliminating its channel-wise outliers. SageAttention achieves 2 $\times$ speedup than FlashAttention2 and is the first quantized attention that incurs negligible end-to-end metrics loss across language, image, and video generation models. However, SageAttention has two weaknesses. **(W1)** INT8 Matmul achieves only half the speed of INT4. **(W2)** FP16 Matmul with FP16 accumulators provides a speedup only on RTX 4090 and RTX 3090 GPUs. To leverage the faster INT4 tensor cores for $QK^\top$ and using a method that can accelerate $\widetilde{P}V$ on a broader range of GPUs, we propose to quantize Q, K to INT4 and $\widetilde{P}, V$ to FP8.

Challenges. We identified two main challenges when quantizing Q, K to INT4 and $\widetilde{P}, V$ to FP8: **(C1)** INT4's limited numerical range causes significant quantization errors when Q and K have abnormal values. **(C2)** The FP32 accumulator for FP8 matrix multiplication in tensor cores (`mma.f32.f8.f8.f32`) is actually FP22, with 1 sign bit, 8 exponent bits, and 13 mantissa bits, leading to accuracy loss in $\widetilde{P}V$.

Our approach. To address **(C1)**, we found that per-block quantization of Q, K in SageAttention does not provide sufficient accuracy for INT4 quantization. To avoid the latency from fine-grained per-token dequantization, we propose a per-thread quantization method based on the GPU thread-memory mapping from PTX `mma` instructions. This ensures each thread uses a single quantization scale, achieving finer quantization granularity without extra dequantization overhead. Additionally,

*Equal contribution.
†Corresponding author.

to handle significant channel-wise outliers in Q and K, we adopt smoothing K in SageAttention and further propose subtracting the channel-wise average of Q (denoted $\overrightarrow{Q}_m$), then adding $\overrightarrow{Q}_m K$ after the $QK^\top$ Matmul to preserve attention accuracy. To address **(C2)**, the accuracy loss from using a 22-bit accumulator for FP8 Matmul of $\widetilde{P}V$, we propose a two-level accumulation strategy that uses an FP32 buffer to accumulate the values from the 22-bit accumulator after each block Matmul of $\widetilde{P}V$, confining errors to the block range.

Performance. We provide a high-performance implementation of `SageAttention2` on Ada and Hopper architectures, achieving a peak performance of **481 TOPS** on the RTX4090, outperforming FlashAttention2 and xformers by 3x and 4.5x, respectively. On Hopper GPUs, SageAttention2 matches the speed of FlashAttention3(fp8), while delivering much higher accuracy.

2 PRELIMINARY

FlashAttention. The attention computation can be formulated as: $S = QK^\top/\sqrt{d}$, $P = \sigma(S)$, $O = PV$, where $\sigma(S)_{ij} = \exp(S_{ij})/\sum_k \exp(S_{ik})$. The matrices Q, K, and V each has dimensionality $N \times d$, and S, P are $N \times N$. FlashAttention (Dao, 2024) introduces a GPU-friendly attention implementation, which tiles Q, K, and V from the token dimension into blocks $\{Q_i\}_{i=1}^{n_q}, \{K_i\}_{i=q}^{n_k}, \{V_i\}_{i=1}^{n_v}$ with block sizes of b_q, b_k, b_v ($b_k = b_v$) tokens, respectively, where n_q, n_k, n_v are the number of tiles. FlashAttention computes the output matrix O in parallel in tiles. Each streaming multiprocessor (SM) computes a block O_i (corresponds to a Q_i) by iteratively loads K_j, V_j for each j, and update the output with online softmax (Milakov & Gimelshein, 2018):

$$\begin{aligned} S_{ij} =& Q_i K_j^\top/\sqrt{d}, \quad (m_{ij}, \widetilde{P}_{ij}) = \tilde{\sigma}(m_{i,j-1}, S_{ij}), \\ l_{ij} =& \exp(m_{i,j-1} - m_{ij}) l_{i,j-1} + \mathrm{rowsum}(\widetilde{P}_{ij}), \\ O_{ij} =& \mathrm{diag}\left(\exp(m_{i,j-1} - m_{ij})\right) O_{i,j-1} + \widetilde{P}_{ij} V_j, \end{aligned} \tag{1}$$

where m_{ij} and l_{ij} are b_q-dimenalional vectors, which are initialized with $-\infty$ and 0 respectively. $\tilde{\sigma}(\cdot)$ is an online softmax operator: $m_{ij} = \max\{m_{i,j-1}, \mathrm{rowmax}(S_{ij})\}$, $\widetilde{P}_{ij} = \exp(S_{ij} - m_{ij})$. Finally, the output can be computed as $O_i = \mathrm{diag}(l_{i,n_q})^{-1} O_{i,n_q}$.

3 SAGEATTENTION2

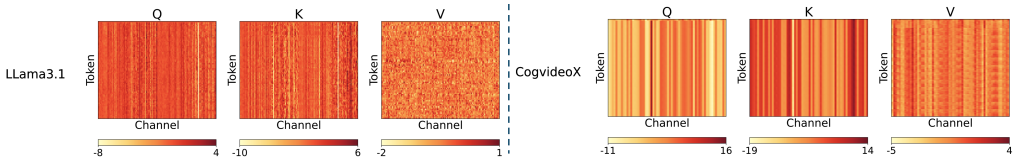


Figure 1: Typical examples of tensors' data distribution in attention.

3.1 SMOOTH Q

First, we discuss how to accurately compute $QK^\top$ with INT4. The numerical range of INT4 is notably restrictive. This affects quantization due to the presence of *outliers* Lin et al. (2024). It is quite likely that most non-outlier elements are quantized to zero, resulting in significant accuracy degradation. Therefore, we propose a smoothing technique inspired by an observation of SageAttention Zhang et al. (2025) that Q, K for all tokens are actually highly similar, with only small variations between different tokens (see Fig. 1). We propose to smooth K as SageAttention does and further smooth Q by subtracting a common mean of each block:

$$\gamma(Q_i) = Q_i - \bar{q}_i, \quad \gamma(K_j) = K_j - \bar{k}, \tag{2}$$

where $\bar{q}_i = \mathrm{mean}(Q_i), \bar{k} = \mathrm{mean}(K)$ are $1 \times D$ vectors, the mean is conducted along the token axis, and $\bar{q}_i, \bar{k}$ are broadcasted to tokens in a block and a tensor for subtraction, respectively.

With the decomposion, we have $S_{ij} = Q_i K_j^\top = (\bar{q}_i + \gamma(Q_i))(\bar{k} + \gamma(K_j))^\top = \gamma(Q_i)\gamma(K_j)^\top + \Delta S_{ij} + b$, where $\Delta S_{ij} = \bar{q}_i \gamma(K_j)^\top$ is an $1 \times N$ vector, and $b = \bar{q}_i \bar{k}^\top + \gamma(Q_i)\bar{k}^\top$ is an $N \times 1$ vector. We do not need to compute b since adding a common bias to an entire row of S does not affect the result after softmax. Therefore, we can accelerate $Q_i K_j^\top$ with INT4 by the following two stages:

(1) *preprocessing*: smooth Q, K according to Eq. (2), apply quantization $(\delta_{Q_i}, \hat{Q}_i) = \psi_Q(\gamma(Q_i)), (\delta_{K_j}, \hat{K}_j) = \psi_K(\gamma(K_j))$, where ψ is the quantization operator, and compute $\Delta S_{ij} = \bar{q}_i \gamma(K_j)^\top$. The smoothing, quantization, and the GEMV (general matrix-vector multiplication) for computing ΔS can be fused into a single kernel, which scans the off-chip Q and K only once.

(2) *attention*: execute the low-precision GEMM, dequantize, and add back the vector ΔS: $S_{ij} = \psi^{-1}_{\delta_{Q_i}\delta_{K_j}}(\hat{Q}_i \hat{K}_j^\top) + \Delta S_{ij}$. These operations are all done on chip, and the vector addition only adds a marginal overhead compared to the expensive `mma` operation for MM (ablated in Table. 13).

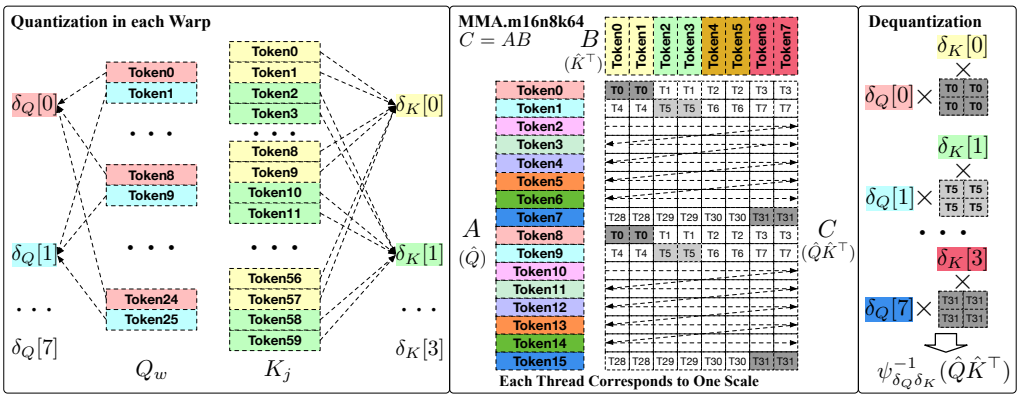


Figure 2: An example of per-thread quantization. The left figure shows the correspondence between the quantization scales and the tokens in each GPU warp. The right figure shows the correspondence between quantization tokens and GPU threads in a `MMA.m16n8k64` instruction, showing that each GPU thread only corresponds to one quantization scale in δ_Q and δ_K in dequantization.

3.2 INT4 Per-thread Quantization

Orthogonal to smoothing, we can mitigate the problem of outliers by refining the quantization granularity so the number of affected elements by outliers becomes smaller. Although per-token offers a high degree of quantization granularity, it results in significant overhead during dequantization. Specifically, each GPU thread in per-token quantization must handle multiple quantization scales, leading to a high latency of the dot product of the quantization scale vectors δ_Q and δ_K. To address this, we propose *per-thread quantization*, a more precise and granular approach than the *per-block quantizer*, also without the additional overhead of vectors dot product between δ_Q and δ_K as in per-token quantization.

Specifically, each Q_i in SageAttention will be split into c_w segments and processed by c_w GPU warps in a GPU streaming processor (SM). We call each segment of Q_i as Q_w, and $k_w = K_j$ since K_j is shared among warps. Then, each warp containing 32 threads uses the `mma.m16n8k64` PTX instruction NVIDIA for the $Q_w K_j^\top$. According to the layout requirement of this instruction, we find that $Q_w[8k+i]$ could share one quantization scale, and $K_j[8k+2i]$ and $K_j[8k+2i+1]$ could share one quantization scale. Such a quantization method is more fine-grained and with no additional overhead. This is because it assigns different GPU threads to distinct quantization groups based on the MMA instruction layout, with each thread performing dequantization only using a single quantization scale value. We show an example of per-thread quantization in Fig. 2. The detailed formulation is shown in Equation 3 and Fig. 14 (Please see Appendix A.3 for more detail).

3.3 FP8 QUANTIZATION FOR $\tilde{P}V$

SageAttention Zhang et al. (2025) choose to retain $\tilde{P}$ and V in FP16, and accelerate the MM by decreasing the accumulator precision. However, this strategy is only effective on very few GPUs. We propose to quantize $\widetilde{P}, V$ to FP8 with 4 exponent bits and 3 mantissa bits (E4M3). The numerical range of E4M3 is $[-448, +448]$. We quantize P with a static scale: $\delta_P = \frac{1}{448}$ since the original P elements is already in $[0, 1]$. We quantize V per-channel to address the channel-wise outliers shown in Fig. 1. Empirical results in Table 11 and Table 12 show the average accuracy and the worst of different data types used for $\widetilde{P}, V$ across all layers of `CogvideoX`. The accumulator is always 32-bit. We can see that the accuracy of E4M3 is very close to that of FP16 and superior to E5M2 and INT8. Most modern GPUs have tensor cores that support FP8 Matmul operations, which are twice as fast as those using FP16.

3.4 FP32 MMA BUFFER FOR FP22 ACCUMULATOR

While FP8 quantization for $\tilde{P}V$ above is accurate in simulation, we observe that the actual CUDA implementation suffers a consistent accuracy degradation. After narrowing down the problem, we find that the accumulator for the `mma(f32f8f8f32)` instruction on the Ada and Hopper architecture is actually FP22, specifically with 1 sign bit, 8 exponent bits, and 13 mantissa bits. Details are presented in Appendix A.4. Consequently, the matrix multiplication of $\widetilde{P}V$, quantized to FP8, incurs an accuracy loss compared to using an FP32 accumulator.

To mitigate this accuracy loss as much as possible, we adopt the two-level accumulation strategy, which uses an FP32 buffer to accumulate the values of $\widetilde{P}_{ij}V_j$ in FP22. Specifically, we rewrite Eq. (1) as $R_{ij} = \widetilde{P}_{ij}V_j, O_{ij} = \mathrm{diag}\left(\exp(m_{i,j-1} - m_{ij})\right) O_{i,j-1} + R_{ij}$. Here, two sets of accumulators R_{ij} and O_{ij} are maintained in the register. R_{ij} is computed with the `mma(f32f8f8f32)` with 22 effective bits, which is sufficient since we only accumulate over a small number of b_k tokens (e.g., $b_k = 64$). Then, R_{ij} is accumulated to O_{ij} in the high FP32 precision.

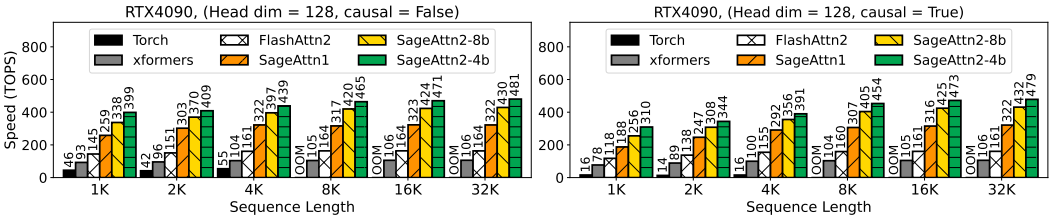


Figure 3: Speed comparison between SageAttention2 and baselines (RTX4090, headdim=128).

4 EXPERIMENT

4.1 SETUP

Models and metrics. For Details about the models and metrics, please refer to Appendix. A.5.

Implemetation. We implement two attention kernels, `SageAttn2-4b` and `SageAttn2-8b` using CUDA. `SageAttn2-4b` uses all techniques introduced in Sec. 3, while `SageAttn2-8b` also uses all techniques but quantizes QK to INT8 without smoothing Q.

Baselines. (1) `SmoothAttn`. Following Qserve (Lin et al., 2024), we apply smooth quant for Q, K with smoothing factor $\alpha = 0.5$. (2) `HadmdAttn`. Following Quarot (Ashkboos et al., 2024), we apply random Hadamard transformation for Q, K before INT4 quantization. (3) `SageAttention` (Zhang et al., 2025), which uses smoothing K, INT8 per-block quantization for Q, K, and FP16 for $\widetilde{P}, V$. (4) `FlashAttn3(fp8)`, the FP8 version of FlashAttention3, which can only run on Hopper GPUs.

Table 1: End-to-end metrics across text, image, and video generation models. ✗ indicates an inability to generate results for evaluation.

Model	Attention	WikiText (Ppl.) ↓	Lambda (Acc.) ↑	MMLU (Acc.) ↑	Longbench ↑
Llama3.1	Full-Precision	6.013	0.815	0.635	49.40
	HadmdAttn	7.872	0.762	0.500	44.07
	SmoothAttn	7.180	0.783	0.541	44.69
	SageAttention	**6.017**	**0.812**	**0.634**	**49.55**
	SageAttn2-4b	**6.256**	**0.798**	**0.607**	**48.79**
	SageAttn2-8b	**6.019**	**0.811**	**0.634**	**49.59**

Model	Attention	CLIPSIM ↑	CLIP-T ↑	VQA-a ↑	VQA-t ↑	FScore ↑
CogvideoX (1.5-5B)	Full-Precision	0.1778	0.9979	70.231	70.928	2.507
	HadmdAttn	0.1576	0.9933	8.990	2.299	✗
	SmoothAttn	0.1559	0.9950	8.812	2.277	✗
	SageAttention	✗	✗	✗	✗	✗
	FlashAttn3-fp8	0.1562	0.9918	6.531	2.181	✗
	SageAttn2-4b	**0.1721**	**0.9978**	**57.729**	**52.989**	**2.884**
	SageAttn2-8b	**0.1775**	**0.9980**	**69.492**	**74.415**	**2.487**
Hunyuan Video	Full-Precision	0.1783	0.9995	82.516	75.934	0.604
	HadmdAttn	0.1727	0.9989	7.514	0.762	0.175
	SmoothAttn	0.1739	0.9988	6.987	0.609	0.148
	SageAttention	**0.1786**	**0.9995**	**82.496**	**79.843**	**0.597**
	FlashAttn3-fp8	0.1742	0.9941	4.433	1.460	✗
	SageAttn2-4b	**0.1751**	**0.9995**	**81.478**	**65.371**	**0.610**
	SageAttn2-8b	**0.1782**	**0.9996**	**81.786**	**75.354**	**0.586**
Mochi	Full-Precision	0.1798	0.9986	45.549	65.416	1.266
	HadmdAttn	0.1733	0.9980	9.053	25.133	0.704
	SmoothAttn	0.1687	0.9978	3.383	3.480	0.241
	SageAttention	**0.1800**	**0.9987**	**48.707**	**63.763**	**1.269**
	FlashAttn3-fp8	0.1762	0.9982	14.964	13.711	0.457
	SageAttn2-4b	**0.1783**	**0.9986**	**35.955**	**43.735**	**1.137**
	SageAttn2-8b	**0.1797**	**0.9986**	**46.760**	**64.901**	**1.255**

Model	Attention	FID ↓	sFID ↓	CLIP ↑	IR ↑
Flux	Full-Precision	10.960	16.648	26.180	1.009
	HadmdAttn	11.353	18.495	26.123	0.965
	SmoothAttn	11.149	19.017	26.109	0.959
	SageAttention	**10.944**	**16.641**	**26.171**	**1.008**
	SageAttn2-4b	**10.577**	**17.497**	**26.141**	**0.998**
	SageAttn2-8b	**10.927**	**16.723**	**26.175**	**1.009**
Stable-Diffusion3.5	Full-Precision	14.105	15.646	25.505	0.902
	HadmdAttn	14.259	15.909	25.513	0.886
	SmoothAttn	14.161	15.649	25.510	0.887
	SageAttention	**14.140**	**15.678**	**25.503**	**0.902**
	SageAttn2-4b	**14.097**	**15.397**	**25.487**	**0.895**
	SageAttn2-8b	**14.106**	**15.647**	**25.499**	**0.901**

Table 2: End-to-end generation latency using SageAttention2 (The latency of Llama3.1 is the time to first token generation using different sequence lengths).

Model	GPU	Original	SageAttn 2-8b	SageAttn 2-4b
CogvideoX (1.5-5B)	RTX4090	1040 s	**577 s**	**555 s**
HunyuanVideo	L20	2221 s	**1486 s**	**1435 s**
Mochi	L20	2336 s	**1316 s**	**1190 s**
Llama3.1 (48K token)	RTX4090	9.2 s	**5.7 s**	**5.6 s**
Llama3.1 (100K token)	L20	39.9 s	**25.4 s**	**23.2 s**

4.2 Speed and Accuracy of Kernels

Kernel Speed. We compare the Speed of SageAttention2 against baselines using configurations with headdim=64 and headdim=128, both with and without Causal Mask (Vaswani, 2017). Specifically, Fig. 3 shows the speed across varying sequence lengths on RTX4090, indicating that SageAttn2-4b and SageAttn2-8b are approximately 3x and 2.7x faster than FlashAtten-

tion2, and about 4.5x and 4x faster than xformers, respectively. Fig. 7, 8, 9, 10, 11, 12, and 13 in Appendix A.2 show more kernel speed comparison on RTX4090, L20, H20, H100 GPUs.

4.3 END-TO-END PERFORMANCE

Metrics loss. We assessed the end-to-end metrics of various models using `SageAttention2`. Detailed evaluation results are presented in Table 1. The results indicate that `SageAttn2-4b` outperforms all baselines and maintains most of the end-to-end accuracy across all models. Additionally, `SageAttn2-8b` incurs almost no metrics loss across various models. More results can be found in Appendix A.6.

Visible image and video examples. Fig.4 and Fig. 6 in Appendix A.1 show some visible comparision examples from `HunyuanVideo`, `Mochi` and `CogvideoX` (1.5-5B). We can observe that `SageAttn2-8b` does not introduce any visible differences compared to full-precision attention, whereas `SageAttn2-4b` has minor differences but is much better than the baselines.

End-to-end speedup. We compared the original generation latency and the latency using `SageAttention2` for models with long sequence lengths in Table 2, observing significant speedup effects. For instance, `SageAttention2` achieved a **1.8x** speedup in `CogvideoX` (1.5-5B) without any metrics loss (`SageAttn2-8b`). `SageAttn2-4b` further accelerated these models but with a little metrics loss.

5 CONCLUSION

We introduce `SageAttention2`, an efficient and accurate quantized attention. First, we propose to quantize matrixes (Q, K) in a thread-level granularity and $(\widetilde{P}, V)$ to FP8. Second, we propose a method to smooth Q, enhancing the accuracy of $QK^\top$. Third, we propose a two-level accumulation strategy to enhance the accuracy of FP8 $\widetilde{P}V$. `SageAttention2` is faster than FlashAttention2 and xformers by approximately **3x** and **4.5x**, respectively. Moreover, `SageAttention2` matches the speed of FlashAttention3(fp8) on the Hopper GPUs, but offers significantly higher accuracy. Extensive testing confirms that our approach maintains end-to-end metrics across language, image, and video generation models.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024.

Black Forest Labs. Flux. `https://github.com/black-forest-labs/flux`, 2023.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng,

Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 8340–8349, 2021.

Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7514–7528, 2021.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1): S63–S63, 1977.

Gregory Kamradt. Llmtest needle in a haystack - pressure testing llms. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`, 2023.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, Kathrina Wu, Qin Lin, Aladdin Wang, Andong Wang, Changlin Li, Duojun Huang, Fang Yang, Hao Tan, Hongmei Wang, Jacob Song, Jiawang Bai, Jianbing Wu, Jinbao Xue, Joey Wang, Junkun Yuan, Kai Wang, Mengyang Liu, Pengyu Li, Shuai Li, Weiyan Wang, Wenqing Yu, Xinchi Deng, Yang Li, Yanxin Long, Yi Chen, Yutao Cui, Yuanbo Peng, Zhentao Yu, Zhiyu He, Zhiyong Xu, Zixiang Zhou, Zunnan Xu, Yangyu Tao, Qinglin Lu, Songtao Liu, Daquan Zhou, Hongfa Wang, Yong Yang, Di Wang, Yuhong Liu, Jie Jiang, and Caesar Zhong. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Daiqing Li, Aleks Kamko, Ehsan Akhgari, Ali Sabet, Linmiao Xu, and Suhail Doshi. Playground v2.5: Three insights towards enhancing aesthetic quality in text-to-image generation. *arXiv preprint arXiv:2402.17245*, 2024.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *arXiv preprint arXiv:2405.04532*, 2024.

Yaofang Liu, Xiaodong Cun, Xuebo Liu, Xintao Wang, Yong Zhang, Haoxin Chen, Yang Liu, Tieyong Zeng, Raymond Chan, and Ying Shan. Evalcrafter: Benchmarking and evaluating large video generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22139–22149, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2022.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

NVIDIA. Parallel thread execution isa version 8.5. `https://docs.nvidia.com/cuda/parallel-thread-execution/`.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, 2016.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

Stability AI. Introducing stable diffusion 3.5. `https://stability.ai/news/introducing-stable-diffusion-3-5`, 2023.

Genmo Team. Mochi 1. `https://github.com/genmoai/models`, 2024.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. *Advances in Neural Information Processing Systems*, 32, 2019.

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

Haoning Wu, Erli Zhang, Liang Liao, Chaofeng Chen, Jingwen Hou, Annan Wang, Wenxiu Sun, Qiong Yan, and Weisi Lin. Exploring video quality assessment on user generated contents from aesthetic and technical perspectives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 20144–20154, 2023.

Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. Imagereward: Learning and evaluating human preferences for text-to-image generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jintao Zhang, Jia Wei, Pengle Zhang, Jianfei Chen, and Jun Zhu. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *The International Conference on Learning Representations*, 2025.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, 2024.

Tianchen Zhao, Tongcheng Fang, Haofeng Huang, Enshu Liu, Rui Wan, Widyadewi Soedarmadji, Shiyao Li, Zinan Lin, Guohao Dai, Shengen Yan, Huazhong Yang, et al. Vidit-q: Efficient and accurate quantization of diffusion transformers for image and video generation. In *International Conference on Learning Representations*, 2025.

Zangwei Zheng, Xiangyu Peng, Tianji Yang, Chenhui Shen, Shenggui Li, Hongxin Liu, Yukun Zhou, Tianyi Li, and Yang You. Open-sora: Democratizing efficient video production for all. *arXiv preprint arXiv:2412.20404*, 2024.

A APPENDIX

A.1 VISIBLE COMPARISON EXMAPLES

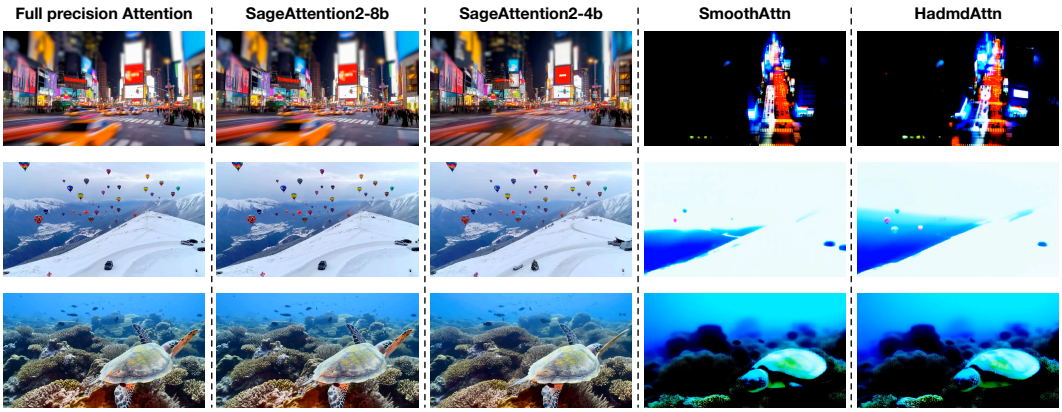


Figure 4: Comparison examples from `HunyuanVideo`, prompts are sampled from open-sora prompt sets.

Figure 5: Comparison examples from `CogvideoX` (2B), prompts are sampled from open-sora prompt sets.

A.2 ADDITIONAL KERNEL SPEED COMPARISON

Fig. 7, 8, 9, 10, 11, 12, and 13 compare the speed of `SageAttention2` against baselines using configurations with headdim=64 and headdim=128, both with and without Causal Mask Vaswani (2017), on RTX4090, L20, H100, and H20 GPUs.

Table 3 summarizes the performance gain of different attention methods against baselines on various modern GPUs.

Table 3: Speedup of different attention methods on various GPUs.

Method	**3090**	**4090**	**A100**	**L40**	**L20**	**H100**	**H20**
FlashAttention2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FlashAttention3	✗	✗	✗	✗	✗	1.37	1.57
FlashAttention3 (fp8)	✗	✗	✗	✗	✗	2.63	3.06
SageAttention1	1.97	1.96	1.37	1.45	1.24	1.53	1.52
SageAttention2	✗	2.93	✗	2.60	2.46	2.61	3.12

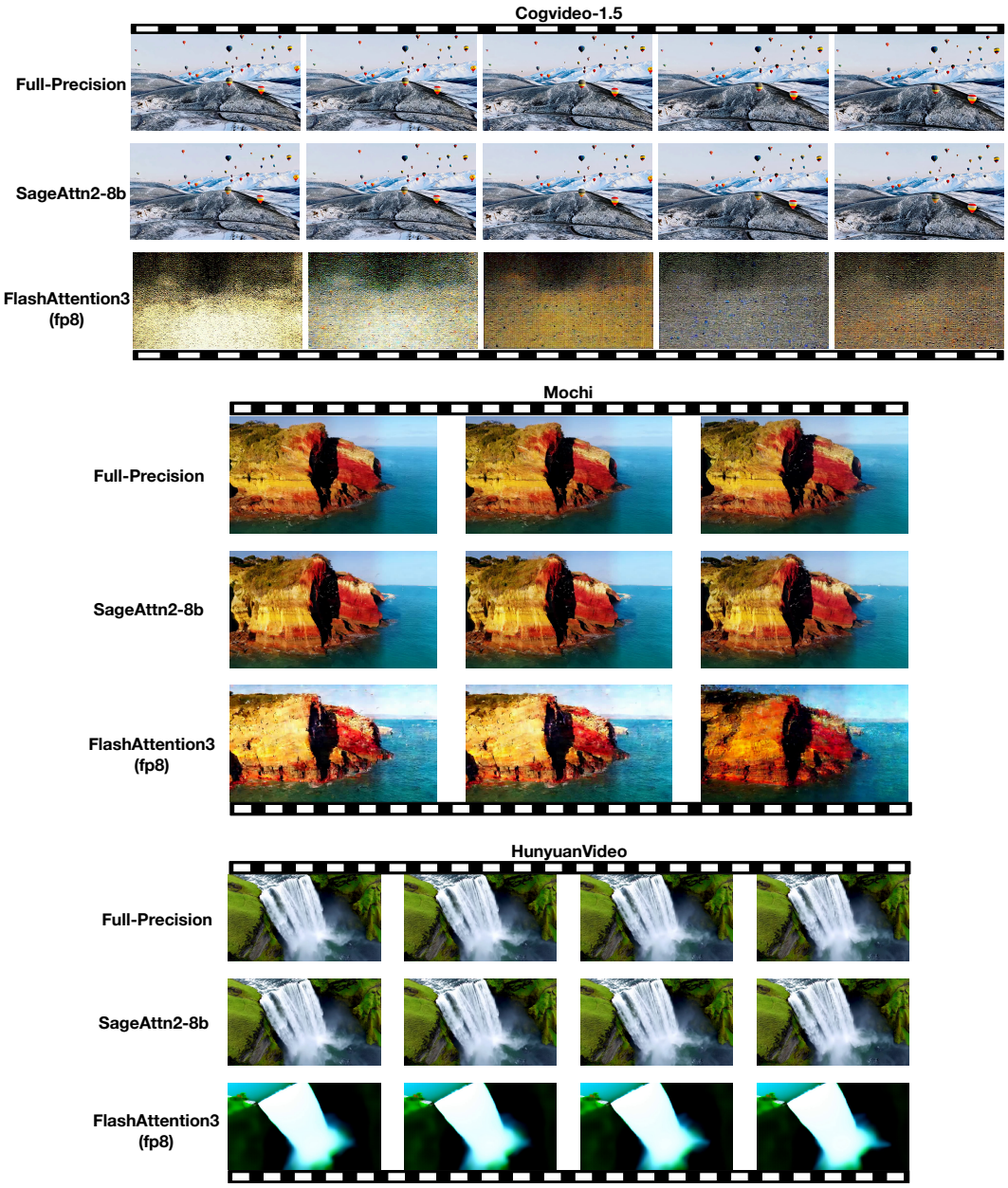


Figure 6: A comparison example using SageAttn2-8b and FlashAttention3 on `CogvideoX`-1.5, `Mochi` and `HunyuanVideo`.

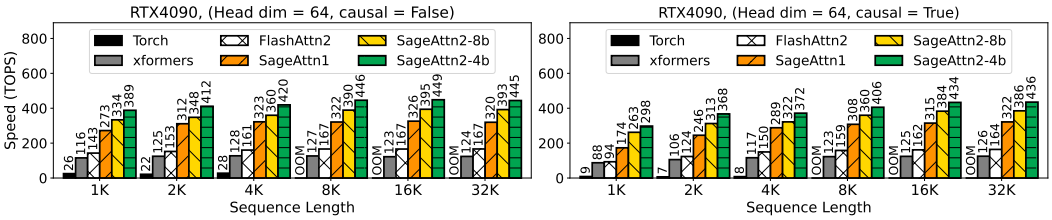


Figure 7: Speed comparison between `SageAttention2` and baselines (RTX4090, headdim=64).

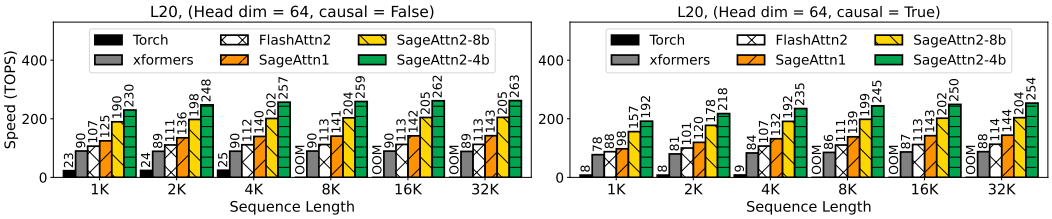


Figure 8: Speed comparison between `SageAttention2` and baselines (L20, headdim=64).

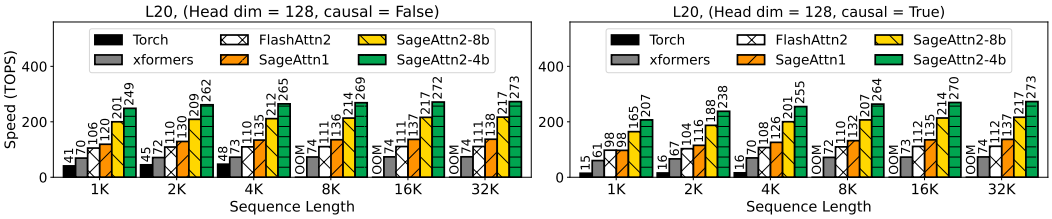


Figure 9: Speed comparison between `SageAttention2` and baselines (L20, headdim=128).

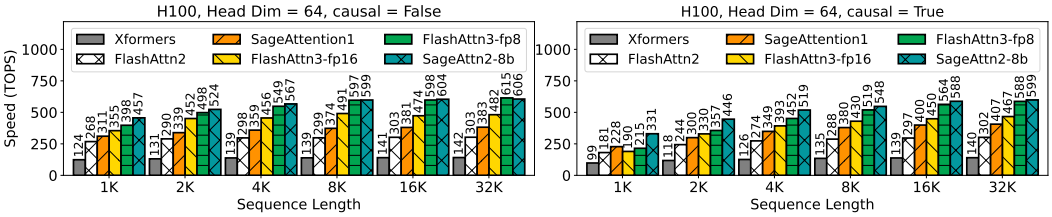


Figure 10: Speed comparison between `SageAttention2` and baselines (H100, headdim=64).

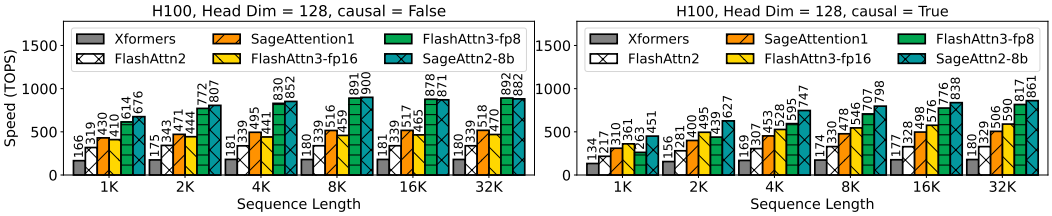


Figure 11: Speed comparison between `SageAttention2` and baselines (H100, headdim=128).

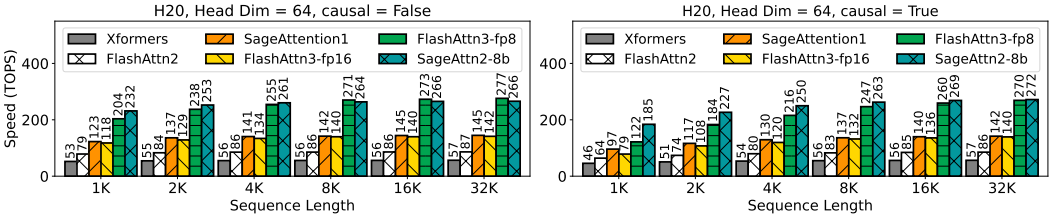


Figure 12: Speed comparison between `SageAttention2` and baselines (H20, headdim=64).

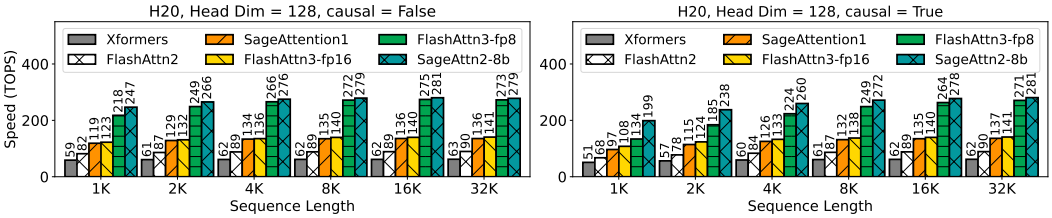


Figure 13: Speed comparison between `SageAttention2` and baselines (H20, headdim=128).

R\C	0 1	2 3	4 5	6 7
0	T0: {c0, c1}	T1: {c0, c1}	T2: {c0, c1}	T3: {c0, c1}
1	T4: {c0, c1}	T5: {c0, c1}	T6: {c0, c1}	T7: {c0, c1}
2				
7	T28: {c0, c1}	T29: {c0, c1}	T30: {c0, c1}	T31: {c0, c1}
8	T0: {c2, c3}	T1: {c2, c3}	T2: {c2, c3}	T3: {c2, c3}
9	T4: {c2, c3}	T5: {c2, c3}	T6: {c2, c3}	T7: {c2, c3}
10				
..				
15	T28: {c2, c3}	T29: {c2, c3}	T30: {c2, c3}	T31: {c2, c3}

Figure 14: Memory layout of INT4/INT8 tensor core for accumulator matrix C and D in $D = A * B + C$ among 32 threads (T0 $\sim$ T31) in a warp. C and D is of shape 16x8. Each thread only holds 4 out of the 128 elements.

A.3 Per-Thread Quantization Formulation

To further clarify the per-thread quantization, we first introduce the INT4 MMA instruction of Tensor Core, and then give the formulation of per-thread quantization.

Tensor cores, first introduced in NVIDIA's Volta architecture, are specialized units designed for efficient matrix-multiply-and-accumulate (MMA) operations. Their usage significantly enhances computational efficiency and performance in AI and high-performance computing (HPC) workloads. Tensor cores compute small tiles of MMA operations, specifically $D = A * B + C$ on a warp (32 contiguous threads) basis. Each thread in the warp holds a fragment of input matrices and will get a fragment of output matrix as a computation result. The INT4 `mma.m16n8k64` tensor core operation computes the product of a 16×64 INT4 matrix A and a 64×8 INT4 matrix B, both stored in registers. It accumulates the result into a 16×8 INT32 matrix C, also stored in registers, and returns the final product matrix D, which has the same shape (16×8), data type (INT32), and storage location (registers). Each thread holds only $\frac{1}{32}$ of the input and output data. Fig. 14 extracted from the PTX document NVIDIA shows the memory layout of matrix C and D among 32 threads in a warp. Each thread only holds 4 out of the 128 result elements.

$$
\begin{aligned}
i_{\delta q} =& \lfloor (n * 8 * c_w / b_q) \rfloor \\
q_i[i_{\delta q}] =& \{8 \times (n\%8) + \lfloor (n * \frac{c_w}{b_q}) \rfloor * \frac{b_q}{c_w}\},\ n \in [0, N] \\
\delta_Q[i_{\delta q}] =& \frac{\max(|\, Q[q_i[i_{\delta q}]] \,|)}{7} \\
\hat{Q}[q_i[i_{\delta q}]] =& \left\lceil \frac{Q[q_i[i_{\delta q}]]}{\delta_Q[i_{\delta q}]} \right\rfloor \\
i_{\delta k} =& \lfloor (n * 4 / b_k) \rfloor \\
k_n[i_{\delta k}] =& \{8 \times (n\%8) + \lfloor n / b_k \rfloor * b_k\} \cup \\
& \{8 \times (n\%8) + 1 + \lfloor n / b_k \rfloor * b_k\},\ \ n \in [0, N] \\
\delta_K[i_{\delta k}] =& \frac{\max(|\, K[k_n[i_{\delta k}]] \,|)}{7} \\
\hat{K}[k_n[i_{\delta k}]] =& \left\lceil \frac{K[k_n[i_{\delta k}]]}{\delta_K[i_{\delta k}]} \right\rfloor
\end{aligned}
\tag{3}
$$

In Eq. 3, c_w is the count of GPU Warps. b_q and b_k are the block size of Q, K. n is the token index of Q, K.

By ensuring results held by each thread share a common dequantization scale (belong to the same quantization group), we can avoid the overhead associated with per-token quantization. Leveraging this observation, we design per-thread quantization, as shown in Fig. 2. For typical block size of $b_q = 128$, $b_k = 64$ and warp number $c_w = 4$ (as used in FlashAttention2), each warp processes a tile of 32 query tokens and 64 key tokens. Query tokens $i, 8+i, 16+i, 24+i$ $(i = 0, 1, \cdots, 7)$ can be made into one quantization group and key tokens $j, 1+j, 8+j, 9+j, \cdots, 56+j, 57+j$ $(j = 0, 1, 2, 3)$ can be made into one quantization group. This design aligns with the memory layout of output matrix D of tensor core shown in Fig. 14, ensuring that each thread only needs one Q scale and one K scale for dequantization.

As a result, this approach creates 32 quantization groups for Q (8 for each of the 4 warps) and 4 quantization groups for K in a 128x64 block, providing $32\times$ and $4\times$ finer granularity compared to per-block quantization for query tokens and key tokens, respectively. Table 9 and Table 10 show the accuracy gains by using per-thread quantization. Per-thread quantization achieves accuracy that closely matches per-token quantization, without introducing any kernel speed degradation (see Fig. 13).

A.4 FP22 Accumulator of FP8 Tensor Core

Table 4: Error of the FP8 (E4M3) Matmul instruction of `mma(f8f8f32)`.

Precision of Accumulated Value	E8M13	E8M23
Error compared to FP32	0	|`mma(f16f16f32)` - `mma(f8f8f32)`|

We use the following experiment to test the number of effective bits of accumulator of `mma(f8f8f32)` instruction, which performs $C = AB + D$, where A, B are tensors in FP8 (E4M3) data type and C, D are tensors with FP32 data type. We initialize the A, B to zero and vary D to test the data type of the accumulator. As shown in Table 4, when D is initialized with 1 sign bit, 8 exponent bits, and no more than 13 mantissa bits, the value of C precisely matches the result of the `mma(f16f16f32)` instruction. However, when D is initialized with more than 13 mantissa bits, the error of C corresponds to the difference between the results of `mma(f16f16f32)` and `mma(f8f8f32)`. This proves that the number of effective bits of accumulator for `mma(f8f8f32)` is 22. On Hopper architecture, the instruction is `wgmma(f8f8f32)` but the number of effective bits is the same.

A.5 Models, Datasets, and Metrics in Experiments

Models. We validate the effectiveness of `SageAttention2` across a diverse set of representative models from language, image, and video generation. Specifically, we conduct experiments on nine models: `Llama3.1` (8B) (Dubey et al., 2024) and `GLM4` (9B) (GLM et al., 2024) for text2text, `CogvideoX` (2B), `CogvideoX` (1.5-5B) (Yang et al., 2025), `HunyuanVideo` (Kong et al., 2024), and `Mochi` (Team, 2024) for text2video, `Flux` (schnell) (Black Forest Labs, 2023) and `Stable-Diffusion3.5` (turbo) (Stability AI, 2023) for text2image, and `TIMM` (Wightman, 2019) for image classification.

Datasets. Text-to-text models are evaluated on four zero-shot tasks: WikiText Merity et al. (2022) to assess the model's prediction confidence, LAMBADA Paperno et al. (2016) evaluate contextual understanding, MMLU Hendrycks et al. (2020) for measuring knowledge across various subjects, and Longbench Bai et al. (2024) for comprehensive assessment of long context understanding capabilities. Text-to-video models are evaluated using the open-sora Zheng et al. (2024) prompt sets. Text-to-image models are assessed on MJHQ-30K Li et al. (2024). `TIMM` is evaluated on on three image datasets: ImageNet Deng et al. (2009), ImageNet-Sketch (Sketch) Wang et al. (2019), and ImageNet-Rendition (ImageNet-r) Hendrycks et al. (2021).

End-to-end metrics. For text-to-text models, we use perplexity (ppl.) Jelinek et al. (1977) for WikiText, Accuracy (Acc.) for LAMBADA and MMLU, and Longbench score Bai et al. (2024). For text-to-video models, following Zhao et al. (2025), we evaluate the quality of generated videos on five metrics: CLIPSIM and CLIP-Temp (CLIP-T) Liu et al. (2024) to measure the text-video alignment; (VQA-a) and (VQA-t) to assess the video aesthetic and technical quality, respectively;

and Flow-score (FScore) for temporal consistency Wu et al. (2023). For text-to-image models, generated images are compared with the images in MJHQ-30K dataset in three aspects: FID Heusel et al. (2017) and sFID Salimans et al. (2016) for fidelity evaluation, *Clipscore* (CLIP) Hessel et al. (2021) for text-image alignment, and *ImageReward* (IR) Xu et al. (2023) for human preference. For `TIMM`, we use classification accuracy.

Accuracy metrics. We use three metrics to assess the accuracy of quantized attention output O' compared to attention output in full-precision O: First, we flatten O' and O into vectors in the shape of $1 \times n$. Then, Cosine similarity: $CosSim = \sum OO' / \sqrt{\sum O^2}\sqrt{\sum O'^2}$, Relative L1 distance: $L1 = \sum |O - O'| / \sum |O|$, Root mean square error: $RMSE = \sqrt{(1/n)\sum (O - O')^2}$.

Table 5: End-to-end metrics on `GLM4` (9B).

Model	Attention	WikiText (Ppl.) ↓	Lambda (Acc.) ↑	MMLU (Acc.) ↑	Longbench ↑
`GLM4`	Full-Precision	7.241	0.432	0.743	49.78
	`HadmdAttn`	7.989	0.435	0.669	45.97
	`SmoothAttn`	8.943	0.449	0.592	42.20
	`SageAttention`	**7.243**	**0.433**	**0.744**	**49.79**
	`SageAttn2-4b`	**7.352**	**0.433**	**0.725**	**49.23**
	`SageAttn2-8b`	**7.242**	**0.432**	**0.745**	**49.60**

Table 6: End-to-end metrics on `CogvideoX` (2B).

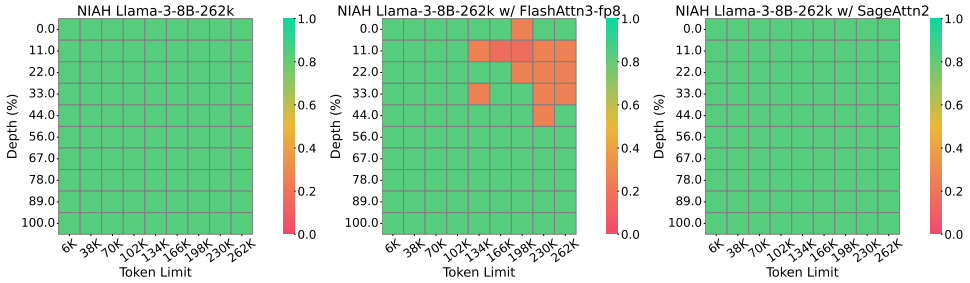
Model	Attention	CLIPSIM ↑	CLIP-T ↑	VQA-a ↑	VQA-t ↑	FScore ↑
`CogvideoX` (2B)	Full-Precision	0.1836	0.9975	77.605	75.360	3.006
	`HadmdAttn`	0.1742	0.9877	29.780	23.985	0.499
	`SmoothAttn`	0.1741	0.9870	41.703	47.043	0.624
	`SageAttention`	**0.1833**	**0.9976**	**76.997**	**71.360**	**2.988**
	`SageAttn2-4b`	**0.1821**	**0.9973**	**77.368**	**74.906**	**2.603**
	`SageAttn2-8b`	**0.1829**	**0.9977**	**76.532**	**74.281**	**2.941**

Table 7: End-to-end metrics on an image classification model.

Model	Attention	ImageNet (Acc.) ↑	Sketch (Acc.) ↑	ImageNet-r (Acc.) ↑
`TIMM`	Full-Precision	84.79%	45.32%	59.55%
	`HadmdAttn`	84.50%	44.89%	58.80%
	`SmoothAttn`	84.40%	44.68%	58.73%
	`SageAttention`	**84.74%**	**45.38%**	**59.95%**
	`SageAttn2-4b`	**86.67%**	**45.24%**	**59.29%**
	`SageAttn2-8b`	**84.79%**	**45.39%**	**59.57%**

Table 8: Comparison with FlashAttention3(fp8) on `Llama-3-262k` (8B) on InfiniBench Zhang et al. (2024) (H100 GPU).

Attention	Eng.Sum	Eng.QA	Eng.MC	Code.Debug	Math.Find	Retr.PassKey	Retr.Num	Retr.KV	Avg.
Full-Precision	18.03	12.5	64.19	24.37	18.29	100.0	100.0	7.0	43.05
`FlashAttn3-fp8`	**19.03**	11.73	55.90	22.59	**22.57**	**100.0**	**100.0**	0.4	41.53
`SageAttention2`	18.17	**12.46**	**64.19**	**25.63**	17.43	**100.0**	**100.0**	**6.6**	**43.06**



(a) Full Precision (b) `FlashAttn3-fp8` (c) `SageAttention2`

Figure 15: Needle In A Haystack results on `Llama-3-262k` (8B).

A.6 Additional Experiments and Analysis

Additional Results. Table 5, 6 and 7 show results of `SageAttention2` and other baselines on `GLM4` (9B), `CogvideoX` (2B) and `TIMM`.

Results of Super-Long Context. We further conduct experiments on super-long context using `Llama-3-262k` (8B)[1] on InfiniBench Zhang et al. (2024) and Needle-in-a-Haystack (NIAH) Kamradt (2023), with sequence lengths reaching up to 262k tokens on an H100 GPU. Results are shown in Table 8 and Fig 15. `SageAttention2` maintains model performance even under super-long context, while FlashAttention3(fp8) suffers from end-to-end accuracy degradation.

Table 9: **Average accuracy** across all layers of `CogvideoX` using different quantization granularities.

Method	Cos Sim ↑	Relative L1 ↓	RMSE ↓
Per-token	99.45%	0.0649	0.0335
Per-thread	99.45%	0.0622	0.0313
Per-block	98.03%	0.1492	0.0744
Per-tensor	97.15%	0.1800	0.0865

Table 10: **Worst accuracy** across all layers of `CogvideoX` using different quantization granularities.

Method	Cos Sim ↑	Relative L1 ↓	RMSE ↓
Per-token	96.76%	0.1916	0.0775
Per-thread	96.72%	0.1932	0.0776
Per-block	90.68%	0.3615	0.1490
Per-tensor	85.85%	0.4687	0.2261

Table 11: **Average accuracy** using different data types of $(\widetilde{P}, V)$ across all layers of `CogvideoX`, where (Q, K) are smoothed.

Q, K	$\widetilde{P}, V$	Cos Sim ↑	Relative L1 ↓	RMSE ↓
INT4	INT8	77.05%	0.5618	0.5044
	E5M2	99.20%	0.0905	0.0903
	E4M3	**99.44%**	**0.0683**	**0.0347**
	FP16	99.45%	0.0649	0.0335

[1] `https://huggingface.co/gradientai/Llama-3-8B-Instruct-262k`

Table 12: **Worst accuracy** using different data types of $(\widetilde{P}, V)$ across all layers of a `CogvideoX` model, where (Q, K) are smoothed.

Q, K	$\widetilde{P}, V$	**Cos Sim** ↑	**Relative L1** ↓	**RMSE** ↓
INT4	INT8	19.52%	0.9579	1.4483
	E5M2	94.94%	0.2327	0.2361
	E4M3	**96.70%**	**0.1956**	**0.0779**
	FP16	96.76%	0.1916	0.0775

Table 13: Overhead of per-thread quantization, smoothing Q, and two-level accumulation techniques measured on L20 GPU.

Method	**TOPS**
Attention (INT4 + FP8)	284
+ Per-thread quantization	283
+ Two-level accumulation	283
+ Smoothing Q	273