Hardware-Efficient Attention for Fast Decoding

Ted Zadouri [1 2] **Hubert Strauss** [1] **Tri Dao** [1 2]

{tz6037,hs6702,td8762}@princeton.edu

Abstract

LLM decoding is bottlenecked for large batches and long contexts by loading the KV cache from high-bandwidth memory, which raises per-token latency, while its sequential nature limits parallelism. We redesign attention to perform more computation per byte of memory transfer, maximizing hardware efficiency without sacrificing parallel scalability. We first present *Grouped-Tied Attention* (GTA), which merges and reuses key and value states to reduce memory traffic without affecting quality. Next, we introduce *Grouped Latent Attention* (GLA), a parallel friendly latent attention enhanced with low-level optimizations for fast decoding at high quality. Experiments show that GTA matches Grouped Query Attention (GQA) quality while using roughly half the KV cache, and GLA matches Multi-head Latent Attention (MLA) yet shards more easily. Our optimized GLA kernel is up to $2\times$ faster than FlashMLA in speculative decoding once the query length exceeds one.

1 Introduction

In light of test-time compute (OpenAI, 2024), inference efficiency now drives progress in AI, demanding a greater emphasis on inference-aware architectures. The sequential nature of token-by-token decoding limits opportunities for parallelization. During decoding, Multi-Head Attention (MHA) (Vaswani et al., 2017) caches the key-value (KV) states of all prior tokens. These cached states scale linearly with batch size and sequence length and quickly exhaust high-bandwidth memory (HBM). Moreover, fetching this large KV cache from off-chip memory dominates execution time, significantly outweighing the relatively small computation performed by the matrix-vector workload at each decoding step. Collectively, these issues highlight the critical need for a hardware-efficient redesign of attention. An ideal attention mechanism should (1) achieve high model quality, (2) scale efficiently across multiple devices, and (3) utilize modern hardware efficiently during inference.

Arithmetic intensity (Williams et al., 2009), the ratio of FLOPs to bytes moved, indicates whether a workload is memory-bound or compute-bound. Multi-Query Attention (MQA) (Shazeer, 2019) increases this ratio by sharing one KV head across all queries, shrinking the KV cache and reducing the reload time from high-bandwidth memory while keeping FLOPs unchanged, but at the cost of quality and parallelism (Pope et al., 2022). Grouped Query Attention (GQA) (Ainslie et al., 2023) modestly increases arithmetic intensity in proportion to group size and scales well in inference; yet with a moderate tensor parallel degree, each GPU still holds a large KV cache and quality drops for very large groups. Multi-head Latent Attention (MLA) from DeepSeek (DeepSeek-AI, 2024; 2025) absorbs its low-rank projection at decode time and uses a single latent head, doubling the arithmetic intensity over MQA, but the latent is duplicated on every device, hampering parallel inference.

We redesign hardware efficient attention through the lens of arithmetic intensity, targeting scalable decoding without quality loss. We denote the group size g_q as the number of query heads per distinct KV head; g_q mainly governs this metric. Increasing g_q increases the arithmetic intensity and proportionally reduces the KV cache. Beyond a threshold, further increases trade higher compute per byte for parallelism, duplicating projection weights and KV cache across devices and limiting scalability. Guided by this analysis, we introduce two attention variants that pair high arithmetic intensity with efficient device scaling, complemented by low-level optimizations.

- Grouped-Tied Attention (GTA) ties key and value into one shared state reused by small query-head groups, halving the KV cache and doubling arithmetic intensity relative to GQA.
- Grouped Latent Attention (GLA) extends latent attention with low-level optimizations, matches MLA quality, and runs up to $2\times$ faster, for example, in speculative decoding when the query length is at least two.

[1]Princeton Language and Intelligence, Princeton University [2]Together AI. Correspondence to: Ted Zadouri <tz6037@princeton.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

- We evaluated in FineWeb-Edu: for the XL 1.47B model, GTA yields 10.12 perplexity versus 10.20 for GQA, while GLA gives 60.0% downstream accuracy with 10.21 perplexity compared with 59.1% and 10.25 for MLA. On the large 876M model, GTA reaches 11.2 perplexity and 57. 6% accuracy compared to GQA's 11.3 and 56.9%. For the medium 433M model, the GLA attains 55.4% accuracy, slightly above MLA's 54.9%.

- We combine their algorithmic design with a collection of low-level optimizations, leading to attention-decoding kernels 1.2-2× faster than FlashMLA [1] (Li, 2025).

2 Methodology

We describe our perspective on designing hardware-efficient attention variants by focusing on the arithmetic intensity. We then demonstrate how to maximize the arithmetic intensity while efficiently parallelizing across devices by tying the key and value states and sharding the cached latent representation.

2.1 Arithmetic Intensity in Decoding: A Hardware-Efficient Perspective

The GPU utilization of standard MHA, where the arithmetic intensity is ~ 1 as shown in Table 1, can drop to as low 7% during decoding (Recasens et al., 2025). In theory, it could accommodate around two and a half orders of magnitude more FLOPs without increasing latency. However, practical speed-ups are naturally smaller due to kernel overheads, limited overlap, and other sources of inefficiency. This low utilization reflects the growing gap between the compute throughput of recent GPU generations and memory bandwidth. Hardware FLOPs have scaled by $\approx 3\times$ every two years, while the HBM memory bandwidth increases by $\sim 1.6\times$ over the same period (Gholami et al., 2024).

Attention Variant	**GLA**	**MLA**	**MQA**	**GQA**	**GTA**	**MHA**	**General Formulation**
Arithmetic Intensity	$\frac{L}{1+\frac{L}{2\cdot g_q}}$	$\frac{L}{1+\frac{L}{2h_q}}$	$\frac{Lh_q}{h_q+L}$	$\frac{Lh_q}{h_q+\frac{h_q}{g_q}L}$	$\frac{2Lh_q}{2h_q+\frac{h_q}{g_q}L}$	$\frac{L}{1+L}$	$\frac{2\cdot L}{2+\frac{m_{kv}}{g_q}L}$
	$\approx 2g_q$	$\approx 2h_q$	$\approx h_q$	$\approx g_q$	$\approx 2g_q$	≈ 1	$\approx \frac{2g_q}{m_{kv}}$

Table 1. Let L be the KV sequence length, h_q the number of query heads, h_{kv} the number of KV heads, and define the group size $g_q = \frac{h_q}{h_{kv}}$ (queries per KV head). The KV multiplicity is $m_{kv} \in \{1,2\}$, with $m_{kv} = 1$ for shared KV states $(K = V)$ and $m_{kv} = 2$ for distinct KV states $(K \neq V)$. We assume $L \gg h_q$.

2.2 Design Strategies for Maximizing Arithmetic Intensity

GQA reuses one KV head for each group of query heads. Because the number of operations remains unchanged while memory reads drop, the arithmetic intensity grows in proportion to the group size g_q (see Table 1). In MQA, which shares a single KV head across all queries, the intensity is roughly the number of query heads, h_q.

DeepSeek's MLA absorbs low-rank projection matrices to avoid materializing KV states, achieving high arithmetic intensity by (i) caching one latent head, (ii) reusing it for both key and value, and (iii) supporting many query heads. A single latent loaded into the chip memory serves K and V, doubling the arithmetic intensity over aggressive MQA designs. The parameter savings let the up projection widen to add more query heads, maintaining capacity and raising intensity. As Table 1 shows, the arithmetic intensity grows with the number of query heads: a larger g_q both shrinks the KV cache and maximizes GPU utilization. Let $m_{kv} \in \{1,2\}$ denote KV multiplicity; increasing m_{kv} from 1 to 2 enlarges the cache and lowers intensity.

$$\mathrm{KV}_{\mathrm{Bytes}} = m_{kv} \cdot B \cdot L \cdot \frac{h_q}{g_q} \cdot d_h \times \mathrm{sizeof(dtype)}$$
$$\text{Arithmetic Intensity} \approx \frac{2 \cdot L \cdot h_q}{2 \cdot h_q + \frac{m_{kv} \cdot h_q}{g_q} L} \approx \frac{2 \cdot g_q}{m_{kv}}$$

For *zero redundancy parallelism*, the number of KV heads $h_{kv} = h_q/g_q$ should be at least N and at most h_q, that is, $g_q \leq h_q/N$. Here, B is the batch size and d_h the dimension per head. The increase g_q increases the intensity of arithmetic but also the duplication factor D; when D equals the tensor parallel (TP) shard count N, each device retains the full parameters and KV cache, eliminating parallel gains. Thus, zero redundancy parallelism requires $h_{kv} = h_q/g_q \geq N$, that is, $g_q \leq h_q/N$.

2.3 Hardware-Efficient & Parallelizable Attention

2.3.1 Grouped-Tied Attention (GTA)

Singular value plots show a steep decay in the key cache: a few principal directions capture almost all variance, so the keys lie in a low-rank subspace (Saxena et al., 2024). Before RoPE, this collapse was even stronger (Sun et al., 2024). Studies find that rotating only a slice of each head with RoPE maintains accuracy, so full-width rotation brings little benefit (Black et al., 2022; Barbero et al., 2025). Rotating just that slice and leaving the remaining low-rank channels unrotated cuts the KV cache without harming quality. GQA already reduces cache by letting each query group share one KV head and scales well across devices. Merging grouping with low rank and partial RoPE, we propose *Grouped-Tied Attention* (GTA), which ties KV to a single state and applies partial rotation for further cache reduction.

Tying the key and value projections yields a single *tied KV* vector that matches one key or value. The value path uses the entire vector, while the key path reuses its first half unrotated. The missing RoPE half comes from a separate one-head projection of the hidden state, broadcast to all groups, and concatenated with the unrotated half to complete the key. Ablations show that rotating the shared half harms quality even if it

[1] Benchmarks were performed using the FlashMLA kernel version dated 28 March 2025.

is later inverted, so it stays unrotated. The resulting query, key, and value then enter standard scaled dot product attention.

$$Q \in \mathbb{R}^{B\times L\times h_q\times d_h}, \mathrm{KV}, V, K \in \mathbb{R}^{B\times L\times h_{kv}\times d_h},$$
$$K_{\text{rope}} \in \mathbb{R}^{B\times L\times 1\times d_h/2}, K_{\text{half}} \in \mathbb{R}^{B\times L\times h_{kv}\times d_h/2};$$
$$K_{\text{half}} = \mathrm{KV}[:,:,:,:d_h/2], \quad V = \mathrm{KV}, \quad ;$$
$$K = \left[K_{\text{half}}, \mathrm{broadcast}(K_{\text{rope}}, h_{kv})\right].$$

By tying the KV states, we load a shared vector to chip memory, reuse it for both keys and values, and share it with each query group. This reuse halves memory traffic and the KV cache and roughly doubles arithmetic intensity relative to GQA at the same group size. GQA 4 has four distinct KV heads; GTA 4 has four tied KV heads. The perplexity and downstream performance match GQA.

2.3.2 Grouped Latent Attention (GLA)

MLA compresses the KV cache by storing one latent head of dimension $d_c = 4d_h$ per token. Under tensor parallelism, the key and value-up projection matrices $W^{UK}, W^{UV} \in \mathbb{R}^{(4d_h\times hd_h)}$ are column-partitioned, so every device must keep the full $4d_h$ latent to reconstruct its heads, duplicating the cache across ranks. The aggregate KV cache therefore grows linearly with the tensor parallel degree, eroding per device memory savings and limiting distributed inference efficiency (see Section 2.2).

Grouped Latent Attention (GLA) compresses tokens into h_c latent heads, each of dimension $d_c = 2\,d_h$ (half of MLA's $4\,d_h$). During training, every latent head and its up-projection matrices reconstruct different key and value characteristics for the $g_q = h_q/h_c$ query heads in its group, so an up-projection has column size $g_q d_h$ instead of MLA's $h_q d_h$. After weight absorption, each latent attends only to its query group. Sharding the h_c latent heads across tensor-parallel ranks avoids KV duplication when h_c equals the tensor-parallel degree and reduces it otherwise.

For illustration, take $h_c = 2$ as in our experiments. This choice preserves the logical KV cache at $4d_h$ but halves the per-device cache at $\mathrm{TP} \geq 2$. With $\mathrm{TP} = 2$ GLA splits the latent into $c_{\mathbf{0}}^{\mathrm{KV}}$ and $c_{\mathbf{1}}^{\mathrm{KV}}$ and partitions the query heads accordingly. During decoding, each rank computes attention with its latent and queries, applies its slice of the output projection, then performs an AllReduce to sum the partial outputs.

The larger latent head of MLA ($4d_h$) can exhaust the KV cache per device for long sequences or large batches, limiting the batch size or context length versus GLA. *Example.* GLA–4 (h_c=4, $2d_h$ each) under (TP=4, DP=2) halves the per-device cache, fetches a smaller cache each step, but doubles the logical KV capacity. *Our setup.* We use h_c=2, giving GLA the same logical cache as MLA (d_c=4 d_h) while halving the cache per device when TP$\geq$ 2 and matching MLA quality up to 1.471 B parameters. *Arithmetic intensity.* During decoding, GLA achieves $\approx 2g_q$ (twice GQA); GLA–2 reaches $\approx h_q$ FLOPs per byte - on par with aggressive MQA but with higher quality (Table 1).

$$c_{\mathbf{0}}^{\mathrm{KV}}, c_{\mathbf{1}}^{\mathrm{KV}} \in \mathbb{R}^{B\times L\times 2d_h}, \quad Q_{\mathbf{0}}, Q_{\mathbf{1}} \in \mathbb{R}^{B\times 1\times \frac{h_q}{2}\times(2d_h)}$$
$$W_{\mathbf{0}}^{vo}, W_{\mathbf{1}}^{vo} \in \mathbb{R}^{\left(\frac{h_q}{2}\cdot 2d_h\right)\times D},$$
$$O_{\mathbf{0}} = \mathrm{softmax}\Big(Q_{\mathbf{0}}(c_{\mathbf{0}}^{\mathrm{KV}})^\top\Big)c_{\mathbf{0}}^{\mathrm{KV}}, \quad ,$$
$$O_{\mathbf{1}} = \mathrm{softmax}\Big(Q_{\mathbf{1}}(c_{\mathbf{1}}^{\mathrm{KV}})^\top\Big)c_{\mathbf{1}}^{\mathrm{KV}},$$
$$\tilde{O}_{\mathbf{0}} = O_{\mathbf{0}}W_{\mathbf{0}}^{vo}, \quad \tilde{O}_{\mathbf{1}} = O_{\mathbf{1}}W_{\mathbf{1}}^{vo},$$
$$O = \mathrm{AllReduce}\Big(\tilde{O}_{\mathbf{0}} + \tilde{O}_{\mathbf{1}}\Big).$$

3 System Optimization: Asynchrony & Distributed Offset Calculation

We describe the system optimization to achieve peak performance for MLA, GTA, and GLA on modern hardware such as H100 GPUs. Thanks to very fast specialized matrix multiplication units such as Tensor Cores, we need careful software pipelining of memory loading and tensor core instructions to always keep the tensor cores busy. This is achieved through the warp specialization technique to exploit asynchrony on modern hardware.

3.1 Asynchrony with software pipelining and warp specialization

We use two techniques to overlap compute and memory loading:

1. Software pipelining: we load the next KV block while the current KV block is being used in computation. This classical technique (Lam, 1988) avoids having the tensor cores waiting for memory loading.
2. Warp specialization: we have separate warps performing memory loading with either TMA (tensor memory accelerator) or asynchronous copy (`cp.async` instruction), and separate warps performing matrix-multiply-accumulate (MMA). The former act as *producer* warps, while the latter act as *consumer* warps (Bauer et al., 2014). This is commonly used in matrix multiplication (Thakkar et al., 2023) and attention (Shah et al., 2024a). This decoupling simplifies the software pipelining, allowing the warp scheduler to overlap the memory loading and compute.

3.2 Distributed offset calculation for paged KV

As new attention variants such as MLA, GTA, and GLA stress both the compute and the memory subsystems, one would have to perform memory loading as quickly as possible. Paged KV (Kwon et al., 2023) has become a standard way of storing the KV cache. However, paged KV makes it difficult to use the TMA, a specialized hardware unit that performs address calculation and bound checking to load contiguous blocks of memory. Instead, we use the asynchronous copy instruction (`cp.async`) where each thread separately issues individual load instructions. The challenge is that address computation is surprisingly expensive, as it requires 64-bit integer indexing, which translates to multiple instructions per integer multiplication. We instead have multiple threads in the same warp that cooperate to calculate the addresses. As an example, for head dimension 128, to load a block of size 128 x 128 from global memory to shared memory, we use 128 threads, with 16 threads loading per row:

1. Group 128 threads into 8 groups, each consisting of 16 consecutive threads. Each group g for $g = 0,1,\ldots,7$ will be assigned to load rows $g, g+8, \ldots, g+120$.

2. For each thread t, which belongs to the group $g = \lfloor t/16 \rfloor$, read the page index from the page table in row $g + (t \bmod 16) * 8$. Using the page index, compute the global memory address of the paged KV corresponding to this row and store in registers.

3. For row r in $g, g+8, \ldots, g+120$, all 16 threads t in the same group use warp shuffle to get the global memory address from thread index $g*16 + (r-g)/8$. Use this address to load the KV cache elements corresponding to this row.

We see that each thread only needs to store the address offset of 1 row (instead of 16 rows), as the address offsets of 16 rows assigned to each group are spread across 16 threads.

This enables high efficiency for arbitrary page size (such as page size 1), where page size 1 suffers no slowdown compared to page size 64, despite a much larger number of address computations. This unlocks use cases such as prefix caching (Kwon et al., 2023) with RadixAttention (Zheng et al., 2024) which requires page size 1. We benchmark the speed of paged KV with GLA in appendix B.5, showing a 1.2-1.5x speedup.

4 Experiments and Results

4.1 Model Quality

We train GPT-3–style Llama 3 models with 183 M, 433 M, 876 M, and 1.471 B parameters on FineWeb-Edu (25 B tokens for the smallest model, 50 B for the others) and evaluate validation perplexity on FineWeb-Edu, Wikipedia, C4, Cosmopedia, and Pile—reporting their average in Tables 2 and 3—plus zero-shot accuracy on ARC-Easy, HellaSwag, PIQA, WinoGrande, MMLU, SciQ, and OpenBookQA; full hyperparameters, RoPE settings, and additional results appear in Appendix B.1–B.2.1.

At medium and large scales, GLA-2 reduces validation perplexity on both FineWeb Edu and the five-dataset average compared to MLA. GTA surpasses GQA while removing its KV-cache duplication at low TP. In the large model, GLA-2 and GLA$_q$-2 share the best average perplexity (24.49–24.51) versus MLA's 24.93. The pattern persists at the XL scale: GTA-4 edges GQA-4 (10.12 vs. 10.20 on FineWeb Edu), and GLA stays ahead of MLA (10.21 vs 10.25).

	Medium (433M)		**Large (876M)**	
Method	**FineWeb**	**Avg**	**FineWeb**	**Avg**
MLA	12.561	28.230	11.363	24.929
GLA$_q$-2	**12.433**	27.840	11.276	24.511
GLA-2	12.456	**27.586**	11.293	**24.492**
GTA-4	12.785	29.952	**11.232**	24.994
GQA-4	12.922	30.144	11.340	25.286
MHA	12.979	29.990	11.501	25.837
MQA	13.068	30.524	11.413	25.206

Table 2. Validation perplexities (lower is better) on FineWeb-Edu across two model sizes (medium and large), along with the average perplexity across five datasets (FineWeb-Edu validation set, Cosmopedia, RPV1 C4, RPV1 Wikipedia, and Pile). The lowest perplexity is in bold, and the second lowest is underlined.

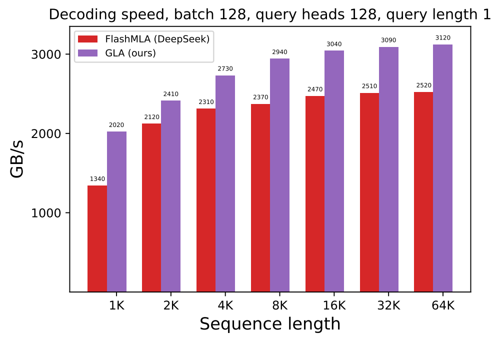


Figure 1. Decoding speed of MLA and GLA on an H100 80 GB SMX5 GPU (theoretical max BF16 compute 989 TFLOPS/s and memory 3350 GB/s) for query length 1.

Method	**FineWeb PPL**	**Avg PPL**	**Avg Downstream**	**KV cache** TP=2
MHA	10.311	21.206	60.1	4096
GQA-4	10.202	21.073	**60.2**	1024
GTA-4	**10.129**	**20.823**	**60.2**	**640**
GLA-2	10.218	21.163	60.0	**640**
MLA	10.256	21.199	59.1	1152

Table 3. Validation perplexity (lower is better) for the 1.471B model on FineWeb-Edu and the average over five validation datasets (FineWeb-Edu, Cosmopedia, RedPajama v1 C4, RedPajama v1 Wikipedia, and Pile). Downstream accuracy (higher is better).

Across downstream benchmarks on the XL scale (1.471 B parameters), our variants meet or exceed baseline accuracy: GLA-2 reaches 60. 0 % versus MLA 59. 1 %, while GTA-4 and GQA-4 each reach 60.2 % (Table 3). Detailed results for smaller models appear in the Appendix B.3.

4.2 Speed

We benchmark the decoding kernels for MLA (1 latent head of dimension 512, RoPE dim 64) and GLA (2 latent heads of dimension 256 each, RoPE dim 64) as shown in Figure 1, with paged KV with page size 64.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL https://arxiv.org/abs/2305.13245.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale, 2022. URL https://arxiv.org/abs/2207.00032.

Federico Barbero, Alex Vitvitskyi, Christos Perivolaropoulos, Razvan Pascanu, and Petar Veličković. Round and round we go! what makes rotary positional encodings useful?, 2025. URL https://arxiv.org/abs/2410.06205.

Michael Bauer, Sean Treichler, and Alex Aiken. Singe: Leveraging warp specialization for high performance on gpus. In *Proceedings of the 19th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 119–130, 2014.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. Gpt-neox-20b: An open-source autoregressive language model, 2022. URL https://arxiv.org/abs/2204.06745.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan Kelly. Reducing transformer key-value cache size with cross-layer attention, 2024. URL https://arxiv.org/abs/2405.12981.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. Palu: Compressing kv-cache with low-rank projection, 2024. URL https://arxiv.org/abs/2407.21118.

Yiting Chen and Junchi Yan. What rotary position embedding can tell us: Identifying query and key weights corresponding to basic syntactic or high-level semantic information. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=e5Mv7iWfVW.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. Magicpig: Lsh sampling for efficient llm generation, 2024. URL https://arxiv.org/abs/2410.16179.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL https://arxiv.org/abs/2307.08691.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer. Ai and memory wall, 2024. URL https://arxiv.org/abs/2403.14123.

Nils Graef and Andrew Wasielewski. Slim attention: cut your context memory in half without loss of accuracy – k-cache is all you need for mha, 2025. URL https://arxiv.org/abs/2503.05840.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, and et. al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Inc. Groq. What is a language processing unit?, 2024. URL https://groq.com/wp-content/uploads/2024/07/GroqThoughts_WhatIsALPU-vF.pdf. Groq white paper.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

Jiaao He and Jidong Zhai. Fastdecode: High-throughput gpu-efficient llm serving using heterogeneous pipelines, 2024. URL https://arxiv.org/abs/2403.11421.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2024. URL https://arxiv.org/abs/2401.18079.

Jingcheng Hu, Houyi Li, Yinmin Zhang, Zili Wang, Shuigeng Zhou, Xiangyu Zhang, Heung-Yeung Shum, and Daxin Jiang. Multi-matrix factorization attention, 2025. URL https://arxiv.org/abs/2412.19255.

Tao Ji, Bin Guo, Yuanbin Wu, Qipeng Guo, Lixing Shen, Zhan Chen, Xipeng Qiu, Qi Zhang, and Tao Gui. Towards economical inference: Enabling deepseek's multi-head latent attention in any transformer-based llms, 2025. URL https://arxiv.org/abs/2502.14837.

Seijin Kobayashi, Yassir Akram, and Johannes Von Oswald. Weight decay induces low-rank attention layers, 2024. URL https://arxiv.org/abs/2410.23819.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Monica Lam. Software pipelining: An effective scheduling technique for vliw machines. In *Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 318–328, 1988.

Jiashi Li. Flashmla: Efficient mla decoding kernels. https://github.com/deepseek-ai/FlashMLA, 2025.

Zhenghao Lin, Zihao Tang, Xiao Liu, Yeyun Gong, Yi Cheng, Qi Chen, Hang Li, Ying Xin, Ziyue Yang, Kailai Yang, Yu Yan, Xiao Liang, Shuai Lu, Yiming Huang, Zheheng Luo, Lei Qu, Xuan Feng, Yaoxiang Wang, Yuqing Xia, Feiyang Chen, Yuting Jiang, Yasen Hu, Hao Ni, Binyang Li, Guoshuai Zhao, Jui-Hao Chiang, Zhongxin Guo, Chen Lin, Kun Kuang, Wenjie Li, Yelong Shen, Jian Jiao, Peng Cheng, and Mao Yang. Sigma: Differential rescaling of query, key and value for efficient language models, 2025. URL https://arxiv.org/abs/2501.13629.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Fanxu Meng, Zengwei Yao, and Muhan Zhang. Transmla: Multi-head latent attention is all you need, 2025. URL https://arxiv.org/abs/2502.07864.

Meta AI. The Llama 4 Herd: The Beginning of a New Era of Natively Multimodal AI Innovation, 2025. URL https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Accessed 29 Apr 2025.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4, 2023. URL https://arxiv.org/abs/2306.02707.

NVIDIA. NVIDIA H100 tensor core gpu architecture. https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper, 2022.

NVIDIA. NVIDIA Blackwell architecture technical brief. https://resources.nvidia.com/en-us-blackwell-architecture, 2024.

NVIDIA Corporation. Nvlink, 2024. URL https://www.nvidia.com/en-us/data-center/nvlink/.

OpenAI. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022. URL https://arxiv.org/abs/2211.05102.

Pol G. Recasens, Ferran Agullo, Yue Zhu, Chen Wang, Eun Kyung Lee, Olivier Tardieu, Jordi Torres, and Josep Ll. Berral. Mind the memory gap: Unveiling gpu bottlenecks in large-batch llm inference, 2025. URL https://arxiv.org/abs/2503.08311.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for kv cache compression, 2024. URL https://arxiv.org/abs/2408.05646.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37: 68658–68685, 2024a.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision, 2024b. URL https://arxiv.org/abs/2407.08608.

Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu, 2023. URL https://arxiv.org/abs/2303.06865.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention, 2024. URL https://arxiv.org/abs/2406.02542.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference, 2024. URL https://arxiv.org/abs/2410.21465.

Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, Matt Nicely, Duane Merrill, Dustyn Blasig, Fengqi Qiao, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Manish Gupta. CUTLASS, January 2023. URL https://github.com/NVIDIA/cutlass.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, and et. al. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Ben Wang and Aran Komatsuzaki. Gpt-j-6b: a 6 billion parameter autoregressive language model, 2021.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009. ISSN 0001-0782. doi: 10.1145/1498765.1498785. URL https://doi.org/10.1145/1498765.1498785.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation, 2023. URL https://arxiv.org/abs/2203.16487.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

Bowen Yang, Bharat Venkitesh, Dwarak Talupuru, Hangyu Lin, David Cairuz, Phil Blunsom, and Acyr Locatelli. Rope to nope and back again: A new hybrid attention strategy, 2025. URL https://arxiv.org/abs/2501.18795.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089.

Yifan Zhang, Yifeng Liu, Huizhuo Yuan, Zhen Qin, Yang Yuan, Quanquan Gu, and Andrew Chi-Chih Yao. Tensor product attention is all you need, 2025. URL https://arxiv.org/abs/2501.06425.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

A Related Work

A.1 Attention Variants

A.1.1 Algorithmic

Pre-training: Follow-up works to DeepSeek's MLA include Multi-matrix Factorization Attention (MFA) (Hu et al., 2025), which resembles MQA but uses a larger head dimension and factorized query projections, where it shares similar limitations with MQA, particularly regarding inefficient KV cache utilization per device due to duplication and lack of compatibility with tensor parallelism. Tensor Product Attention (TPA) (Zhang et al., 2025) factorizes queries, keys, and values with two rank R projection matrices per state, so the per token cache is $(R_K + R_V)(h + d_h)$ elements. We report validation perplexity in our ablations for TPA in the Appendix B.3. The low-rank structure of TPA supports straightforward head-level TP sharding, but its KV cache scales linearly with R and already exceeds MLA once $R \geq 4$; ranks four and above are important for quality, especially at larger scales, so it can quickly lose the memory-saving advantage.

Post-hoc adaptation: In (Lin et al., 2025) they propose SIGMA, which utilizes a novel differentially scaled QKV module specifically optimized and applied during the fine-tuning stage to improve inference efficiency by compressing K while lightly compressing V and increasing Q. Slim Attention (Graef and Wasielewski, 2025), a post-training approach that keeps only the key vectors and recreates the values on the fly, cutting the context memory for any multi-head attention in half. Our proposed method cuts the KV cache further than both of these approaches. It differs from the techniques above, as it attempts to restructure the attention architecture during pre-training, as opposed to changing the existing attention of the already pre-training model. However, similar distillation methods that have been applied to MLA (Meng et al., 2025; Ji et al., 2025) can be adopted for GLA in the post-training stage to realize the benefits of the low KV cache footprint and easy parallelization.

A.1.2 Systems

FlashAttention (Dao et al., 2022) reorders attention computation with an I/O-aware tiling strategy that keeps data in high-speed memory, avoiding the need to materialize the entire attention matrix. It drastically reduces memory overhead and yields significant speedups, particularly during decoding with large sequence lengths. FlashAttention-2 (Dao, 2023) further refines the attention kernel by reducing non-matrix multiplication operations and improving parallel work partitioning, delivering additional gains in hardware utilization. Its system-level improvements provide a notable increase in throughput over the original FlashAttention. Then FlashAttention-3 (Shah et al., 2024b) leverages next-generation GPU features, such as asynchronous memory operations and low-precision computation, pushing attention efficiency closer to hardware limits. Natively trainable sparse attention (NSA) (Yuan et al., 2025) introduces hardware-aligned sparse attention with a dynamic hierarchical pattern. This design reduces computational complexity while maintaining near-full attention fidelity, enabling efficient decoding over extremely long sequences. Our proposed methods are orthogonal to these system-level attention optimizations.

A.2 Additional Approaches to Accelerating Decoding

The following post hoc adaptation design features work in conjunction with the GLA and GTA design architecture, enhancing its performance capabilities.

Algorithmic: There have been many algorithmic efforts such as token eviction (Zhang et al., 2023; Xiao et al., 2024) or sharing KV cache between adjacent layers (Brandon et al., 2024), batching to improve GPU utilization (Mukherjee et al., 2023), and speculative decoding (Xia et al., 2023). **Systems**: On the system side, there has been work on quantization (Hooper et al., 2024), CPU offloading (Aminabadi et al., 2022; Sheng et al., 2023; He and Zhai, 2024), and memory management using PagedAttention (Kwon et al., 2023) to mitigate memory fragmentation problems. **Hardware**: In addition, there have been efforts on the hardware side that benefit inference, such as FP4 support (NVIDIA, 2024) or NVLink (NVIDIA Corporation, 2024), to hardware chips designed solely for fast inference (Groq, 2024).

A.3 Low-Rank Projections

Empirical findings indicate that, before applying RoPE, key activations have a sharply decaying singular-value spectrum (**?**Chen et al., 2024), implying that many dimensions contribute minimally. Furthermore, (Singhania et al., 2024) show that keys exhibit substantially reduced intrinsic dimensionality across models, suggesting an inherently low-rank space. For example, (Kobayashi et al., 2024) finds that the regularization of the weight decay drives the combined key-query mapping to an even lower rank. In contrast, value activations exhibit mixed low-rank tendencies. Some studies have shown that cached values do not compress well without a severe accuracy penalty (Chang et al., 2024; Singhania et al., 2024). However, additional findings demonstrate that partial compression can be achieved with acceptable performance degradation (Saxena et al., 2024; Sun et al., 2024). These inconsistencies suggest that the effective rank of values is model- and method-dependent.

GTA utilizes the insights from these aforementioned works for its design of tying the key and value within each query group, thereby reducing the KV cache size.

A.4 Rotary Position Encoding (RoPE)

Per head dimension (Barbero et al., 2025) suggests that it may not be necessary to apply RoPE (Su et al., 2023) to every head dimension since the highest frequencies already provide positional discrimination. Other studies (DeepSeek-AI, 2024; Black et al., 2022; Wang and Komatsuzaki., 2021) find that they can preserve the quality of the model by applying RoPE to only half of the dimension. Inspired by the same insight, GTA rotates only a partial slice of the head dimension and ties the rest to the portion with half of the value head dimension, reducing the KV cache size while preserving accuracy.

Per layer (Chen and Yan, 2024) shows that RoPE contributes the most in the early transformer layers, where attention is focused on local syntactic relations. In contrast, the deeper layers shift toward semantic cues, implying diminishing returns from positional rotation later in the stack. (Yang et al., 2025) present RNoPE, a design that interleaves RoPE layers with NoPE layers and limits RoPE to a sliding window, achieving markedly better retrieval at very long context lengths and influencing the architecture choices of Llama 4 (Meta AI, 2025). Overall, applying RoPE to partial layers is beneficial for GLA, considering that the decoupled RoPE, with a single head that is half the head dimension, can be eliminated.

B Full Experimental Results

B.1 Experimental Setup

We use the Llama 3 tokenizer (Grattafiori et al., 2024) with a vocabulary size of 128K tokens. We use the AdamW (Loshchilov and Hutter, 2019) optimizer with (β_1,β_2) = (0.9, 0.95), a weight decay of 0.1, and gradient clipping at 1.0. We follow the training recipe from Gu and Dao (2024), using a learning rate scaled by $5\times$ relative to GPT-3 for a model of the same size, with decay of cosine to 1% of the maximum learning rate. We use the configuration of the GPT-3 model (Brown et al., 2020). We first use the configuration of the GPT-3 model for a given parameter size for our MHA baseline, which has the largest parameter budget. Then, we widen the MLPs of every other attention variant until each model matches the MHA parameter count. Essentially, MHA's parameter size is the anchor point.

In GQA-4 & GTA-4, the 4 represents the number of groups or the number of KV heads, $h_{kv} = \frac{h_q}{g_q}$. GLA$_q$ refers to the GLA version in which the latent query is also sharded.

Model Size	**#Param**	**Micro-batch Size**	**Batch Size**	**Learning Rate**	**#Layer**	d_{model}	h_{q}	d_{h}
Small	183.65M	16	512	2.6 x 10^{-4}	12	768	12	64
Medium	433.77M	16	512	1.45 x 10^{-4}	24	1024	16	64
Large	876.55M	8	512	1.2 x 10^{-4}	24	1536	16	96
XL	1471.12M	8	256	1.0 x 10^{-4}	24	2048	16	128

Table 4. Model configuration for the four model sizes in our experiments. We adopted the GPT-3 model configuration, with the Llama 3 architecture as the backbone and its tokenizer as well.

Method	Model Param	Intermediate size
MLA	183.65M	2128
GLA-2	183.51M	2208
MHA	183.45M	2048
MQA	183.53M	2520
GTA-4	183.40M	2462
GQA-4	183.53M	2392

Table 5. Model parameters and FFN intermediate size for a small model.

Method	Model Param	Intermediate size
GLA-2	433.89M	3152
GLA$_q$-2	433.89M	3280
MLA	433.55M	3062
GTA-4	433.57M	3320
GQA-4	433.77M	3248
MHA	433.77M	2736
MQA	433.77M	3376

Table 6. Model parameters and FFN intermediate size for a medium model.

Method	Model Param	Intermediate size
MHA	876.55M	4096
GQA-4	876.55M	4864
MQA	876.55M	5056
GTA-4	876.55M	4976
MLA	876.73M	4640
MLA (d_R : 48)	876.74M	4592
GLA-2 (d_R : 48)	876.96M	4914
GLA-2	876.73M	4768
GLA$_q$-2	876.44M	4936

Table 7. Model parameters and FFN intermediate size for a large model. d_R denotes the RoPE dimension and the default is 32 for this model size

Method	Model Param	Intermediate size
MLA	1470.58M	6120
GLA-2	1470.78M	6292
MHA	1471.12M	5464
GTA-4	1471.22M	6638
GQA-4	1470.83M	6486

Table 8. Model parameters and FFN intermediate size for a XL model.

B.2 Quality

B.2.1 Validation Perplexity

Method	**FineWeb-Edu**	**Cosmopedia**	**RPV1 C4**	**Pile**	**RPV1 Wikipedia**	**Avg**
MHA	16.715	20.542	31.628	40.444	99.800	41.826
GQA-4	16.578	20.599	32.059	43.841	99.525	42.520
MQA	16.972	22.094	32.245	44.308	103.915	43.907
GTA-4	16.607	20.768	32.911	42.181	100.932	42.680
GLA-2	16.371	20.542	31.628	40.444	94.037	40.604
GLA$_q$-2	16.333	20.110	31.517	**38.725**	**92.820**	**39.901**
MLA	**16.318**	**20.063**	**31.484**	39.528	94.056	40.290

Table 9. Validation perplexity for the small model (lower is better). The lowest perplexity is in bold, and the second lowest is underlined. RPV1 refers to RedPajama v1.

Method	FineWeb-Edu	Cosmopedia	RPV1 C4	Pile	RPV1 Wikipedia	Avg
GLA-2	$\underline{12.456}$	**13.722**	$\underline{24.308}$	**27.676**	**59.766**	**27.586**
GLA$_q$-2	**12.433**	$\underline{13.917}$	**24.263**	$\underline{28.224}$	$\underline{60.359}$	$\underline{27.840}$
MLA	12.561	14.039	24.507	28.602	61.438	28.230
GQA-4	12.845	14.532	25.159	30.401	65.871	29.761
GTA-4	12.785	14.812	25.009	30.447	66.708	29.952
MHA	12.979	14.666	25.331	30.772	66.201	29.990
GQA-4 ($qo_R : 4 \cdot d_h ; h_q : 48$)	12.922	15.024	25.282	31.510	65.980	30.144
MQA	13.068	15.163	25.585	31.504	67.302	30.524

Table 10. Validation perplexity for the medium model (lower is better). The lowest perplexity is in bold, and the second lowest is underlined. qo_R refers to the rank of the low-rank query and output projections, resulting in reduced model parameters. To offset these lost parameters for fair comparison with the baselines, we increase the query heads h_q to 48. It's beneficial since arithmetic intensity depends on the number of query heads. RPV1 refers to RedPajama v1.

Method	FineWeb-Edu	Cosmopedia	RPV1 C4	Pile	RPV1 Wikipedia	Avg
MHA	11.501	12.605	22.496	27.651	54.933	25.837
GQA-4	11.340	12.358	22.219	26.635	53.878	25.286
MQA	11.413	12.437	22.383	26.521	53.274	25.206
GTA-4	**11.232**	12.159	$\underline{22.059}$	26.136	53.383	24.994
MLA	11.363	12.468	22.294	25.685	52.837	24.929
MLA ($d_R : 48$)	$\underline{11.245}$	**12.021**	**22.053**	25.246	**52.212**	24.555
GLA$_q$-2 ($d_R : 48$)	11.337	12.144	22.234	**24.620**	52.612	24.589
GLA$_q$-2	11.276	$\underline{12.100}$	22.126	$\underline{24.681}$	52.371	$\underline{24.511}$
GLA-2	11.293	12.106	22.130	24.698	$\underline{52.233}$	**24.492**

Table 11. Validation perplexity for the large model (lower is better). d_R refers to the RoPE dimension and the default is 32 for this model size. RPV1 refers to RedPajama v1.

Method	FineWeb-Edu	Cosmopedia	RPV1 C4	Pile	RPV1 Wikipedia	Avg
MHA	10.311	10.540	20.117	22.432	42.628	21.206
GQA-4	$\underline{10.202}$	$\underline{10.418}$	$\underline{19.986}$	22.642	$\underline{42.119}$	$\underline{21.073}$
GTA-4	**10.129**	**10.399**	**19.849**	**22.184**	**41.551**	**20.823**
GLA-2	10.218	10.482	20.020	$\underline{22.298}$	42.796	21.163
MLA	10.256	10.561	20.041	22.516	42.624	21.199

Table 12. Validation perplexity for the XL model (lower is better). Bold indicates the lowest score in each column; underlined indicates the second lowest. RPV1 refers to RedPajama v1.

Method	FineWeb-Edu PPL	Avg PPL	Avg Downstream	KV cache (bytes/token)		
				TP=1	TP=2	TP=4
MHA	10.311	21.206	$\underline{60.1}$	8192	4096	2048
GQA-4	$\underline{10.202}$	$\underline{21.073}$	**60.2**	$\underline{2048}$	$\underline{1024}$	$\underline{512}$
GTA-4	**10.129**	**20.823**	**60.2**	**1152**	**640**	**384**
GLA-2	10.218	21.163	60.0	**1152**	**640**	640
MLA	10.256	21.199	59.1	**1152**	1152	1152

Table 13. Validation perplexities (lower is better) for the 1.471B model on FineWeb-Edu along with the average perplexity across five datasets (FineWeb-Edu validation, Cosmopedia, RedPajama v1 C4, RedPajama v1 Wikipedia, and Pile). The lowest perplexity is in bold, and the second lowest is underlined. Average Downstream evaluation (higher is better), where the highest accuracy is in bold, the second highest is underlined. TP refers to the tensor parallelism, and we report the KV cache of a token in bytes per device across various TP degrees.

B.2.2 Downstream Evaluation

Method	**Winogrande**	**SciQ**	**PiQA**	**OpenBookQA**	**MMLU**	**HellaSwag**	**Arc-Easy**	**Avg**
GLA$_q-2$	55.2	84.9	**70.5**	35.6	25.2	<u>47.9</u>	**66.3**	<u>55.1</u>
GQA-4$_{qo_R}$	52.4	83.6	69.7	36.0	25.5	45.7	64.9	54.0
GQA-4	53.8	<u>85.7</u>	69.7	36.2	25.4	46.3	64.6	54.5
GTA-4	54.2	85.5	69.0	34.0	<u>25.9</u>	46.8	64.2	54.2
MQA	<u>55.5</u>	84.6	69.5	<u>37.0</u>	**26.2**	45.9	60.5	54.2
GLA-2	**56.7**	84.1	<u>70.3</u>	**37.2**	**26.2**	**48.2**	<u>65.3</u>	**55.4**
MLA	54.5	**86.1**	70.2	36.8	25.1	47.2	64.2	54.9
MHA	55.2	84.8	69.3	35.0	25.5	46.2	63.0	54.1

Table 14. Downstream evaluation for the medium model (higher is better). Bold indicates the highest score in each column; underlined indicates the second highest. qo_R denotes the rank of the low rank query and output projections, set to $4d_h$. To compensate for the reduced parameter count and ensure a fair comparison with the baselines, we increase the number of query heads h_q to 48, which also benefits arithmetic intensity since it scales with the number of query heads.

Method	**Winogrande**	**SciQ**	**PiQA**	**OpenBookQA**	**MMLU**	**HellaSwag**	**Arc-Easy**	**Avg**
GLA-2	57.4	**91.8**	73.9	40.4	**26.1**	58.2	<u>72.1</u>	60.0
GQA-4	<u>59.0</u>	<u>91.5</u>	<u>74.1</u>	**41.6**	25.2	<u>58.5</u>	71.6	**60.2**
GTA-4	58.2	91.0	**75.1**	40.8	25.3	**58.6**	**72.5**	**60.2**
MLA	56.4	89.5	73.5	39.4	25.3	58.1	71.8	59.1
MHA	**60.5**	90.7	73.1	<u>41.0</u>	<u>25.9</u>	57.6	71.9	<u>60.1</u>

Table 15. Downstream evaluation for the XL model (higher is better). Bold indicates the highest score in each column; underlined indicates the second highest.

B.3 Ablations

Different attention variants reduce the learned parameters and the representational capacity per layer; therefore, these saved parameters need to be redistributed elsewhere. For example, Llama 2 (Touvron et al., 2023) increases the width of the FFNs for MQA and GQA in their ablation to make a fair comparison to MHA. In addition, MQA initially proposed to increase the width to match the parameters to MHA (Shazeer, 2019). Meanwhile, (DeepSeek-AI, 2024) adjusts the depth of the model, increasing the number of layers for a fair comparison. Altering the depth is less common because it is challenging to make head-to-head comparisons, as there is less flexibility in moderately scaled models to match parameters. (Pope et al., 2022) shrink the head dimension of MHA to match the parameters of MQA and TPA, while (Zhang et al., 2025) increases the number of query heads to align the parameter count, essentially distributing the saved parameters into the query projections. For instance, in the case of GQA and GTA, the KV heads need to be divisible by the query heads, so there is less flexibility in terms of altering the number of query heads to match as closely as possible to the baseline for fair comparison. In the case of MLA and GLA, increasing the query heads is beneficial since, during decoding, we do not materialize KV. Essentially, a single latent head is shared across all or groups of query heads; therefore, increasing the query heads trivially improves GPU utilization while decoding, as we demonstrated earlier in Table 1 where the arithmetic intensity for MLA and GLA boils down to the number of query heads.

B.3.1 Ablations: Small Model 188M Parameters

Attention Type	Model Param	FineWeb-Edu	RPV1 C4	Cosmopedia	RPV1 Wikipedia	Pile	Avg
MHA	183.45M	16.71	32.24	$\underline{20.54}$	99.79	40.44	41.94
GQA-4	174.01M	17.07	32.93	21.62	104.1	45.09	44.16
MQA	170.47M	17.39	33.70	22.53	108.0	46.79	45.68
GTA-4	171.95M	17.04	32.95	21.82	103.5	44.67	44.00
TPA (r=2)	172.10M	17.06	32.92	21.81	99.58	44.17	43.11
TPA (r=4)	174.90M	16.90	32.63	21.59	99.43	43.15	42.74
MLA	181.44M	**16.40**	**31.61**	**20.49**	$\underline{95.46}$	**40.04**	$\underline{40.80}$
GLA$_q$-2	175.54M	$\underline{16.56}$	$\underline{31.91}$	20.66	**94.49**	$\underline{40.13}$	**40.75**

Table 16. We ablate by keeping the width of the FFNs (2048) and number of query heads ($h_q : 12$) constant across the attention variants. Validation perplexity for the small model (lower is better). Bold marks the lowest value in each column, and underlined marks the second-lowest. We include the benchmark for Tensor Product Attention (TPA) (Zhang et al., 2025) with ranks 2 and 4 for the low-rank projection matrices of keys and values. Bold marks the lowest value in each column, and underlined marks the second-lowest. RPV1 refers to RedPajama v1.

Method	FineWeb-Edu	RPV1 C4	Cosmopedia	RPV1 Wikipedia	Pile	Avg
GLA$_q$-2 (h_q:20)	16.450	31.706	20.849	95.273	40.116	40.879
MLA	$\underline{16.338}$	**31.516**	$\underline{20.111}$	$\underline{94.273}$	$\underline{39.168}$	$\underline{40.281}$
GLA$_q$-2	**16.337**	$\underline{31.517}$	**20.110**	**92.820**	**38.726**	**39.902**
GTA-4 (d_R:16)	16.870	32.732	21.089	103.508	43.103	43.460
GTA-4 ($qo_R : 4$ h_q:24)	16.517	31.946	20.727	99.962	41.462	42.123
GTA-4 ($qo_R : 3$ h_q:36)	16.496	32.048	20.537	101.030	43.787	42.780
GQA-4 ($qo_R : 3$ h_q:36)	16.546	32.048	20.786	99.684	43.787	42.570
GQA-4 ($qo_R : 4$ h_q:24)	16.405	31.754	20.530	97.979	43.302	41.994

Table 17. The ablations are for different query head counts and projection ranks. Given that arithmetic intensity during decoding depends on the number of query heads, we increase the number of query heads, h_q, and the query and output projections are low-rank, denoted by qo_R, to compensate for the added parameters. d_R denotes the RoPE dimension. For instance, $d_R = 16$ for GTA-4, we apply RoPE to only 25% of the head dimensions instead of 50% in our proposed approach. Validation perplexity for the small model (lower is better). Bold marks the lowest value in each column, and underlined marks the second-lowest. RPV1 refers to RedPajama v1.

Method	Model Param	h_q	FineWeb-Edu	RPV1 C4	Cosmopedia	RPV1 Wikipedia	Pile	Avg
MHA	183.45M	12	16.71	32.24	20.54	99.79	40.44	41.94
MQA	183.45M	23	17.03	32.97	21.79	104.30	44.51	44.12
GQA-4	183.45M	20	16.79	32.44	21.19	101.50	43.05	42.99
GTA-4	181.40M	20	16.83	32.58	21.22	104.70	44.46	43.96
GTA-4	186.11M	24	16.55	32.07	$\underline{20.49}$	99.13	42.78	42.20
TPA (r=2)	183.05M	21	16.74	32.33	22.32	103.60	43.32	43.66
TPA (r=4)	183.68M	19	16.75	32.32	21.63	99.78	41.81	42.46
MLA	183.02M	13	**16.33**	$\underline{31.70}$	20.84	$\underline{95.27}$	**39.16**	$\underline{40.66}$
GLA$_q$-2	183.51M	13	$\underline{16.44}$	**31.51**	**20.11**	**94.27**	$\underline{40.11}$	**40.49**

Table 18. We ablate by keeping the width of the FFNs (2048) constant across different variants, but increasing the query heads, h_q, to match the parameters for fair comparison. Recall that the arithmetic intensity of attention during decoding depends on the number of query heads. Validation perplexity for the small model (lower is better) across different numbers of h_q and identical FFN width. Bold marks the lowest value in each column, and underlined marks the second-lowest.

B.3.2 Ablations: Medium Model 433M Parameters

In the primary experiment, the medium model (433 M) is trained on 50B tokens, whereas the ablation studies and baseline within this section are trained on 25B tokens.

Method	**Model Param**	**FineWeb-Edu**	**RPV1 C4**	**Cosmopedia**	**RPV1 Wikipedia**	**Pile**	**Avg**
GTA-4	433.57M	13.250	25.804	15.051	70.733	31.091	31.19
MLA	433.55M	13.066	25.367	14.515	<u>66.667</u>	30.027	29.93
GLA-2	433.60M	<u>12.985</u>	<u>25.216</u>	**14.422**	**65.730**	**29.515**	**29.57**
GLA$_q$-2	433.89M	**12.957**	**25.108**	<u>14.434</u>	67.182	29.909	<u>29.92</u>
MLA	433.55M	13.087	25.411	14.582	66.847	<u>29.725</u>	29.93
GQA-4	433.77M	13.395	25.999	15.211	70.403	31.650	31.33
MHA	433.77M	13.552	26.286	15.330	72.488	32.124	31.96
MQA	433.77M	13.574	26.436	15.615	72.789	33.513	32.39
TPA (r=2)	433.77M	13.186	25.612	15.186	68.709	31.269	30.79
TPA (r=4)	433.96M	13.143	25.538	14.672	66.877	30.396	30.12

Table 19. The width of the FFN is modified to match parameters as closely as possible across variants. They are all trained on 25B tokens. Validation perplexity for the medium model (lower is better). Bold marks the lowest value in each column, and underlined marks the second-lowest. We benchmark TPA using low rank key and value projection matrices at ranks 2 and 4. RPV1 refers to RedPajama v1.

Method	**Model** Param	**Winogrande**	**SciQ**	**PiQA**	**OpenBook** QA	**MMLU**	**HellaSwag**	**Arc** Easy	**Avg**
TPA (r=2)	433.77M	53.8	84.1	68.3	36.3	<u>25.8</u>	45.4	63.5	53.8
TPA (r=4)	433.96M	51.8	83.7	68.6	35.4	25.2	45.5	<u>65.7</u>	53.6
GTA-4	433.57M	<u>55.2</u>	84.3	69.2	34.9	**26.0**	45.3	63.1	54.0
MLA	433.55M	54.0	82.9	69.3	**39.9**	25.4	46.1	65.4	**54.7**
GLA-2	433.60M	**55.5**	83.8	**70.0**	35.2	25.5	<u>46.2</u>	**66.6**	<u>54.6</u>
GLA$_q$-2	433.89M	54.3	**85.9**	<u>69.4</u>	37.2	24.9	**46.3**	63.8	54.5
MLA	433.55M	53.2	83.9	69.3	**39.9**	25.4	45.9	64.3	54.5
GQA-4	433.77M	55.0	82.8	69.2	34.5	25.3	45.0	63.6	53.6
MHA	433.77M	51.7	<u>85.5</u>	69.3	35.6	25.4	44.2	62.8	53.5
MQA	433.77M	51.5	83.7	68.3	<u>37.4</u>	25.7	44.4	62.6	53.3

Table 20. The width of the FFN is modified to match parameters as closely as possible across variants. All models are trained on 25 B tokens. Downstream evaluation for the medium model (higher is better). Bold indicates the highest score in each column; underlined indicates the second highest. We benchmark TPA using low-rank key and value projection matrices at ranks 2 and 4. RPV1 refers to RedPajama v1.

Method	**Model** Param	h_q	**FineWeb-Edu** PPL	**Avg** PPL	**Avg** Downstream	**KV Cache (bytes/token)** TP=1	TP=2
GLA-2	434.73M	26	**13.236**	**30.358**	53.9	576	320
MQA	433.77M	31	13.703	33.022	53.4	256	256
GQA-4	433.77M	28	13.567	32.019	52.6	1024	512
GTA-4	428.26M	28	13.401	31.475	53.6	576	320
GLA$_q$-2	434.76M	32	13.321	30.909	53.4	576	320
MLA	434.32M	23	<u>13.249</u>	<u>30.875</u>	**54.2**	576	576
MHA	433.77M	16	13.552	31.956	53.5	4096	2048
TPA(r=2)	433.47M	29	13.367	31.171	53.2	744	624
TPA(r=4)	432.59M	26	13.404	31.030	<u>54.0</u>	1440	1232

Table 21. We run ablation by keeping the width of the FFNs (2736) constant across different variants but increasing the query heads, h_q, to match the parameters for fair comparison. Recall that the arithmetic intensity of attention during decoding depends on the number of query heads. We report the validation perplexity (lower is better) for FineWeb-Edu, along with the average perplexity across five datasets: FineWeb-Edu validation set, Cosmopedia, RedPajama v1 C4, RedPajama v1 Wikipedia, and Pile. The lowest perplexity is in bold, and the second lowest is underlined. We report the average downstream evaluation (higher is better), where the highest accuracy is in bold, and the second-highest is underlined. TP refers to the tensor parallelism, and we report the KV cache of a token in bytes per device across various TP degrees. We benchmark TPA using low-rank key and value projection matrices at ranks 2 and 4. RPV1 refers to RedPajama v1.

Method	**Model Param**	h_q	**FineWeb-Edu**	**RPV1 C4**	**Cosmopedia**	**RPV1 Wikipedia**	**Pile**	**Avg**
GLA-2	434.73M	26	**13.236**	**25.710**	**14.665**	**67.764**	**30.416**	**30.358**
MQA	433.77M	31	13.703	26.721	15.732	75.651	33.305	33.022
GQA-4	433.77M	28	13.567	26.341	15.514	72.141	32.534	32.019
GTA-4	428.26M	28	13.401	26.036	15.217	71.095	31.628	31.475
GLA$_q$-2	434.76M	32	13.321	25.859	<u>15.085</u>	69.604	<u>30.677</u>	30.909
MLA	434.32M	23	<u>13.249</u>	<u>25.734</u>	15.089	<u>69.569</u>	30.735	<u>30.875</u>
MHA	433.77M	16	13.552	26.286	15.330	72.488	32.124	31.956
TPA(r=2)	433.47M	29	13.367	25.936	15.322	70.091	31.138	31.171
TPA(r=4)	432.59M	26	13.404	26.059	15.409	69.805	30.473	31.030

Table 22. We ablate by keeping the width of the FFNs (2736) constant across different variants but increasing the number of query heads, h_q, to match the parameters for fair comparison. Recall that the arithmetic intensity of attention during decoding depends on the number of query heads. Validation perplexity for the medium model (lower is better). Bold marks the lowest value in each column, and underlined marks the second-lowest. There is less flexibility for GQA and GTA to match parameters since the number of KV heads h_{kv} needs to be divisible by the h_q. We benchmark TPA using low-rank key and value projection matrices at ranks 2 and 4. RPV1 refers to RedPajama v1.

Method	**Model** Param	**Winogrande**	**SciQ**	**PiQA**	**OpenBook** QA	**MMLU**	**HellaSwag**	**Arc** Easy	**Avg**
GLA	434.73M	52.5	84.1	69.2	36.3	25.6	**45.4**	<u>64.3</u>	<u>53.9</u>
MQA	433.77M	**54.6**	83.9	67.9	34.7	<u>25.8</u>	43.7	63.6	53.4
GQA-4	433.77M	52.2	82.3	68.0	34.9	24.9	43.9	62.6	52.6
GTA-4	428.26M	53.4	85.1	68.2	34.9	**25.9**	44.8	63.6	53.6
GLA$_q$-2	434.76M	53.2	83.7	68.4	35.2	25.1	44.8	63.6	53.4
MLA	434.32M	<u>53.9</u>	**85.6**	**69.5**	<u>35.4</u>	24.4	<u>44.9</u>	**65.9**	**54.2**
MHA	433.77M	51.7	<u>85.5</u>	<u>69.3</u>	**35.6**	25.4	44.2	62.8	53.5
TPA(r=2)	433.47M	51.9	84.3	68.9	35.0	25.1	44.7	62.1	53.2
TPA(r=4)	432.59M	52.9	83.3	68.4	38.2	25.8	44.7	65.1	54.0

Table 23. We run ablations by keeping the FFN width (2736) constant across variants while increasing the number of query heads h_q to match parameters for fair comparison. The arithmetic intensity of attention during decoding depends on h_q. Downstream evaluation for the medium model (higher is better). Bold indicates the highest score in each column; underlined indicates the second-highest. GQA and GTA have less flexibility because h_{kv} must divide h_q. TPA is benchmarked with low-rank key and value projections at ranks 2 and 4. RPV1 = RedPajama v1.

B.4 Per Token KV Cache Size per Device

Method	KV cache per Token	KV cache per token per Device (2 GPUs)	KV cache per token per Device (4 GPUs)	KV cache per token per Device (8 GPUs)
MHA	$64d_h$	$32d_h$	$16d_h$	$8d_h$
GQA-4	$16d_h$	$8d_h$	$4d_h$	$2d_h$
MQA	$2d_h$	$2d_h$	$2d_h$	$2d_h$
MLA	$4.5d_h$	$4.5d_h$	$4.5d_h$	$4.5d_h$
GLA-2	$4.5d_h$	$2.5d_h$	$2.5d_h$	$2.5d_h$
GTA-4	$8.5d_h$	$4.5d_h$	$2.5d_h$	$1.5d_h$

Table 24. An example of KV cache per token for llama 3 8B model configuration with $h_q : 32$ and $h_{kv} : 8$ across various TP degrees. d_h denotes head dimension.

B.5 Speed

We show here that our technique (distributed offset calculation) significantly speeds up the attention kernel when using paged KV. Typically, the attention kernel speed slows down when the page size is small since there is more overhead of address calculation (Kwon et al., 2023). However, a smaller page size reduces fragmentation and unlocks new use cases, such as prefix caching (Zheng et al., 2024). We benchmark the speed of the decoding kernels for GLA (2 latent heads of dimension 256 each, RoPE dimension 64) with paged KV, as shown in Figure 2. We compare page size 1 and page size 64, with or without distributed offset calculation. With distributed offset calculation, page size 1 does not suffer from the slowdown, matching the

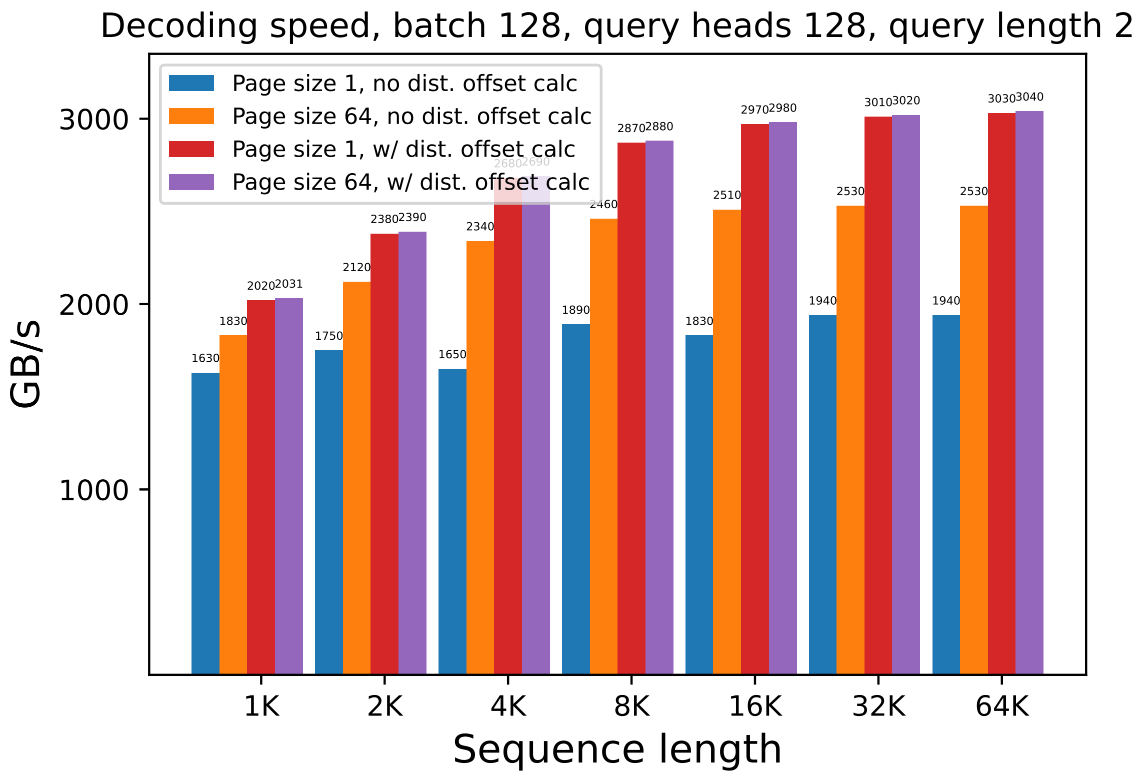


Figure 2. Decoding speed of GLA on H100 80GB SMX5 GPU (theoretical max BF16 compute 989 TFLOPS/s and memory 3350 GB/s), for query length 2, with BF16 format. Distributed offset calculation gives 1.2-1.5x speedup, allowing page size 1 to match the speed of page size 64.

speed of page size 64. On the other hand, without distributed offset calculation, page size 1 is 1.3× slower than page size 64. We see that the distributed offset calculation gives a speedup of 1.2× for page size 64 and a speedup of 1.5× for page size 1.

B.5.1 Tensor Parallelism: GLA vs. MLA

In this configuration with TP degree 8 across x8 H100 GPUs, GLA-8 employs eight latent heads, where each token has to cache a latent dimension of 256, whereas MLA maintains a 512-dimensional latent cache duplicated across devices. Both methods have decoupled the RoPE dimension of 64. Figures 3 and Table 25 reveal consistent gains for GLA-8 at every load level. With 16 concurrent requests, GLA-8 reduces the median end-to-end latency from 136 to 117 seconds, a reduction of approximately 15%, while increasing token throughput by approximately 17%. When the concurrency limit increases to 64, GLA-8 completes in 179 seconds compared to 381 seconds for MLA, cutting latency by 53% percent; the first token now arrives after 12 seconds rather than about 3 minutes, and throughput grows by about 70% to 1461 tokens per second. Even with 128 concurrent requests, GLA-8 still reduces latency by around 24% and maintains a throughput lead of nearly 60%. These advantages stem from the smaller KV cache footprint of GLA-8 per device, which reduces memory traffic, allows more active requests to fit on the GPUs, and shortens the waiting time before computation can begin.

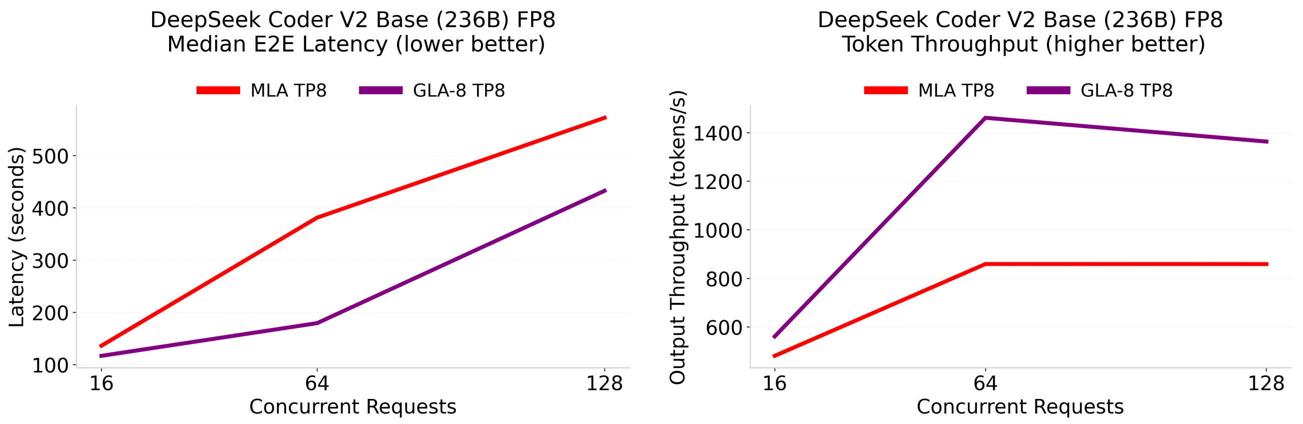


Figure 3. Median end-to-end latency (left), lower is better, and output throughput (right), higher is better, of MLA and GLA-8 under pure TP service on eight GPUs. Prefill/Decode are fixed at 8 K/4 K tokens as concurrency is swept over 16, 64, 128. Because GLA-8 stores roughly half the KV-cache per token under a TP degree of 8, it fetches less data during decoding and consistently outperforms MLA.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Median** E2E Latency (s)	**Median** TTFT (s)	**Median** ITL (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	8K/4K	16/1280	116.83	3.74	27.10	561.06
MLA *(TP8)*	8K/4K	16/1280	136.23	3.98	31.77	481.09
GLA-8 *(TP8)*	8K/4K	64/1280	179.32	11.96	38.16	1460.61
MLA *(TP8)*	8K/4K	64/1280	381.13	192.70	43.03	858.95
GLA-8 *(TP8)*	8K/4K	128/1280	432.54	223.09	45.99	1362.84
MLA *(TP8)*	8K/4K	128/1280	572.20	392.07	43.04	858.69

Table 25. Median service-level metrics for MLA and GLA on x8 GPU TP server; the table reports end-to-end latency, time to first token, inter-token latency, and output throughput at concurrency limits of 16, 64, and 128. GLA surpasses MLA on every measure, cutting latency by more than half and lifting throughput by roughly 70% at the mid-load point of 64 concurrent requests.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Mean** E2E Latency (s)	**Mean** TTFT (s)	**Mean** ITL (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	8K/4K	16/1280	116.80	3.59	27.64	561.06
MLA *(TP8)*	8K/4K	16/1280	136.21	3.83	32.32	481.09
GLA-8 *(TP8)*	8K/4K	64/1280	179.45	11.94	40.90	1460.61
MLA *(TP8)*	8K/4K	64/1280	301.57	118.99	44.58	858.95
GLA-8 *(TP8)*	8K/4K	128/1280	370.62	168.84	49.27	1362.84
MLA *(TP8)*	8K/4K	128/1280	589.07	406.52	44.58	858.69

Table 26. Mean service-level metrics for MLA and GLA-8 on x8 GPU TP server; the table reports end-to-end latency, time to the first token, inter-token latency, and output throughput at concurrency limits of 16, 64, and 128

B.5.2 Data Parallelism + Tensor Parallelism: GLA vs. MLA

Figures 4 and Table 27 demonstrate how the balance between compute capacity and memory traffic changes when parallel data attention is introduced. Under mixed TP 2 + DP 4, as shown in Figures 4 and Table 27, GLA-2 with two latent heads each with dimension 256, shortens the median end-to-end latency from 137 to 120 seconds and increases throughput from 477 tokens per second to 544 tokens per second at a light load of 16 concurrent requests. At 64 concurrency, the advantage grows to roughly 16% lower latency (196 s relative to 166 s) and 19% higher throughput of 1334 tokens per second vs. 1584 tokens per second. Similarly, in mixed TP 4 + DP 2, as shown in Figures 5 and Table 29, GLA-4 consistently outperforms

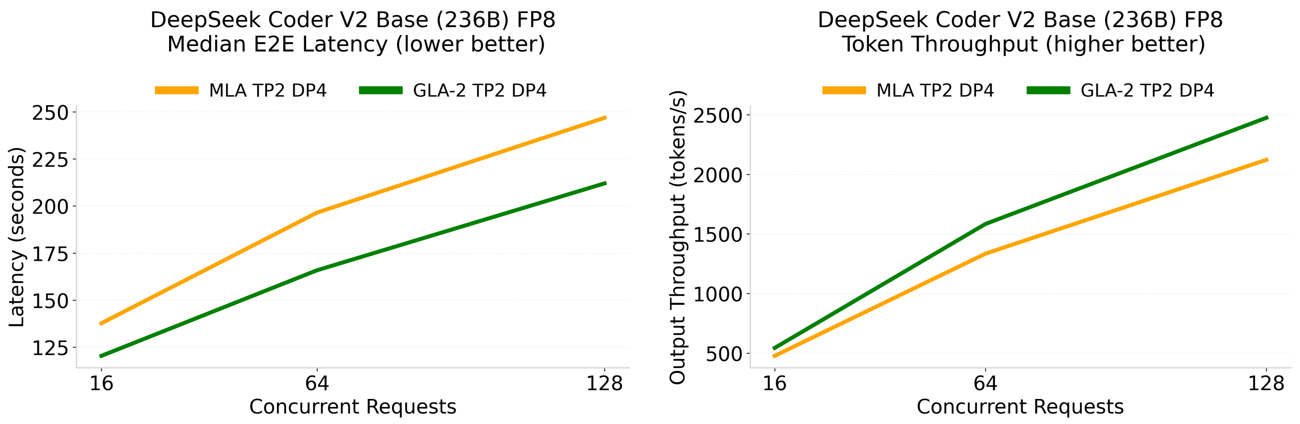


Figure 4. Median end-to-end latency (left), lower is better, and output throughput (right), higher is better, of MLA and GLA under expert parallelism across 8 GPUs with 4 DP groups solely for attention. Prefill/Decode are fixed at 8K/4K tokens as concurrency is swept over 16, 64, 128. GLA outperforms MLA consistently under various concurrencies.

MLA under various concurrent requests.

However, when the limit reaches 128 requests, as shown in Figures 6 and Figures 7, MLA with the hybrid of TP with degree 2 and DP with degree 4, overtakes GLA-8 in pure TP with degree 8 by using the extra replicas to spread the batch and saturate all compute units; MLA now delivers about 56% more tokens per second (2122 tokens per second vs. 1363 tokens per second) and finishes roughly 43% earlier (247 seconds relative to 433 seconds). The cross-over occurs because the added compute lanes of data parallelism offset its cache duplication overhead once the server is heavily loaded. In contrast, GLA-8 in pure TP has already reached the memory bandwidth ceiling and cannot scale further, demonstrating that data parallelism is useful only at large concurrency.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Median** E2E Latency (s)	**Median** TTFT (s)	**Median** ITL (ms)	**Output Throughput** (token/s)
GLA-2 *(TP2, DP4)*	8K/4K	16/1280	120.43	5.56	27.95	543.77
MLA *(TP2, DP4)*	8K/4K	16/1280	137.33	5.92	31.97	477.30
GLA-2 *(TP2, DP4)*	8K/4K	64/1280	165.86	14.12	35.01	1583.51
MLA *(TP2, DP4)*	8K/4K	64/1280	196.47	14.78	42.35	1334.18
GLA-2 *(TP2, DP4)*	8K/4K	128/1280	211.98	25.32	40.90	2474.20
MLA *(TP2, DP4)*	8K/4K	128/1280	246.81	26.93	49.12	2121.88

Table 27. Median service-level results for GLA-2 and MLA when both run with eight-way tensor parallelism and four-way data parallel attention. GLA-2 shows lower latency and higher throughput at the two lighter loads; at the heaviest load, MLA narrows the gap, but GLA-2 still leads by about 14% on latency (lower is better) and throughput (higher is better).

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Mean** E2E Latency (s)	**Mean** TTFT (s)	**Mean** ITL (ms)	**Output Throughput** (token/s)
GLA-2 *(TP2, DP4)*	8K/4K	16/1280	120.51	5.18	28.16	543.77
MLA *(TP2, DP4)*	8K/4K	16/1280	137.29	5.50	32.18	477.30
GLA-2 *(TP2, DP4)*	8K/4K	64/1280	165.52	14.20	36.95	1583.51
MLA *(TP2, DP4)*	8K/4K	64/1280	196.46	14.76	44.37	1334.18
GLA-2 *(TP2, DP4)*	8K/4K	128/1280	211.86	25.39	45.53	2474.20
MLA *(TP2, DP4)*	8K/4K	128/1280	247.04	26.57	53.84	2121.88

Table 28. Mean service-level results for GLA-2 and MLA when both run with eight-way tensor parallelism and four-way data parallel attention. GLA-2 shows lower latency under various metrics.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Median** E2E Latency (s)	**Median** TTFT (s)	**Median** ITL (ms)	**Output Throughput** (token/s)
GLA-4 *(TP4, DP2)*	8K/4K	16/1280	118.34	4.51	27.48	553.29
MLA *(TP4, DP2)*	8K/4K	16/1280	135.86	4.71	31.66	482.42
GLA-4 *(TP4, DP2)*	8K/4K	64/1280	170.66	12.80	36.07	1542.96
MLA *(TP4, DP2)*	8K/4K	64/1280	205.39	13.36	44.51	1276.25
GLA-4 *(TP4, DP2)*	8K/4K	128/1280	222.36	23.87	43.36	2357.85
MLA *(TP4, DP2)*	8K/4K	128/1280	462.03	237.35	49.66	1341.89

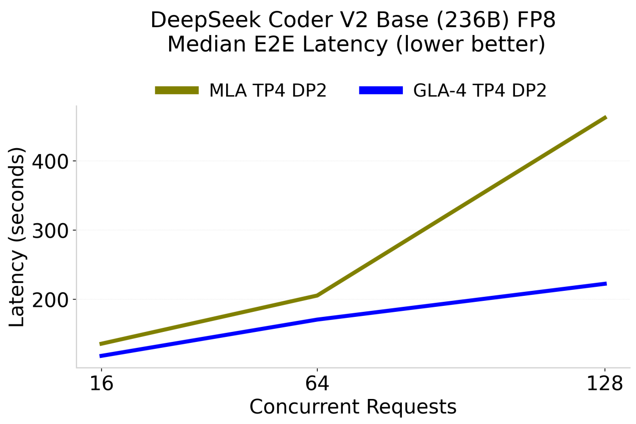
Table 29. Median service-level results for GLA-4 and MLA when both run with eight-way tensor parallelism and four-way data parallel attention. GLA-4 shows slightly lower latency (lower is better) and higher throughput (higher is better) at the two lighter concurrent requests; at the heaviest load GLA-4 performs significantly better than MLA.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Mean** E2E Latency (s)	**Mean** TTFT (s)	**Mean** ITL (ms)	**Output Throughput** (token/s)
GLA-4 *(TP4, DP2)*	8K/4K	16/1280	118.44	4.21	27.89	553.29
MLA *(TP4, DP2)*	8K/4K	16/1280	135.84	4.44	32.09	482.42
GLA-4 *(TP4, DP2)*	8K/4K	64/1280	169.87	12.77	38.36	1542.96
MLA *(TP4, DP2)*	8K/4K	64/1280	205.37	13.38	46.88	1276.25
GLA-4 *(TP4, DP2)*	8K/4K	128/1280	222.30	23.87	48.46	2357.85
MLA *(TP4, DP2)*	8K/4K	128/1280	380.39	165.82	52.40	1341.89

Table 30. Mean service-level results for GLA-4 and MLA when both run with eight-way tensor parallelism and four-way data parallel attention.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Median** E2E Latency (s)	**Median** TTFT (s)	**Median** ITL (ms)	**Output Throughput** (token/s)
GLA-2 *(TP8)*	32K/4K	16/1280	166.18	18.11	32.47	394.76
MLA *(TP2, DP4)*	32K/4K	16/1280	188.37	36.13	34.02	347.88
GLA-2 *(TP8)*	64K/4K	16/1280	219.90	61.94	35.70	224.29
MLA *(TP2, DP4)*	64K/4K	16/1280	313.68	118.37	37.00	208.59

Table 31. Median service-level results for GLA-2 only with eight-way tensor parallelism and MLA under mix parallelism scheme with eight-way tensor parallelism and four-way data parallel attention. GLA-2 has 14% higher throughput (higher is better) relative to MLA for prefill length of 32K while 7% higher throughput for 64K prefill length.



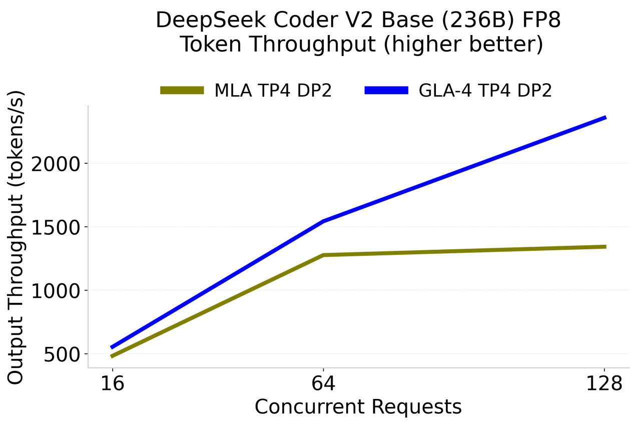
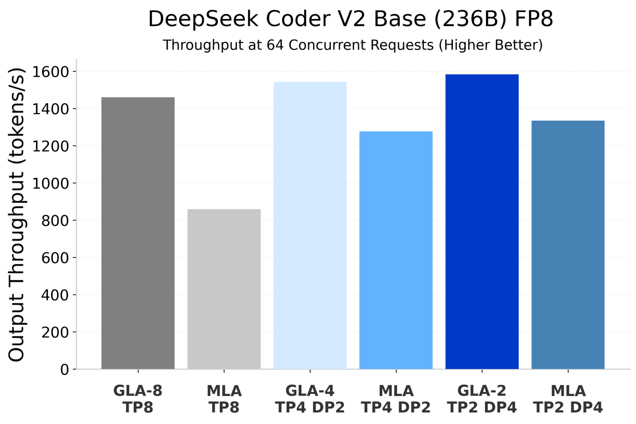


Figure 5. Mean service-level metrics for MLA and GLA on x8 GPU TP server; the table reports end-to-end latency, time to first token, inter-token latency, and output throughput at concurrency limits of 16, 64, and 128 with fixed prefill/decode length of 8K/4K



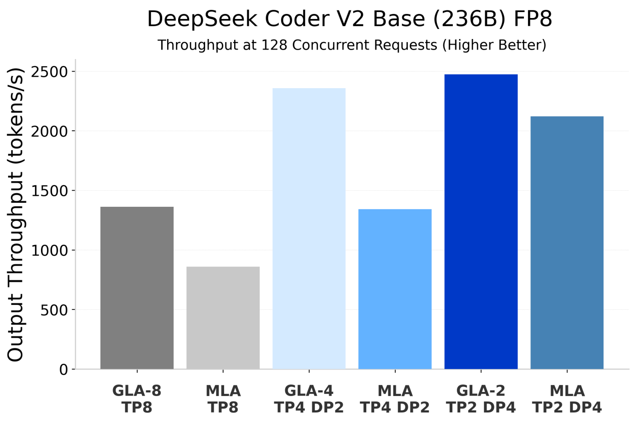


Figure 6. Token throughput at 64 concurrent requests (left) and 128 concurrent requests (right), where higher is better. The prefill/decode sequence length is 8192/4096. GLA outperforms MLA under equivalent parallelism configurations, but MLA with a hybrid of TP and DP at the 128 concurrent requests has higher throughput than GLA under pure TP.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Mean** E2E Latency (s)	**Mean** TTFT (s)	**Mean** ITL (ms)	**Output Throughput** (token/s)
GLA-2 *(TP8)*	32K/4K	16/1280	166.00	18.09	36.12	394.76
MLA *(TP2, DP4)*	32K/4K	16/1280	188.37	31.25	38.36	347.88
GLA-2 *(TP8)*	64K/4K	16/1280	291.90	112.39	43.63	224.29
MLA *(TP2, DP4)*	64K/4K	16/1280	314.16	102.16	51.77	208.59

Table 32. Mean service-level results for GLA-2 only with eight-way tensor parallelism and MLA under mix parallelism scheme with eight-way tensor parallelism and four-way data parallel attention.

B.5.3 Data Parallelism: Workload Imbalance

The random ratio parameter is a fraction of the minimum length that the benchmarks' random-request generator may assign to any individual prefill or decode sequence. For example, with a random ratio of 0.125 and a sequence length of 4096 tokens, each request is created with lengths drawn uniformly from the integer range of 512 to 4096 tokens, giving every batch a consistent lower bound while retaining a realistic spread of sequence sizes. The random ratio is applied to prefill sequence

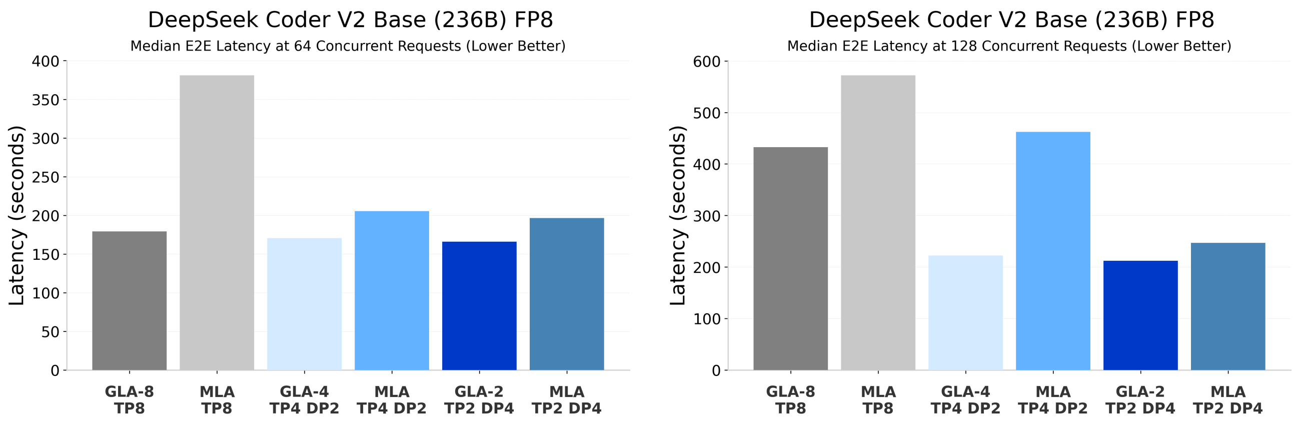


Figure 7. Median E2E latency at 64 concurrent request (left) and 128 concurrent request (right), where higher is better. The prefill/decode sequence length is 8192/4096. GLA outperforms MLA under equivalent parallelism configurations, but MLA with a hybrid of TP and DP at the 128 concurrent requests has higher throughput than GLA-8 under pure TP.

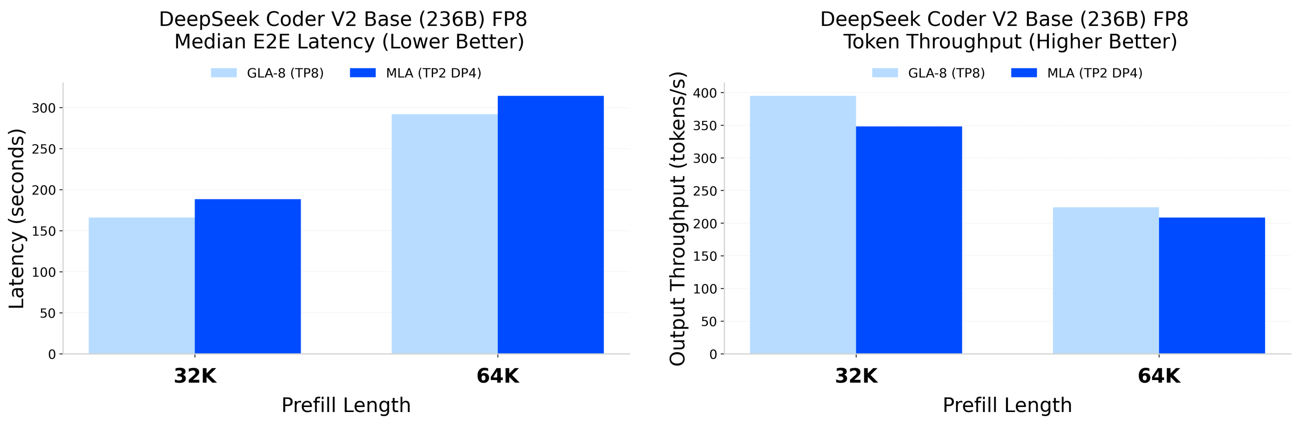


Figure 8. Median end-to-end latency (left), lower is better, and output throughput (right), higher is better, under TP and DP groups solely for attention and expert parallelism for GLA and MLA respectively, across 8 GPUs. Demonstrating long-context with a moderately high number of concurrent requests.

lengths (131K) and decode (4K). The experiments in this section demonstrate workload imbalance with varying sequence lengths across the batch, which can leave GPUs idle.

In Figure 9 and Table 33, we demonstrate where for a long prefill of 131K and a relatively long decode of 4K, where the sequence length is uniformly sampled within the batch, GLA-8 with TP degree 8 has about 2.7× higher throughput than MLA in hybrid TP with degree 2 in four data parallel ranks. Because every NCCL collective in a data-parallel group must be entered by all ranks, one replica that is still busy with a very long sequence forces every other replica, and its tensor-parallel shards, to wait, so throughput collapses to the speed of that single straggler, with pure TP-8, there is no extra data-parallel barrier, so only the eight shards that hold the weights pause for one another; a long sequence slows that shard group, but leaves the rest of the cluster working, keeping GPU utilization much higher.

Method	**Prefill/Decode length**	**Rand. ratio**	**Max conc. /#Prompts**	**Median E2E Latency** (s)	**Median TTFT** (s)	**Median ITL** (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	131K/4K	0	4/1280	80.21	6.42	25.10	101.59
MLA *(TP2, DP4)*	131K/4K	0	4/1280	203.04	32.03	28.64	37.50
GLA-8 *(TP8)*	131K/4K	0.125	4/1280	89.57	7.58	25.45	100.68
MLA *(TP2, DP4)*	131K/4K	0.125	4/1280	233.69	38.54	28.66	37.20
GLA-8 *(TP8)*	32K/4K	0.125	4/1280	55.97	1.14	22.32	165.78
MLA *(TP2, DP4)*	32K/4K	0.125	4/1280	73.82	3.29	25.73	125.31

Table 33. With a *random ratio of 0*, each request chooses its prefill and decode lengths uniformly from a single token up to the maximum lengths. With a *random ratio of 0.125*, the lengths are sampled uniformly, but now the range starts at 12.5% of the maximum specified length. GLA-8 with pure TP has higher throughput and lower median end-to-end latency than the hybrid TP + DP MLA configuration across both long and moderate context settings.

Method	**Prefill/Decode length**	**Rand. ratio**	**Max conc. /#Prompts**	**Mean E2E Latency** (s)	**Mean TTFT** (s)	**Mean ITL** (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	131K/4K	0	4/1280	80.93	7.78	35.54	101.59
MLA *(TP2, DP4)*	131K/4K	0	4/1280	219.43	41.82	86.31	37.50
GLA-8 *(TP8)*	131K/4K	0.125	4/1280	91.73	8.72	35.93	100.68
MLA *(TP2, DP4)*	131K/4K	0.125	4/1280	248.35	46.89	87.21	37.20
GLA-8 *(TP8)*	32K/4K	0.125	4/1280	55.66	1.19	23.59	165.78
MLA *(TP2, DP4)*	32K/4K	0.125	4/1280	73.65	3.79	30.26	125.31

Table 34. With a random ratio of 0, each request draws its prefill and decode lengths uniformly from one token up to the maximum lengths. With a random ratio of 0.125, the range begins at 12.5% of the maximum length. Across both long and moderate context settings, GLA-8 in pure tensor parallel form sustains higher throughput and lower mean end-to-end latency than MLA that combines tensor and data parallelism.

Method	**Prefill/Decode length**	**Rand. ratio**	**Max conc. /#Prompts**	**p99 E2E Latency** (s)	**p99 TTFT** (s)	**p95 ITL** (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	131K/4K	0	4/1280	175.47	19.91	26.78	101.59
MLA *(TP2, DP4)*	131K/4K	0	4/1280	566.48	117.44	30.80	37.50
GLA-8 *(TP8)*	131K/4K	0.125	4/1280	182.06	19.98	26.98	100.68
MLA *(TP2, DP4)*	131K/4K	0.125	4/1280	572.05	119.69	30.77	37.20
GLA-8 *(TP8)*	32K/4K	0.125	4/1280	99.08	2.49	23.49	125.31
MLA *(TP2, DP4)*	32K/4K	0.125	4/1280	135.87	8.61	27.48	165.78

Table 35. For ninety-ninth percentile values of latency, TTFT, and ITL (lower is better), GLA-8 with pure tensor parallel remains faster for the extreme long-context workload, while MLA with hybrid parallelism shows higher output throughput in the moderate context run.

B.5.4 Latency Sensitive Workloads

In latency-sensitive workloads, the predominant objective is to minimize end-to-end response time, particularly time to first token, to meet strict service level objectives rather than to maximize aggregate throughput. Because a larger batch can increase the queueing and prefill delay, latency-sensitive serving keeps the batch size very small, at the expense of throughput to deliver faster responses. In Table 36, GLA-8 with pure TP at eight degrees manages to reduce latency by x2 and cut the time to first token by almost x4 relative to MLA with a hybrid of TP and DP, where it is necessary to mitigate the duplication of the KV cache.

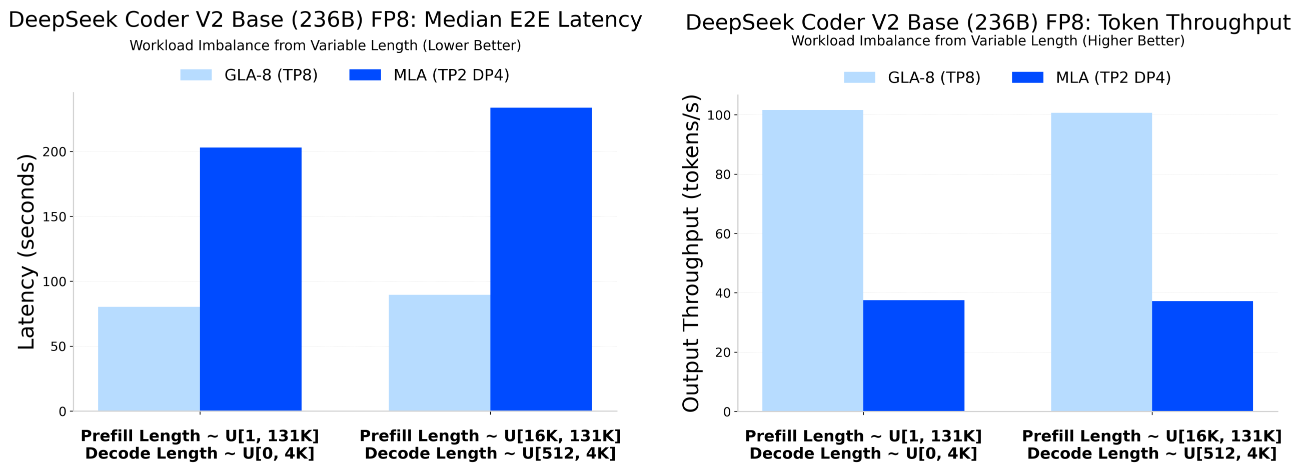


Figure 9. Median end-to-end latency (left), lower is better, and output throughput (right), higher is better, where the sequence length can vary and it is sampled from a uniform distribution. GLA using pure TP outperforms MLA with hybrid TP and DP.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Median** E2E Latency (s)	**Median** TTFT (s)	**Median** ITL (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	64K/256	3/1280	24.60	12.96	24.54	31.17
MLA *(TP8, DP4)*	64K/256	3/1280	54.25	46.76	28.14	14.14

Table 36. Under latency-sensitive scenarios, GLA with only tensor parallelism outperforms MLA with a mix of TP and DP solely for attention for long context short decode scenarios by over 50% for both end-to-end median latency (lower is better) and output throughput (higher is better).

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Mean** E2E Latency (s)	**Mean** TTFT (s)	**Mean** ITL (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	64K/256	3/1280	24.62	12.76	46.51	31.17
MLA *(TP2, DP4)*	64K/256	3/1280	54.26	46.47	30.55	14.14

Table 37. Mean service-level results for GLA and MLA. GLA shows lower latency (lower is better) than MLA under various metrics.

B.5.5 Decode Heavy Workloads

In decode-heavy workloads, the generated continuation is so long that the sequential decode phase dominates wall-clock time, resulting in latency and memory bandwidth for the KV cache being the primary bottleneck. Since the model will be performing sequential decoding most of the time, batching offers minimal benefit. In Figure 10, where there is a short prefill of 256 and long decoding of up to 32K, with GLA-8 and MLA across eight-degree parallelism, GLA-8 can generate up to 2.5x higher throughput.

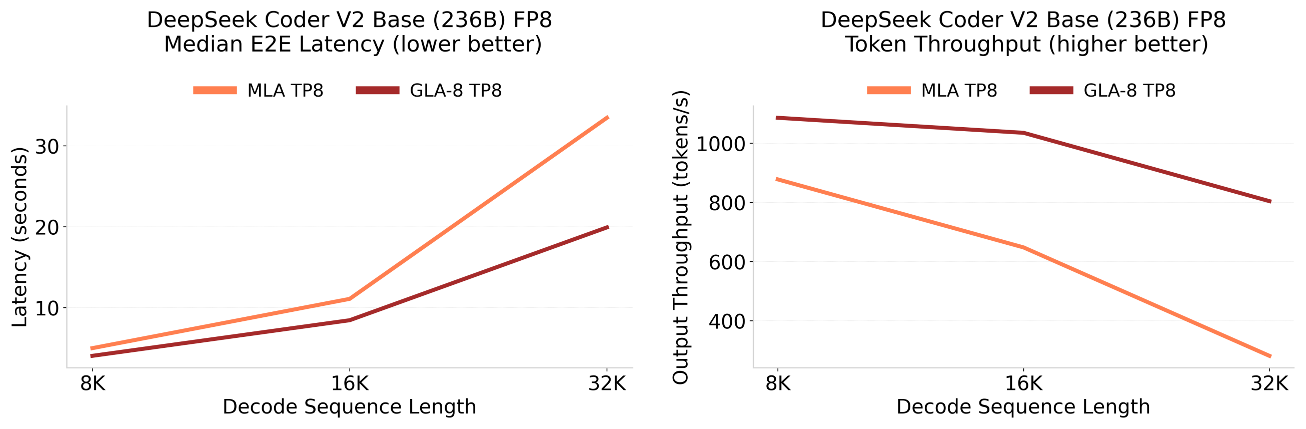


Figure 10. Demonstration of MLA and GLA for TP for degree of 8 on long decode tasks. With 256 number of prompts and 32 concurrent requests across various decode sequence lengths with fixed 2K prefill sequence length.

B.5.6 Small Context and Short Chat

Small Context describes inference requests in which both the prompt and the generated continuation are very short relative to the model context window. For example, a voice assistant answers a brief query in a single response. In Table 38, GLA-8 with eight latent heads in eight-degree parallelism has a lower latency relative to MLA with hybrid TP with degree 2 and DP with four data-parallel ranks since it is a single batch setting, the GPUs in the three out of four DP rank parallel groups remain idle, and GLA-8 has to fetch half the KV cache per layer; therefore, it outperforms MLA.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Median** E2E Latency (s)	**Median** TTFT (s)	**Median** ITL (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	256/128	1/1280	2.49	0.11	18.72	51.45
MLA *(TP2, DP4)*	256/128	1/1280	2.91	0.12	21.94	43.96

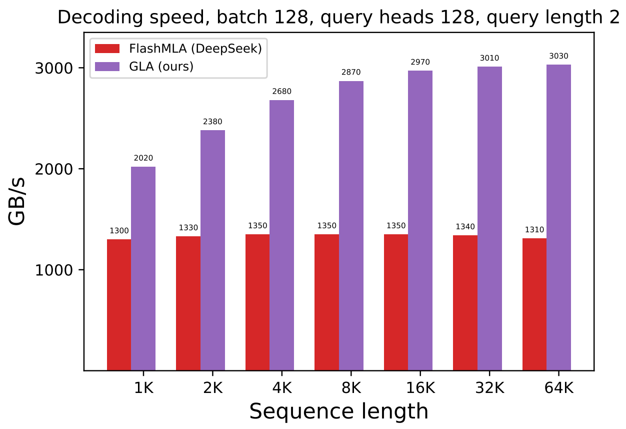
Table 38. Under short chat scenario where there is usually one concurrent request, GLA with only tensor parallelism has 17% higher throughput (higher is better) than MLA with mix of tensor parallelism and data parallelism

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Mean** E2E Latency (s)	**Mean** TTFT (s)	**Mean** ITL (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	256/128	1/1280	2.49	0.11	18.73	51.45
MLA *(TP2, DP4)*	256/128	1/1280	2.91	0.12	21.95	43.96

Table 39. Mean service-level results for GLA and MLA. GLA shows lower latency (lower is better) than MLA under various metrics

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Median** E2E Latency (s)	**Median** TTFT (s)	**Median** ITL (ms)	**Output Throughput** (token/s)
GLA-8 *(TP8)*	2K/2K	8/1280	47.18	0.86	22.54	346.92
MLA *(TP2, DP4)*	2K/2K	8/1280	56.35	0.82	27.04	290.91

Table 40. For moderate size prefill and decode sequence lengths with moderate number of concurrent requests, GLA with only tensor parallelism has roughly 19% higher throughput (higher is better) than MLA with mix of tensor parallelism and data parallelism.



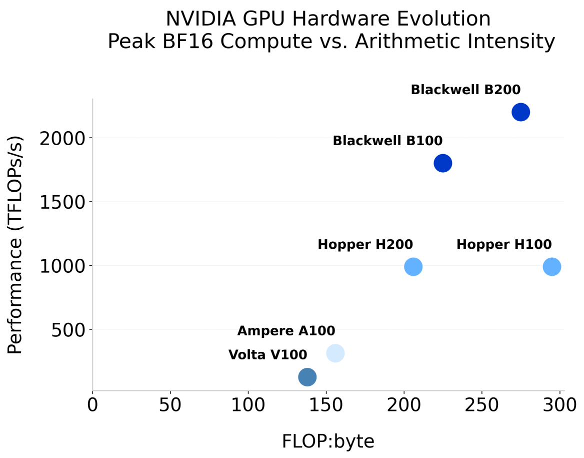


Figure 11. **Left**: Decoding speed of MLA and GLA on H100 80GB SMX5 GPU (theoretical max BF16 compute 989 TFLOPS/s and memory 3350 GB/s), for query length 2. At query length 2, GLA saturates compute (700 TFLOPS/s) and memory (3030 GB/s). **Right**: Peak BF16 theoretical peak FLOPs (TFLOPS/s) versus the arithmetic intensity for successive NVIDIA GPUs (Volta V100 with FP16). Performance computing has historically grown faster than bandwidth, with the H100 architecture (NVIDIA, 2022), which has the most drastic FLOPs-to-byte ratio increase relative to its predecessor. The decoding workload lies far left, so every device, even Blackwell B200, stays memory-bound and reaches only a few percent of its nominal TFLOP rate.

Method	**Prefill/Decode length**	**Max conc. /#Prompts**	**Mean** E2E Latency (s)	**Mean** TTFT (s)	**Mean** ITL (ms)	**Output Throughput** (token/s)
MLA *(TP2, DP4)*	2K/2K	8/1280	56.37	0.81	27.12	290.91
GLA-8 *(TP8)*	2K/2K	8/1280	47.22	0.82	22.67	346.92

Table 41. Mean service-level results for GLA and MLA. GLA shows lower latency (lower is better) than MLA under various metrics

B.6 Kernel Execution Time

We benchmark the latency of the attention kernels in these two settings on H100 GPUs (ignoring communication overhead) in Tables 42 and 43. GLA with TP = 2 can be 1.3-1.5 times faster than MLA with DP in these settings.

Seqlen	**MLA (DP)**	**GLA (TP=2)**
2048	15.0 μs	16.1 μs
8192	20.8 μs	19.1 μs
32768	35.9 μs	27.6 μs
131072	81.0 μs	55.0 μs

Table 42. Attention kernel latency (μs) for MLA vs. GLA on two GPUs with `batch=1`

Seqlens in batch	**MLA (DP)**	**GLA (TP=2)**
$[1024]*15+[8192]$	23.8 μs	25.4 μs
$[1024]*15+[16384]$	29.8 μs	26.2 μs
$[1024]*15+[32768]$	41.1 μs	30.6 μs
$[1024]*15+[65536]$	56.0 μs	42.6 μs

Table 43. Attention kernel latency (μs) with 2 H100 GPUs (`8B` model), imbalanced workload