ICEBERG ATTENTION: SUBLINEAR LONG-CONTEXT DECODING WITH MIXTURE-OF-GAUSSIANS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive decoding over long contexts repeatedly scans the entire KV cache, though only a small fraction of cached tokens strongly influence each query. We introduce *Iceberg Attention*, a class of attention mechanisms that exploits this asymmetry by attending exactly to a small, query-dependent subset of the KV cache while analytically approximating the contribution of remaining tokens. Unlike sparse or retrieval-based attention, Iceberg Attention does not discard the residual cache; instead, it represents its aggregate contribution using a compact approximation. This formulation decouples the cost of exact attention from the context length and enables sublinear KV-cache access.

We present *Mixture-of-Gaussians (MoG) Attention*, an Iceberg Attention mechanism. MoG hierarchically partitions the keys of each attention head into $\mathcal{O}(\sqrt{N})$ clusters and $\mathcal{O}(\log N)$ subclusters per cluster, representing each group by a Gaussian distribution. During decoding, the query is scored against these compact representations to identify the most relevant clusters and subclusters, which are attended exactly. The remaining cache is approximated from the means and variances of its Gaussian representations. MoG reduces per-token attention from $\mathcal{O}(N)$ to $\mathcal{O}(\sqrt{N}\log N)$ without retraining or fine-tuning the model.

We implement MoG as a drop-in replacement for attention in vLLM with a custom CUDA implementation that substantially reduces KV-cache data movement. At a context length of 1M tokens, MoG decoding is 27$\times$ faster than vLLM with FlashAttention and 2$\times$ faster than RetroInfer, a state-of-the-art sparse attention system, on an Nvidia H100 GPU. Across eleven long-context benchmarks using Qwen3-4B, Olmo3-7B, and Qwen3-14B at 8K–128K tokens, MoG matches exact attention while reading less than 5.3% of the KV cache. Ablations on real decode queries show the residual lowers the attention-output error of renormalized selection by about a quarter. These results demonstrate that long-context decoding can avoid reading most of the KV cache without modifying or retraining the underlying model.

1 INTRODUCTION

Long-context decoding is bound by reading the KV cache rather than by the computation of the model. Large language models are now used with contexts of hundreds of thousands to millions of tokens, over code repositories, long documents, extended conversations, and agentic workflows (Gemini Team et al., 2024; Meta AI, 2025; Yang et al., 2025b; Jimenez et al., 2024; Sun et al., 2025b). Every decode step reads every cached key and value, so even with IO-aware kernels (Dao et al., 2022; 2023; Zadouri et al., 2026) the step is bandwidth-bound and its cost grows linearly with the context. For a 4B-parameter model with eight key–value heads and 36 layers, the cache occupies 19 GB at 128k tokens and 147 GB at one million; once it exceeds GPU memory the read must also cross the CPU–GPU interconnect, whose 32–64 GB/s is one to two orders of magnitude below the 0.7–3.9 TB/s of GPU memory (NVIDIA Corporation, 2022a;b), and a decode step takes seconds (Sheng et al., 2023; Lee et al., 2024).

Existing methods reduce the read by eviction or retrieval (see Section 2), and both rest on the *sparsity hypothesis*: for a given query, a few tokens carry the attention mass and the rest can be treated as if it carried none. Eviction acts on it permanently, dropping entries before the query that needs them arrives; retrieval keeps the full cache and selects a query-dependent subset, but normalizes the softmax over that subset alone. The first half of the hypothesis is right: a few tokens dominate any

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

query. The second half fails at long context: each unselected token carries little mass, but there are many of them, and with a fixed budget their share of the cache grows with N while their aggregate mass does not shrink. Published budget sweeps make this concrete: at 16k tokens the top 20% of keys capture only 70–80% of the attention mass (Chen et al., 2025b), matching dense accuracy on LongBench needs 10–20% of the cache read (Tang et al., 2024b), reaching 90% of the mass takes $10\times$ the tokens of reaching 50% (Zhu et al., 2025), and systems that retrieve under 2% recover about 95% of dense performance rather than all of it (Chen et al., 2025a; Synk et al., 2025), consistent with attention mass dispersing over uninformative tokens as N grows (Vasylenko et al., 2026).

We introduce **Iceberg Attention**, a class of attention mechanisms built instead on the *iceberg hypothesis*: attention mass is concentrated on a few tokens but not negligible elsewhere. A few tokens matter individually; the rest matter in aggregate. An Iceberg mechanism computes attention exactly over a small, query-dependent *tip* of the cache and analytically over the *submerged* remainder, which is summarized rather than discarded, so the output is a correctly normalized attention over the whole cache in which only the tip is read. Retrieval attention is the member of the class whose residual estimate is zero; members differ in how the tip is chosen and how the remainder is summarized.

Mixture-of-Gaussians (MoG) Attention, our instance of the class, answers both questions with one closed-form expression. The keys of an attention head form dense, well-separated clusters (Roy et al., 2021; Hooper et al., 2024a; Liu et al., 2024c; Zhang et al., 2024), and attention depends on the cache only through the sums $\sum_j e^{q\cdot k_j}$ and $\sum_j e^{q\cdot k_j} v_j$. When the keys of a cluster are modeled as draws from a Gaussian, the first sum is the cluster's moment-generating function at q: a closed form in the cluster's mean, variance and size that estimates the attention mass the query assigns to the cluster without reading a key of it. MoG uses it to rank clusters, so that the query itself decides which slices of the cache form the tip, and to weight every cluster left unread, so that the remainder enters the softmax as pseudo-tokens. With $\mathcal{O}(\sqrt{N})$ clusters of $\mathcal{O}(\log N)$ subclusters per head, each stored as a contiguous slice, a decode step reads $\mathcal{O}(\sqrt{N}\log N)$ of the cache rather than $\mathcal{O}(N)$, with no retraining and two hyperparameters fixed throughout the paper.

We implement MoG as a drop-in attention backend in vLLM (Kwon et al., 2023) with custom CUDA kernels, and evaluate cost and fidelity separately, since the evidence for each lives in a different regime. The reduction in data movement is measured and grows without bound, from $1.9\times$ at 8k tokens to $49\times$ at ten million; the wall-clock win is decisive once the cache exceeds GPU memory, because the cluster statistics stay resident while only the selected slices cross the interconnect. At one million tokens a decode step takes 86 ms on an Nvidia A40 against 3.9 s for vLLM's FlashAttention decode kernel ($45\times$), and 34 ms on an H100 against 0.93 s ($27\times$), $2\times$ faster than RetroInfer (Chen et al., 2025a). Below that point the current implementation is roughly neutral on a single request: at 8k to 100k tokens the bytes saved are offset by a one-time clustering fit and per-layer host overhead (Appendix B.1), properties of the implementation rather than of the method. Per layer, MoG's attention output has a mean cosine similarity with exact attention of 0.98–0.99 at 8k–32k tokens and 0.94 at 128k (Appendix C). Accuracy is measured using eleven benchmarks spanning retrieval, code and reasoning with context length ranging 8k to 128k tokens on Qwen3-4B, Qwen3-14B (Yang et al., 2025a) and OLMo-3-7B (OLMo Team et al., 2025). While reading 1–5% of the cache, MoG matches exact attention on most benchmarks.

2 Background and Related Work

Auto-regressive transformers generate text one token at a time. During *prefill*, the prompt is processed in a single forward pass and, for every layer and attention head, the key and value vectors of all prompt tokens are stored in the KV cache. During *decoding*, each new token contributes one query vector $q \in \mathbb{R}^d$ per head, attends over the cached keys and values $k_i, v_i \in \mathbb{R}^d$, and appends its own key and value to the cache. The attention output of one head is

$$o \;=\; \frac{\sum_{j=1}^{N} e^{\,q\cdot k_j}\, v_j}{\sum_{j=1}^{N} e^{\,q\cdot k_j}}, \tag{1}$$

where the scaling (either by $1/\sqrt{d}$ or QK normalization) is absorbed into q. Both the numerator and the denominator are sums over every cached token, so a decode step reads the entire cache of every head in every layer. With grouped-query attention (Ainslie et al., 2023), which all models in this

paper use, several query heads share one set of keys and values, but each query head still evaluates Equation 1 over the full cache of its group. The cache grows linearly with N, and since a decode step performs only a few floating-point operations per byte read, its execution time is set by memory bandwidth rather than by compute.

Equation 1 can be written as partial pieces. For any subset S of the cached tokens, define the triple

$$m_S = \max_{j \in S} q \cdot k_j, \qquad \ell_S = \sum_{j \in S} e^{\,q \cdot k_j - m_S}, \qquad a_S = \sum_{j \in S} e^{\,q \cdot k_j - m_S}\, v_j, \tag{2}$$

where the running maximum m_S keeps the exponentials in range. Two triples over disjoint subsets combine into the triple of their union by rescaling both to the larger maximum, $m = \max(m_S, m_T)$, $\ell = \ell_S e^{m_S - m} + \ell_T e^{m_T - m}$, and likewise for a. The output is a/ℓ once every token has been accounted for. This online-softmax rule (Milakov & Gimelshein, 2018) is how flash-attention processes the cache in tiles (Dao et al., 2022). It is exact, and does not depend on how the cache is split. It is what makes an Iceberg mechanism possible: the tip, evaluated over real keys, and the submerged remainder, evaluated from cluster statistics, merge into one correctly normalized output.

Prior work reduces the linear cost of attending every cached token through one of three approaches. The first family changes the attention mechanism and thus needs the model to be retrained with the change: fixed sparsity patterns (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020), low-rank and kernel approximations (Wang et al., 2020; Katharopoulos et al., 2020; Choromanski et al., 2021; Xiong et al., 2021; Gerami & Duraiswami, 2025), content-based routing through hashing or clustering (Kitaev et al., 2020; Roy et al., 2021; Vyas et al., 2020), compressive and memory-augmented models (Rae et al., 2020; Wu et al., 2022; Munkhdalai et al., 2024; Chevalier et al., 2023), and architectures to shrink the cache (DeepSeek-AI, 2024; Gu & Dao, 2023). The clustering methods replace groups of keys by centroids, but inside a model trained to attend to centroids, and they do not account for the residual tokens. *None applies to an existing checkpoint without retraining*, which may degrade the model relative to the original, especially on limited data.

The second family shrinks the cache after prefill without modifying the model. Eviction policies keep a fixed token budget based on accumulated attention weights (Zhang et al., 2023; Liu et al., 2023; Ge et al., 2024; Adnan et al., 2024; Oren et al., 2024; Li et al., 2024), allocate it unevenly across layers or heads (Cai et al., 2024; Feng et al., 2024; Fu et al., 2024; Tang et al., 2024a), keep a sliding window with a few sink tokens (Xiao et al., 2024b; Han et al., 2023), or merge entries across layers (Liu et al., 2024a). They bound memory and time at the cost of permanent information loss: entries are discarded independently of any query, or on the basis of early ones, and cannot be recovered when a later query needs them, which degrades exactly the tasks that make long contexts valuable; the need to protect "retrieval heads" (Wu et al., 2024) is a symptom.

The third family, which this paper belongs to, keeps the full cache and decides at every decoding step which subset of it to read. To identify this subset, prior methods operate at various granularities. Some select contiguous blocks summarized by extreme or mean keys (Tang et al., 2024b; Xiao et al., 2024a; Lu et al., 2024; Yang et al., 2024), while others rely on low-rank or quantized key sketches (Ribar et al., 2024; Singhania et al., 2024; Zhang et al., 2024), post-prefill clusters (Hooper et al., 2024a; Liu et al., 2024c), or approximate nearest neighbors identified via vector search and locality-sensitive hashing (Liu et al., 2024b; Chen et al., 2025b). Systems built on these selections route tokens between GPU and host (Sun et al., 2025a; Chen et al., 2025a; Lee et al., 2024), and similar ideas accelerate prefill (Jiang et al., 2024; Gao et al., 2024). These methods discard nothing permanently. They differ in how the subset is found, and most of them normalize the softmax over the selected subset alone, the sparsity hypothesis of Section 1 applied at every decode step. The closest to MoG are Squeezed Attention (Hooper et al., 2024a) and ClusterKV (Liu et al., 2024c), which also cluster each head's keys after prefill and select clusters by centroid; MoG keeps that selection and adds a closed-form estimate of the attention mass of every cluster it does not read (Section 3), so the softmax is normalized over the whole cache.

Cache quantization (Hooper et al., 2024b; Liu et al., 2024d; Zandieh et al., 2026) and offloading (Sheng et al., 2023; Lee et al., 2024) are orthogonal: neither changes how much of the cache a step must touch, and both apply to the statistics and slices MoG reads in the same way as they apply to exact attention.

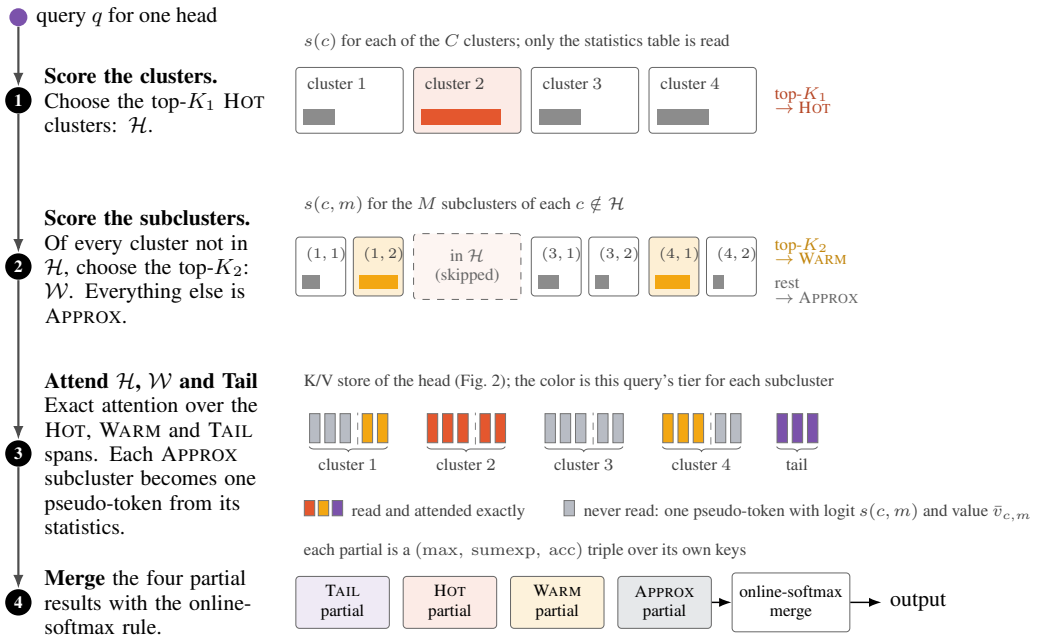


Figure 1: **Overview of one decode step of MoG for an attention head.** Stages 1 and 2 read only the mean, variance and count of the clusters and subclusters to score them and decide their tier for this query (bars are scores and the highlighted boxes are the selected ones). Stage 3 reads only the selected clusters (HOT), subclusters (WARM) and generated tokens during inference (TAIL). These spans are attended exactly, and the remaining subclusters each enter the softmax as a single pseudo-token without its keys or values being read. Stage 4 merges the four partial results into one output. In this example, there are two subclusters per cluster and (i, j) is subcluster j of cluster i.

3 METHOD

MoG replaces exact attention at every decoding step with exact attention over a small, query-dependent tip of the N cached tokens plus an analytic summary of the submerged remainder. After prefill, the keys of each head are partitioned into clusters and subclusters and a diagonal Gaussian is fitted to each group (Section 3.1); we keep only the diagonal of the covariance because the measured correlation between dimensions is weak (Appendix D). Each decode step evaluates four tiers and merges them with the online-softmax rule of Section 2 (Fig. 1). Three are exact: TAIL, over the tokens generated since the fit; HOT, over the members of the top-K_1 scoring clusters; and WARM, over the top-K_2 scoring subclusters of the clusters HOT did not select. The fourth, APPROX, is the submerged part: every remaining subcluster enters the softmax as one pseudo-token weighted from its Gaussian statistics. The merge is exact, so all approximation error lives in APPROX, and one closed-form score, Equation 3, both selects the exact tiers and weights the residual. The only hyperparameters are K_1 and K_2, fixed at 5 and 20 throughout.

3.1 PREFILL CLUSTERING

For each KV head, once the KV cache is computed during prefill, we partition the N cached keys into $C \approx \sqrt{N}$ clusters with k-means, then split each cluster into $M \approx \log_2 N$ subclusters of nearly equal size. Fig. 2 illustrates the clustering for one head. Tokens generated during inference belong to no cluster. They are kept in a small separate TAIL buffer and always attended to exactly.

For every cluster c and every subcluster (c, m) we store the statistics of a diagonal Gaussian fitted to its keys, i.e. the per-dimension mean μ and variance σ^2, together with the mean $\bar{v}$ of its values, which the APPROX tier needs later. The KV are rewritten in cluster-then-subcluster order, so that the members of any cluster/subcluster occupy contiguous spans of memory. This way, reading a subcluster is coalesced, doesn't require gather, and makes selection cheap at decode time.

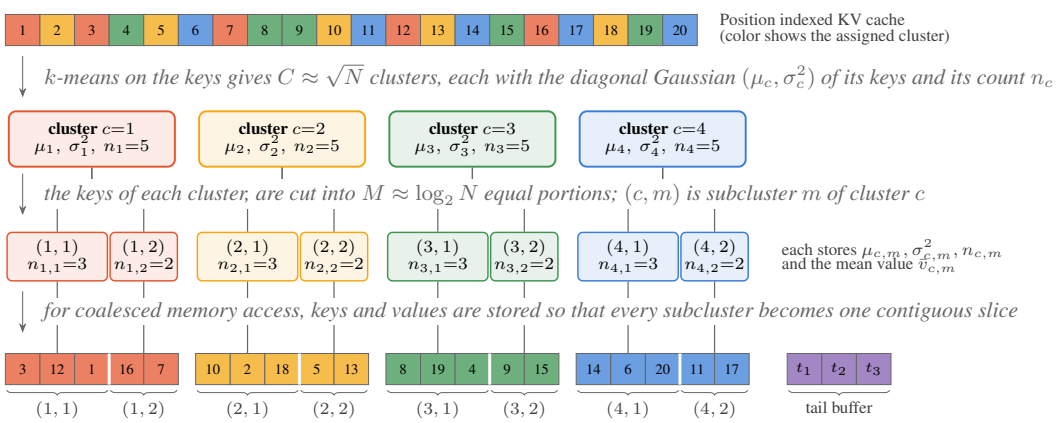


Figure 2: **Each head's cache clustering for MoG. Input: KV cache in position order. Output: KV cache in MoG clustering format.** The N=20 keys and values as they arrive, in position order, numbered by position (top row). k-means groups the keys into C=4 clusters (second row). The keys of each cluster are then cut into M=2 subclusters, here 3 and 2 (third row); (c, m) denotes sub-cluster m of cluster c. Every cluster and sub-cluster carries a diagonal Gaussian (μ, σ^2) of its keys and its member count n, and every sub-cluster also the mean $\bar{v}$ of its values. The keys and values of each subcluster are stored contiguously (bottom row) for efficient coalesced memory access. Tokens generated during inference belong to no cluster and are kept in a separate tail buffer.

3.2 Tiered Attention

Hot tier. At each decode step each of the C clusters of each head is scored against the query q:

$$s(c) \;=\; \underbrace{q \cdot \mu_c \;+\; \tfrac{1}{2} \sum_{i=1}^{d} q_i^2 \, \sigma_{c,i}^2}_{\text{per-key log-MGF}} \;+\; \underbrace{\log \max(n_c, \epsilon)}_{\text{count prior}}, \tag{3}$$

where n_c is the number of keys in cluster c, fixed at fit time, and ϵ avoids $\log 0$ for an empty cluster. The first two terms are the log moment-generating function of the logit $q \cdot k$ when a single key is drawn from $k \sim \mathcal{N}(\mu_c, \mathrm{diag}(\sigma_c^2))$. The count prior turns this per-key quantity into an estimate of the whole cluster's mass. Without this term, two clusters with identical per-key statistics but very different sizes would score identically. Computing $s(c)$ requires no real key to be read, and statistics μ_c, σ_c^2 and n_c are all recorded once during prefill. The K_1 clusters with the highest score are selected, and exact attention is computed over them, read directly from the contiguous store of Section 3.1. This produces the Hot triple of Equation 2.

Warm tier. The Warm tier adds a second, finer selection, restricted to the clusters Hot did not select. Every subcluster (c, m) of every such cluster is scored with Equation 3 evaluated on its own statistics $(\mu_{c,m}, \sigma^2_{c,m}, n_{c,m})$, giving $s(c, m)$, and the top-K_2 are chosen. For parallelism this runs in two stages: the candidates are split into a fixed number of partitions, each finds its local top-K_2, and a final top-K_2 is taken over the union of the local winners. Exact attention over the members of the chosen subclusters, read as contiguous slices, gives the Warm triple of Equation 2.

Approx tier. Every subcluster not covered by Hot or Warm is represented by an analytical approximation, avoiding the need to read its individual members. For a query q and a subcluster (c, m) with $n_{c,m}$ members $\{(k_j, v_j)\}$, its contribution to attention consists of two sums

$$Z_{c,m} \;=\; \sum_{j \in (c,m)} e^{\,q \cdot k_j}, \qquad A_{c,m} \;=\; \sum_{j \in (c,m)} e^{\,q \cdot k_j} \, v_j, \tag{4}$$

which enter the denominator and numerator of the attention output respectively. Computing them exactly requires reading every key and value in the subcluster; we approximate them instead. Because the members of a subcluster occupy a small region of key space around $\mu_{c,m}$, the discrete sum

in $Z_{c,m}$ can be approximated as an integral: treating the $n_{c,m}$ keys as samples from a continuous density $p(k)$ gives $Z_{c,m} \approx n_{c,m} \int e^{\,q\cdot k}\, p(k)\, dk$. The higher the sample count, the more accurate the approximation. For a Gaussian density with mean $\mu_{c,m}$ and diagonal covariance $\mathrm{diag}(\sigma^2_{c,m})$, this integral is the moment-generating function, and the estimate has the closed form

$$\hat{Z}_{c,m} \;=\; n_{c,m}\, \exp\Big(q\cdot\mu_{c,m} + \tfrac{1}{2}\sum_{i=1}^{d} q_i^2\, \sigma^2_{c,m,i}\Big) \;=\; \exp\big(s(c,m)\big), \tag{5}$$

The factor $n_{c,m}$ in front of the integral is the count prior $\log n_{c,m}$ of Equation 3. For the numerator, evaluating the same integral with the values attached gives $A_{c,m} \approx n_{c,m} \int e^{\,q\cdot k}\, v(k)\, p(k)\, dk$. By replacing the value of every member with the subcluster's mean $\bar{v}_{c,m}$, we obtain

$$\hat{A}_{c,m} \;=\; \hat{Z}_{c,m}\, \bar{v}_{c,m} \;=\; \exp\big(s(c,m)\big)\, \bar{v}_{c,m}. \tag{6}$$

With Eqs. 5 and 6, for a given query q, the entire subcluster behaves like a single pseudo-token with logit $s(c,m)$ and value $\bar{v}_{c,m}$. Evaluating this pseudo-token costs $O(d)$ and requires reading only the statistics $(\mu_{c,m}, \sigma^2_{c,m}, n_{c,m}, \bar{v}_{c,m})$ recorded during prefill. As a result, no key or value of an APPROX subcluster is read during inference.

The tier rests on two approximations: the Gaussian model of the key vectors, which determines $\hat{Z}_{c,m}$, and the replacement of every member's value by $\bar{v}_{c,m}$, which determines $\hat{A}_{c,m}$ and assumes that within a subcluster a value is not correlated with its logit $q \cdot k_j$. Both errors are scaled by $\hat{Z}_{c,m}$, so they are largest in the subclusters that receive the most mass, which are exactly the ones HOT and WARM remove from the residual and attend to exactly. Appendix G measures what each tier buys: renormalized HOT alone leaves a relative L_2 error of about 62%, since every unselected cluster is dropped from the normalizer; WARM reduces it to about 17% and APPROX to about 13%.

4 EXPERIMENTS

We separately evaluate MoG's efficiency and fidelity. Section 4.1 measures inference cost by measuring data transfer and per-step decode latency across three GPU types (8k to 10M tokens), with Appendix F isolating kernel speed. For downstream performance, Section 4.2 evaluates seven needle-in-a-haystack benchmarks against exact and retrieval baselines, while Section 4.3 assesses long-context reasoning. All experiments use a single, untuned configuration ($K_1 = 5$, $K_2 = 20$). Details of the experiment setups are presented in Appendix B. Per layer, MoG's attention output has a mean cosine similarity of 0.94–0.99 with that of exact attention (Appendix C).

4.1 INFERENCE COST

During inference, decoding speed is primarily memory-bandwidth bound. MoG mitigates this bottleneck across all levels of the memory hierarchy by restricting data transfer to $\mathcal{O}(\sqrt{N}\log N)$ bytes of cluster statistics and selected slices. Figs. 4a and 4b illustrate CPU–GPU and GPU off-chip to on-chip traffic across sequence lengths from 8k to 10M on A40, H100, and RTX 5000 GPUs. Per-step decode times (mean of 3–5 steps) are shown in Fig. 3, while Table 1 shows data movement, step time, and MoG speedup over exact attention and RetroInfer at 1M tokens. The results are for Qwen3-4B (36 layers of full attention) and the results for other models are in Appendix I.

Data Movement Off-chip to on-chip GPU data transfer (Fig. 4b) is hardware-agnostic: exact attention loads the complete KV cache, whereas MoG retrieves only the cluster statistics and relevant slices. Consequently, MoG's traffic reduction grows from $1.9\times$ at 8k tokens to $6.4\times$ at 128k, $18\times$ at 1M, and $49\times$ at 10M tokens. Host-to-GPU transfers (Fig. 4a) manifest once context allocations exhaust available VRAM—a threshold dictated by card capacity alongside baseline model and buffer footprints (230k tokens on RTX 5000, 300k on A40, and 700k on H100). Beyond these thresholds, exact attention streams all offloaded layers every step (48–108 GB per step at 1M tokens). MoG streams under 1GB by transferring only the missing slices not already cached in its slot buffer.

Decode Time While the KV cache fits in VRAM, step latency scales directly with GPU memory traffic. MoG outperforms exact attention starting at 8k tokens on the RTX 5000, 11k on the A40, and

GPU	GPU memory	GPU memory to SMs (MB per layer and step)		host to GPU (GB per step)		time per decode step (ms, 36 layers)			speedup over exact	speedup over Retro.
		exact	MoG	exact	MoG	exact	MoG	Retro.		
A40	41 GB	3805	214	96.6	0.84	3923	86.2	93.4	45×	1×
H100 NVL	88 GB	3805	214	48.3	0.28	927	34.1	68.4	27×	2.0×
RTX 5000	28 GB	3805	214	107.7	0.61	4355	68.6	138	64×	2.0×

Table 1: **Performance comparison of single-token decoding at a 1M-token context length using Qwen3-4B across three GPUs.** MoG accessing 0.6% of the key-value cache compared to exact attention. Host-interconnect traffic varies by GPU based on local cache residency. **TTFT is identical for both exact and MoG, as prefill execution and initial token sampling precede MoG clustering.** Speedup is the ratio of exact and RetroInfer (Retro.) to MoG decode latency.

approximately 110k on the H100, with performance gains widening as sequence length N grows. Once the cache spills into host memory, exact attention latency becomes bottlenecked by interconnect throughput, increasing by an order of magnitude over small increments in N. MoG maintains its sub-linear scaling trajectory. At 1M tokens, MoG achieves speedups of $64\times$ on the RTX 5000, $45\times$ on the A40, and $27\times$ on the H100. At 10M tokens, where exact attention approaches a 60-s step time on the A40, MoG decodes in sub-second latency. Compared to RetroInfer, which fetches index-selected pages from host memory, MoG incurs lower overhead at 1M tokens (34ms vs. 68ms on an H100); RetroInfer also fails to execute beyond 2M tokens.

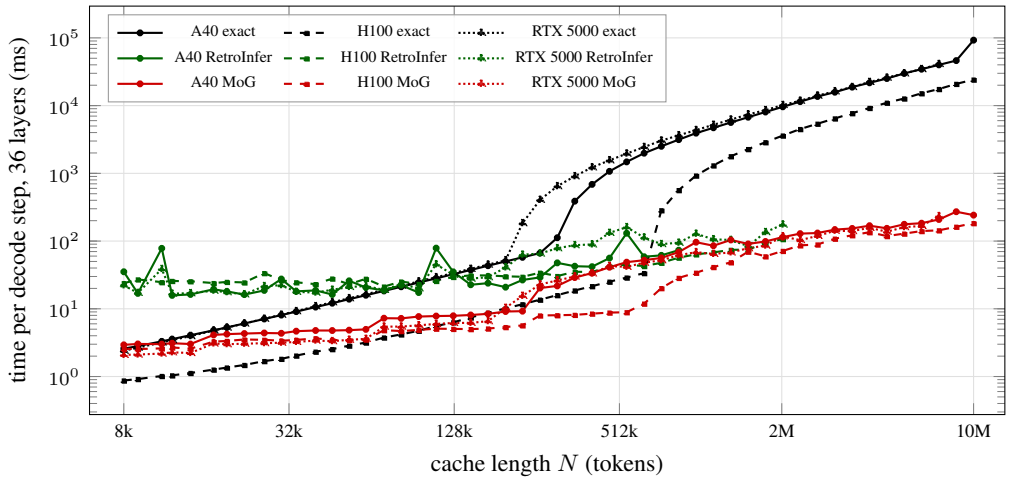


Figure 3: **Per-step inference decode time.** Comparing exact attention, MoG, and RetroInfer. Results are shown from 8k to 10M tokens using the maximum available GPU cache capacity on three GPUs. Within host GPU memory limits, performance gaps reflect differences in total bytes read from the cache; once capacity is exceeded, exact attention becomes limited by interconnect bandwidth, widening the gap by two orders of magnitude. RTX 5000 curves terminate at 7.5M tokens because a single cache layer (35 GB) exceeds host memory. Note that RetroInfer fails beyond 2M.

4.2 NEEDLE-IN-A-HAYSTACK BENCHMARKS

We evaluate seven benchmarks at three context lengths: single-needle retrieval (NIAH), multi-needle retrieval (NEEDLEBENCH), latent-association retrieval (NOLIMA), code retrieval (REPOQA), bug localization in a code stack (BICS), multi-hop question answering over news articles (MULTIHOP_RAG) and question answering over long conversations (LOCOMO), every method on identical prompts. Table 2 reports the mean score of exact attention, MoG and RetroInfer (Chen et al., 2025a), a retrieval-based baseline that keeps the full cache and attends to the nearest-neighbor keys found through a

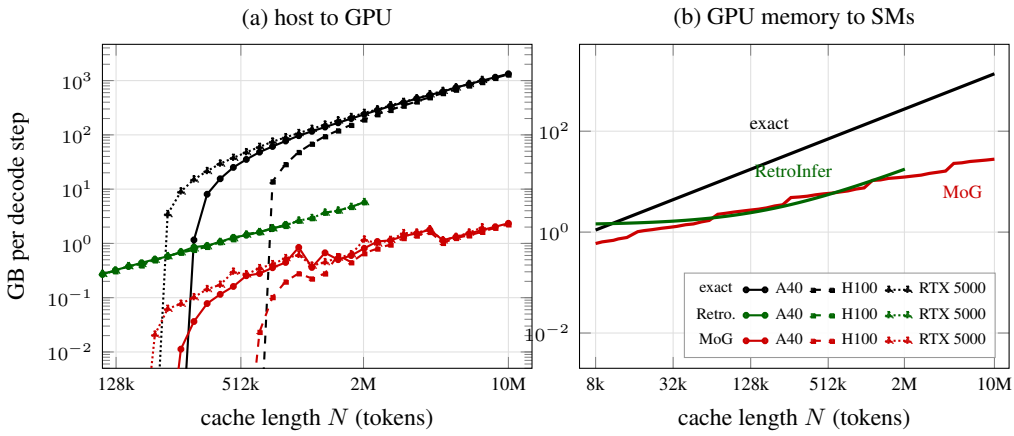


Figure 4: **Data movement per decode step.** **(a)** Host-to-GPU memory transfer per GPU. The curves begin where the KV cache exceeds available GPU memory, which varies by model. Colors indicate the attention mechanism, and line styles the GPU model. **(b)** Memory traffic generated by attention kernels within GPU VRAM. MoG transfers are derived from cluster shapes, while RetroInfer transfers are calculated from index shapes. Kernel-level traffic is GPU-agnostic. Exact attention transfers the entire cache across both memory tiers, whereas MoG limits transfers to the statistics and selected slices.

vector index, for Qwen3-14B. For the sake of space, the tables for Qwen3-4B-Instruct-2507 and OLMo-3-7B, and benchmark details are in Appendix B.2.

		8k			32k			128k		
Benchmark	n	exact	MoG	Retro.	exact	MoG	Retro.	exact	MoG	Retro.
NIAH	5	1.000	1.000	1.000	1.000	1.000	1.000	0.600	0.400	0.400
NEEDLEBENCH	20	1.000	1.000	1.000	1.000	1.000	1.000	0.883	0.950	0.883
NOLIMA	50	0.900	0.900	0.880	0.740	0.740	0.740	0.400	0.400	0.400
REPOQA	200	0.491	0.491	0.491	0.471	0.466	0.455	0.430	0.344	0.350
BICS	100	0.320	0.330	0.400	0.100	0.070	0.270	0.040	0.020	0.020
MULTIHOP_RAG	2255	0.859	0.858	0.858	0.854	0.852	0.853	0.820	0.829	0.820
LOCOMO	1540	0.527	0.525	0.526	0.500	0.490	0.500	0.420	0.410	0.420

Table 2: Needle-in-a-haystack benchmarks. Mean score of exact attention, MoG and RetroInfer (Retro.) on Qwen3-14B. n is the number samples per cell.

At 8k and 32k tokens MoG and exact attention are indistinguishable. On the two largest benchmarks, MULTIHOP_RAG with 2,255 questions and LOCOMO with 1,540, the two agree within 0.01 and paired bootstrap intervals over samples include zero, and on the retrieval benchmarks MoG recovers every needle that exact attention recovers, while reading under 5% of the cache. At 128k tokens the agreement largely persists. The exception is REPOQA, whose answers are entire functions that must be reproduced nearly verbatim, so that a single missed sub-cluster invalidates the sample; there MoG scores 0.344 against 0.430. NIAH and NEEDLEBENCH have 5 and 20 samples, so their 128k differences amount to one needle each. RetroInfer matches its own exact baseline at 8k and 32k on this model; at 128k its implementation failed on most cells (Appendix B.2).

4.3 LONG-CONTEXT REASONING BENCHMARKS

The needle benchmarks of Section 4.2 ask whether a method can still *find* a fact once the cache is large. A method that approximates attention could plausibly preserve retrieval while degrading the multi-step composition that long-context reasoning needs. This section tests that directly on four reasoning benchmarks. We use LONGREASON (Ling et al., 2025) and LOCOBENCH (Qiu et al., 2025), which can be run at controlled context lengths (8k, 32k and 128k, 100k for Qwen3-4B);

LONGBENCH-V2 (Bai et al., 2025) and AA-LCR (Artificial Analysis, 2025) supply real documents whose length is a property of the sample, and run at that length. For the sake of space, the benchmark construction, scoring rules, model details and an analysis of RetroInfer's failures are in Appendix B.3.

			8k			32k			128k (100k for 4B)		
Benchmark	Model	n	exact	MoG	Retro.	exact	MoG	Retro.	exact	MoG	Retro.
LONGREASON	Qwen3-4B-Instruct-2507	120	0.775	0.767	0.775	0.725	0.708	0.717	0.692	0.692	0.275
	OLMo-3-7B-Instruct	120	0.525	0.517	0.525	0.525	0.500	*coll.*	0.442	0.425	–
	Qwen3-14B	120	0.792	0.783	0.792	0.725	0.725	0.717	0.717	0.692	0.717
LOCOBENCH	Qwen3-4B-Instruct-2507	60	0.424	0.429	0.426	0.444	0.420	0.372	0.331	0.340	0.157
	OLMo-3-7B-Instruct	60	0.470	0.490	0.441	0.437	0.465	*coll.*	0.389	0.369	–
	Qwen3-14B	60	0.456	0.463	0.464	0.437	0.434	0.423	0.350	0.340	0.371

Table 3: Long-context reasoning at controlled length: mean score of exact attention, MoG ($K_1 = 5$, $K_2 = 20$) and RetroInfer (Retro.). n is the number of samples per cell. *coll.*: RetroInfer emitted token 0 for every generation in the cell, on every setting we tried; –: not run (RetroInfer's index build crashes on OLMo-3 at 128k, Appendix B.3).

Benchmark	Model	n	exact	MoG	Retro.
LONGBENCH-V2	Qwen3-4B-Instruct-2507	160	0.375	0.388	0.269
	OLMo-3-7B-Instruct	160	0.263	0.269	–
	Qwen3-14B	159–160	0.406	0.394	0.403
AA-LCR	Qwen3-4B-Instruct-2507	100	0.379	0.297	0.181
	OLMo-3-7B-Instruct	100	0.278	0.263	–
	Qwen3-14B	100	0.411	0.350	0.380

Table 4: Long-context reasoning at native document length. LONGBENCH-V2 and AA-LCR supply real documents whose length is a property of the sample, so these cells run at the benchmark's own length rather than a target budget. *coll.*: RetroInfer emitted token 0 for every generation in the cell; –: not run (Appendix B.3).

MoG preserves reasoning as it preserves retrieval. On LONGREASON and LOCOBENCH it is within 0.03 of exact attention in every cell, for all three models and at every length, and above it in several; on LONGBENCH-V2 the two are within 0.013. The one consistent gap is AA-LCR, whose questions combine facts spread across several long documents: MoG scores 0.297 against 0.379 on Qwen3-4B, 0.350 against 0.411 on Qwen3-14B and 0.263 against 0.278 on OLMo-3. RetroInfer matches exact attention at 8k, where it runs its dense fallback, and on Qwen3-4B loses accuracy with length as more of its generations collapse (Appendix J).

5 Conclusion and Next Steps

We presented Iceberg Attention, and Mixture-of-Gaussians (MoG) attention as an instance: a training-free, decode-time method that slashes data movement from $\mathcal{O}(N)$ to $\mathcal{O}(\sqrt{N}\log N)$ by reading only a fraction of the KV cache. By approximating clustered key weights with a closed-form Gaussian integral, MoG dynamically ranks cache slices while supplying an analytic residual logit for unread tokens. This residual lowers the relative L_2 error of renormalized cluster selection by about a quarter (Appendix G), providing safe and accurate decoding without needing to renormalize over partial cache selections.

The performance gains are largest where long-context inference hits memory bandwidth bottlenecks. At 1 million tokens, MoG cuts per-step decode latency on an A40 GPU from 3.9 s to 86 ms while matching exact attention on most benchmarks at 8k–128k tokens. MoG shows that interactive-speed generation on ultra-long contexts is possible precisely because most of the cache never needs to be read. A natural next step would be to replace the Gaussian distribution with von Mises–Fisher, which in theory better matches the unit-norm keys that QK-normalization produces, although its moment-generating function has no clean closed form, so the residual would need a new approximation method or a lookup table.

6 AI USE STATEMENT

Generative AI technologies were utilized in the preparation of this work to enhance presentation clarity and support technical implementation. Specifically, Large Language Models (LLMs) were employed to refine the manuscript's grammar, polish prose style, and ensure adherence to academic writing standards. Additionally, AI-assisted coding tools were utilized to facilitate parts of the software implementation and streamline the execution of benchmarking scripts. All core scientific methodologies, experimental designs, data analyses, and interpretations remain entirely the original work of the authors. The authors have thoroughly reviewed, verified, and take full responsibility for all content and codebase outputs presented in this paper.

REFERENCES

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J. Nair, Ilya Soloveychik, and Purushotham Kamath. Keyformer: KV cache reduction through key tokens selection for efficient generative inference. In *Proceedings of Machine Learning and Systems*, 2024. arXiv:2403.09054.

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. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2023. arXiv:2305.13245.

Artificial Analysis. Artificial analysis long context reasoning benchmark (LCR). Artificial Analysis, Inc., 2025.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jing Liu, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, et al. RetroInfer: A vector storage engine for scalable long-context LLM inference. *arXiv preprint arXiv:2505.02922*, 2025a.

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. In *International Conference on Learning Representations*, 2025b. arXiv:2410.16179.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2023. arXiv:2305.14788.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with Performers. In *International Conference on Learning Representations*, 2021. arXiv:2009.14794.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022. arXiv:2205.14135.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-Decoding for long-context inference. https://pytorch.org/blog/flash-decoding/, 2023.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. *arXiv preprint arXiv:2407.11550*, 2024.

Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. Not all heads matter: A head-level KV cache compression method with integrated retrieval and reasoning. *arXiv preprint arXiv:2410.19258*, 2024.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. SeerAttention: Learning intrinsic sparse attention in your LLMs. *arXiv preprint arXiv:2410.13276*, 2024.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *International Conference on Learning Representations*, 2024. arXiv:2310.01801.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Armin Gerami and Ramani Duraiswami. Transformer based linear attention with optimized GPU kernel implementation. *arXiv preprint arXiv:2510.21956*, 2025.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. LM-Infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*, 2023.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, June Paik, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length LLM inference. *arXiv preprint arXiv:2411.09688*, 2024a.

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. In *Advances in Neural Information Processing Systems*, 2024b. arXiv:2401.18079.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2407.02490.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, 2020. arXiv:2006.16236.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. arXiv:2001.04451.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, 2023. arXiv:2309.06180.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *USENIX Symposium on Operating Systems Design and Implementation*, 2024. arXiv:2406.19707.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2404.14469.

Zhan Ling, Kang Liu, Kai Yan, Yifan Yang, Weijian Lin, Ting-Han Fan, Lingfeng Shen, Zhengyin Du, and Jiecao Chen. LongReason: A synthetic long-context reasoning benchmark via context expansion. *arXiv preprint arXiv:2501.15089*, 2025.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. MiniCache: KV cache compression in depth dimension for large language models. *arXiv preprint arXiv:2405.14366*, 2024a.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. RetrievalAttention: Accelerating long-context LLM inference via vector retrieval. *arXiv preprint arXiv:2409.10516*, 2024b.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. ClusterKV: Manipulating LLM KV cache in semantic space for recallable compression. *arXiv preprint arXiv:2412.03213*, 2024c.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2305.17118.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning*, 2024d. arXiv:2402.02750.

Yi Lu, Xin Zhou, Wei He, Jun Zhao, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. LongHeads: Multi-head attention is secretly a long context processor. *arXiv preprint arXiv:2402.10685*, 2024.

Meta AI. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. `https://ai.meta.com/blog/llama-4-multimodal-intelligence/`, 2025.

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

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with Infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.

NVIDIA Corporation. NVIDIA A40 datasheet. `https://images.nvidia.com/content/Solutions/data-center/a40/nvidia-a40-datasheet.pdf`, 2022a.

NVIDIA Corporation. NVIDIA H100 tensor core GPU architecture whitepaper. Technical report, NVIDIA, 2022b. Version 1.01.

OLMo Team, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. OLMo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. *arXiv preprint arXiv:2401.06104*, 2024.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *International Conference on Learning Representations*, 2024. arXiv:2309.00071.

Jielin Qiu, Zuxin Liu, Zhiwei Liu, Rithesh Murthy, Jianguo Zhang, Haolin Chen, Shiyu Wang, Ming Zhu, Liangwei Yang, Juntao Tan, et al. LoCoBench: A benchmark for long-context large language models in complex software engineering. *arXiv preprint arXiv:2509.09614*, 2025.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. arXiv:1911.05507.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ attention: Bandwidth-efficient LLM inference. In *International Conference on Machine Learning*, 2024. arXiv:2312.04985.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning*, 2023. arXiv:2303.06865.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *Advances in Neural Information Processing Systems*, 37: 16692–16723, 2024.

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. In *International Conference on Machine Learning*, 2025a. arXiv:2410.21465.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon LLM agent via context-folding. *arXiv preprint arXiv:2510.11967*, 2025b.

Ryan Synk, Monte Hoover, John Kirchenbauer, Neel Jain, Alex Stein, Manli Shu, Josue Melendez Sanchez, Ramani Duraiswami, and Tom Goldstein. Exploiting sparsity for long context inference: Million token contexts on commodity GPUs. *arXiv preprint arXiv:2502.06766*, 2025.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. RazorAttention: Efficient KV cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024a.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *International Conference on Machine Learning*, 2024b. arXiv:2406.10774.

Pavlo Vasylenko, Hugo Pitorro, André F. T. Martins, and Marcos Treviso. Long-context generalization with sparse attention. In *International Conference on Learning Representations*, 2026.

Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. Fast transformers with clustered attention. In *Advances in Neural Information Processing Systems*, 2020. arXiv:2007.04825.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. *arXiv preprint arXiv:2404.15574*, 2024.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations*, 2022. arXiv:2203.08913.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *Advances in Neural Information Processing Systems*, 2024a. arXiv:2402.04617.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024b. arXiv:2309.17453.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A Nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. arXiv:2102.03902.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2.5-1M technical report. *arXiv preprint arXiv:2501.15383*, 2025b.

Shuo Yang, Ying Sheng, Joseph E. Gonzalez, Ion Stoica, and Lianmin Zheng. Post-training sparse attention with double sparsity. *arXiv preprint arXiv:2408.07092*, 2024.

Ted Zadouri, Markus Hoehnerbach, Jay Shah, Timmy Liu, Vijay Thakkar, and Tri Dao. FlashAttention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling. *Proceedings of Machine Learning and Systems*, 8:912–926, 2026. arXiv:2603.05451.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, 2020. arXiv:2007.14062.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. TurboQuant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, volume 2026, pp. 56418–56439, 2026.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. PQCache: Product quantization-based KVCache for long context LLM inference. *arXiv preprint arXiv:2407.12820*, 2024.

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. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2306.14048.

Kan Zhu, Tian Tang, Qinyu Xu, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context LLMs. *arXiv preprint arXiv:2502.12216*, 2025.

A CUDA IMPLEMENTATION

While the Method section defines the mathematical formulation of MoG at each decode step, this section details the systems-level engineering required for efficient execution on contemporary GPU hardware. Memory-bound attention mechanisms present three key design challenges during decoding: (i) maximizing memory bandwidth utilization relative to the theoretical baseline established by tier-specific attention coverage; (ii) optimizing GPU compute utilization across a four-tier pipeline by minimizing redundant computation, register spilling, and warp divergence; and (iii) ensuring the entire per-layer, per-request decode step can be captured and executed as a single CUDA graph despite variable-length inputs.

A.1 PHYSICAL MEMORY LAYOUT

During the prefill phase, key and value tensors are physically restructured into contiguous memory segments ordered by cluster and sub-cluster indices (c, m). This contiguous layout eliminates dynamic gather operations during decoding, reducing candidate retrieval to simple array slicing.

Furthermore, because each cluster is composed of M contiguous sub-cluster spans, the HOT and WARM tiers share a single underlying physical key-value store. The coverage for HOT at cluster c is equivalent to the concatenation of its constituent sub-clusters, which WARM accesses individually. Unifying these representations into a single nested buffer avoids duplicate data storage, reducing the key-value memory footprint by approximately 33% relative to maintaining separate cluster- and sub-cluster-level buffers without altering the data accessed by either tier.

A.2 COALESCED ATTENTION KERNELS

The real-attention kernels for both HOT and WARM decompose memory access into two primary phases: a scoring phase (query-key dot products) and a weighted aggregation phase (softmax-weighted value accumulation). Both phases enforce warp-coalesced memory reads to maximize memory throughput:

- **Scoring Phase:** A single warp is assigned to each target token. The 32 threads within a warp execute vectorized 8-byte (`uint2`) read instructions to load a 256-byte key row in a single contiguous transaction. Intra-warp partial dot products are subsequently aggregated using warp shuffle primitives. Assigning one token per thread introduces memory striding across 32 distinct rows per warp instruction, which degraded throughput by 8–30% on an NVIDIA A40 GPU due to cache capacity limits.
- **Weighted Aggregation Phase:** Value rows are loaded using four consecutive 16-byte (`uint4`) instructions distributed across 16 threads (8 dimensions per thread). This configuration enables concurrent processing of 8 tokens per execution pass while maintaining full warp-width transactions. Partial accumulators corresponding to the same output dimension are then reduced in shared memory.

All key and value tensors are stored in `bf16` format to minimize memory bandwidth overhead, while intermediate accumulation is performed in `fp32` precision to prevent numerical instability.

A.3 PARALLELIZED SELECTION MECHANICS

Selection within the HOT and WARM tiers requires retrieving the indices corresponding to the top-k scores from C (or $C \cdot M$) candidates. While a serial per-head argmax loop is sufficient for small values of k ($K_1 = 5$), its $O(k \cdot \text{candidates})$ complexity scales poorly. Profiling at larger scales ($K_1 = 50, K_2 = 200$) on an NVIDIA H100 with a 128k context length demonstrated that serial selection consumed 53% of the per-layer decode time (0.51 ms out of 0.97 ms) and restricted $K_2 \leq 384$ due to shared memory limits.

To support high-coverage retrieval at long contexts, selection is implemented via a parallel radix sort:

1. Candidate floating-point scores are transformed into order-preserving unsigned integers.

2. Four 8-bit digit passes construct histograms across 256 radix buckets, iteratively isolating the target bucket containing the k-th largest value.
3. A deterministic ballot-and-popcount compaction pass outputs keys exceeding the threshold, followed by the exact number of tied keys required to satisfy k.

To minimize thread serialization in early digits, lanes with matching digits are grouped using warp-level match operations (`__match_any_sync`), allowing a single lane per match group to execute the atomic increment. Additionally, candidate keys are staged in shared memory prior to processing when candidate memory requirements fall within allocation limits ($\leq$ 40 KB), avoiding repeated global memory accesses. This approach maintains $O(\text{candidates})$ complexity per digit pass, scales arbitrarily with k, and remains bit-deterministic across runs. The serial implementation is retained strictly as a fallback for small k workloads via a runtime threshold.

A.4 CUDA Graph Capture and Execution

Evaluating a multi-tier attention pipeline increases per-step kernel launch overhead, which can cause the model to become host-bound. To eliminate this overhead, each per-layer decode step is captured as a static CUDA graph and replayed across decoding steps. Graph execution requires static memory addresses, fixed launch configurations, and deterministic control flow. These constraints are addressed through the following mechanisms:

Fixed Launch Geometries for Dynamic Content. Sub-clusters are bounded by the near-equal split of Section 3.1, which guarantees a static maximum capacity per sub-cluster. The dynamic tail buffer (tokens generated post-prefill) is backed by a persistent allocation rounded to the nearest power of two. The kernel masks evaluation based on the active sequence length using a scalar execution parameter. Graph recaptures are triggered only when generation length exceeds the current power-of-two allocation bucket. Pre-allocating an initial capacity of 256 tokens avoids frequent graph re-captures during standard generation windows, eliminating host-side re-capture overheads ($\sim$ 220 ms per capture across 36 layers).

Stream Concurrency and Capture Safety. Independent execution paths (e.g., WARM and APPROX tiers) are dispatched on separate CUDA streams and synchronized using event-based fork/join operations, improving end-to-end decode performance by 17–20%. To satisfy CUDA graph capture constraints, all synchronization events explicitly disable timing flags (`cudaEventDisableTiming`). Furthermore, auxiliary streams (including the tail-attention stream) are explicitly synchronized with the primary capture stream via a wait-on-event call prior to initiating operations, ensuring explicit dependency tracking during graph construction.

Host-Side Kernel Fusion. To prevent host-side Python and memory allocator overheads from bottlenecking execution outside the CUDA graph, auxiliary prep-steps were fused into dedicated C++/CUDA kernels. Specifically, scratch tensor initialization and paged KV-cache extraction were fused into a single kernel launch (reducing overhead by $\sim$ 5.3 ms/step across 36 layers). Similarly, latest-token extraction, type conversion to `bf16`, and tail-buffer appending were integrated into a single fused operation, eliminating an additional $\sim$ 15 ms/step in host launch latency.

B Experiment Details and Additional Results

This appendix gives the setup of every experiment of Section 4, one subsection per subsection there, together with the measurements that the main text quotes without showing.

B.1 Inference Cost

Models and engine. Models are served with vLLM 0.8.5; the 100k and 128k cells use YaRN (Peng et al., 2024), and the exact-attention baseline is vLLM's flash-attention decode path in the same engine configuration. The longest context is 128k tokens for Qwen3-14B and 100k tokens for Qwen3-4B, the largest prompt for which the KV cache and the MoG store fit together on a single 80 GB GPU. Every experiment uses $K_1 = 5$ and $K_2 = 20$.

Setup. All three measurements use a full 36-layer decode step of the Qwen3-4B shape (32 query heads, 8 key/value heads, head size 128, bf16) with all of the GPU memory that the working buffers of the step do not need given to the cache: 41 GB on the A40, 88 GB on the H100 NVL and 28 GB on the RTX 5000 Ada. Layers whose data fit in that memory stay resident; the rest are kept in pinned host memory. For exact attention an offloaded layer is streamed chunk by chunk, double-buffered and overlapped with the attention over the previous chunk, and the chunks are merged with the online-softmax rule; the stream is bound by the measured 25 GB/s of the PCIe 4 links of the A40 and the RTX 5000 and 53 GB/s of the H100's PCIe 5 link. For MoG the per-cluster and per-sub-cluster statistics of all layers stay resident, selection runs on them, and only the HOT clusters and WARM sub-clusters missing from a per-layer LRU cache of GPU slots are fetched. The hit rate of that cache, 0.76–0.79, is calibrated to a trace of real selections from Qwen3-4B at 128k tokens, in which consecutive decode steps reuse 60% of the HOT clusters and 44% of the WARM sub-clusters. Offloading changes only where the store is read from: the two-phase kernel that selects on the statistics and attends over the fetched slices is bit-exact against the fused kernel used when the store is resident. Both kernels are replayed from CUDA graphs, and each point is the mean of five steps after two warm-up steps (three steps beyond two million tokens). The transfers, kernels and cache behavior are measured, but the workload is synthetic: the queries drift from step to step by an amount chosen so that the cache hit rate matches the one observed on the real trace, and the cache contents are random. The result is therefore a measurement of the mechanism under a calibrated workload rather than an end-to-end run of a model, which no engine can perform at these lengths. Timing variation between repeated runs of the sweep is below one percent (two independent runs of the exact baseline at one million tokens agree to 0.01%), so error bars would be smaller than the markers and are omitted. The sensitivity to the calibration is small: at one million tokens on the A40 the MoG step is 122 ms at a hit rate of 0.52 and 60 ms at 0.88, against 80 ms at the calibrated 0.77 (measured with a 30 GB cache), because the fetch volume at any of these rates is far below what the link can carry in a step.

Data movement. The two curves of Figure 4 that describe GPU memory traffic are the origin of everything else in this section. Exact attention loads the whole cache, $2NLhd$ bytes, at every step. MoG loads the statistics, which grow as $\sqrt{N}\log N$, and the selected slices, which grow as $\sqrt{N}$; at one million tokens the two amount to 3.8 GB and 214 MB per layer (Table 1). The ratio between the two grows without bound, from $1.9\times$ at 8k tokens to $6.4\times$ at 128k, $18\times$ at one million and $49\times$ at ten million, and the share of the cache that is read exactly falls below one percent beyond 300k tokens. This term does not depend on the GPU. The host-to-GPU curves do: the memory left for the cache after the model weights and the working buffers of the step differs between cards, so the cache spills to the host at a different length on each, about 230k tokens on the RTX 5000, 300k on the A40 and 700k on the H100, and the volume streamed thereafter differs accordingly. Beyond that point the curves show the same mechanism: exact attention must stream every offloaded layer at every step, 97 GB per step at one million tokens on the A40, 48 GB on the H100 and 108 GB on the RTX 5000, whereas MoG fetches only the slices that the LRU cache does not already hold, under 1 GB at the same length on any of the three, since its statistics never leave the GPU.

Time per decode step. Figure 3 shows the consequence. While the cache is resident, the step time of each method follows its GPU-memory traffic: MoG overtakes exact attention at 11k tokens on the A40, at 8k on the RTX 5000 and at about 110k on the H100, whose memory bandwidth keeps the exact kernel competitive over a wider range, and the gap then widens with every doubling of N. Once the cache exceeds GPU memory the exact step becomes a function of the interconnect alone and rises by an order of magnitude within a factor of two in N, while the MoG step continues on its previous trend. At one million tokens the A40 needs 3.9 s per step for exact attention and 86 ms for MoG, a factor of 45; the H100, with twice the memory and twice the link bandwidth, needs 0.93 s and 34 ms, a factor of 27; and the RTX 5000, whose memory holds the smallest share of the cache, needs 4.4 s and 69 ms, a factor of 64. At the next point of the grid (1.13M tokens) the three factors are 55, 32 and 81. The ten-million-token points, where the exact step approaches a minute on the A40, are where the difference between a read that grows linearly and one that grows with the square root of N becomes a difference in kind: a context of that size can be decoded at all only because most of it is never read.

Time to first token. MoG does not change the time to first token. The prompt is processed by the same prefill kernels in both configurations, the key/value cache is computed and written exactly as for exact attention, and the first output token is sampled from the logits of the last prompt position at the end of that pass, before MoG has done anything. The clustering is deferred to the first decode step, so it delays the second token, not the first. Measured through vLLM for Qwen3-4B on two A40s, the time to first token is 0.55 s, 3.5 s and 23 s for prompts of 8k, 32k and 100k tokens with either attention. MoG therefore trades nothing at the first token; its cost and its benefit both begin with the first decode step.

The one-time fit. What MoG adds is a one-time cost per request at the first decode step, when the keys of every layer are clustered, their statistics are computed, the store is rewritten in cluster order and the CUDA graph of each layer's decode step is captured. In the runs above the fit takes 1.0 s, 3.5 s and 11.5 s at 8k, 32k and 100k tokens, half of the prefill time at the longest length, and is the same without graph capture, so the clustering itself dominates. It runs fifteen k-means iterations after a five-iteration power method per head and layer, so its cost is $O(N\sqrt{N}\,d)$ per head, superlinear in N; it could be folded into prefill, where the keys are already in flight, and the present implementation runs it separately through a Python-driven loop over the layers. The break-even generation length, at which the per-step saving repays the fit, is about 5,000 tokens at 100k on this two-GPU configuration (11.5 s against a saving of 2.2 ms per step), and does not exist at 8k and 32k, where the step is not yet faster. In the same runs the steady-state step of the whole model was 17.4, 17.8 and 26.2 ms for exact attention and 20.1, 22.0 and 24.0 ms for MoG: with the model split over two GPUs each device reads only half of the cache, attention is a small share of the step at these lengths, and the roughly three milliseconds that the MoG decode path spends outside its kernels, in the per-layer replay of the captured graphs and the maintenance of the tail buffer, outweigh the bytes saved until the cache is large. Batch size adds a further caveat. The measurements above are at batch size one, which favors MoG: exact attention amortizes one read of the cache over all sequences of a batch that share it, whereas MoG's selected slices are per query. We do not report larger batches or a comparison with offloading systems such as FlexGen and InfiniGen (Sheng et al., 2023; Lee et al., 2024): those systems schedule where the cache lives and when it moves, and are orthogonal to what MoG changes, which is how much of the cache a step must touch. Like quantization of the cache, any such optimization that applies to exact attention applies unchanged to the statistics and slices MoG reads, and the two savings compose. Both the fit and this per-layer overhead are properties of the current implementation rather than of the method, and both are separate from the reduction in data movement that Figures 3 and 4 establish; we report them because a reader deploying MoG today will see them. The step times of Figure 3 and Table 1 are steady-state decode steps: they include neither prefill, which is identical for the two methods, nor the fit amortized over the generation. Amortizing the fit would move the MoG curves up by the fit time divided by the number of generated tokens, a quantity the break-even lengths above put in perspective, and amortizing prefill would not separate the methods, since at multi-million-token contexts its quadratic cost exceeds that of any realistic generation and an amortized curve would show the prefill of both and the decode of neither.

B.2 NEEDLE-IN-A-HAYSTACK BENCHMARKS

How the prompts are built. Every benchmark is run through one harness that builds a prompt of a requested length from the benchmark's own material and inserts what the question is about at a chosen depth. Lengths are counted with the tokenizer of the model under test rather than with the tiktoken or CodeLlama tokenizers of the upstream harnesses, so that "N tokens" is the length of the KV cache the model, and therefore MoG, actually sees; the published length grids of the upstream benchmarks differ slightly for this reason. The context is filled by concatenating units of the benchmark's corpus until the target length is reached and truncating at the token boundary; when the corpus is shorter than the target, the units are cycled, so a longer prompt is a stack of several copies of the same material, as in the original needle-in-a-haystack setup. The units are Paul Graham essays for NIAH and NEEDLEBENCH, sentence-shuffled book text for NOLIMA, clean line-numbered code snippets for BICS, the files of the target repository for REPOQA, news articles of the corpus that are not evidence for the query for MULTIHOP_RAG, and the other nine conversations of the dataset for LOCOMO, so that at 32k and 128k tokens a LOCOMO prompt consists of the target conversation stacked together with several unrelated conversations of the same form. The needle, one sentence for NIAH and NOLIMA, several facts at evenly spread depths for NEEDLEBENCH, one

buggy snippet for BICS, the target function for REPOQA, the full body of every evidence article for MULTIHOP_RAG and the whole target conversation for LOCOMO, is inserted at depths of 0, 25, 50, 75 and 100% of the context, and the question follows the context. Two benchmarks have a length floor: the evidence articles of MULTIHOP_RAG and the conversations of LOCOMO (12k–24k tokens each) are never truncated, so their "8k" column contains prompts of roughly that material's own length, and filler is added only for the longer columns. Decoding is greedy with 64 new tokens. The scorers are those of the upstream benchmarks where they need no external judge: substring or exact match for NIAH, NEEDLEBENCH, NOLIMA, BICS and MULTIHOP_RAG, sentence BLEU with a pass threshold of 0.8 for REPOQA, and token F1 for LOCOMO, with full credit when the normalized gold answer appears verbatim in the prediction; LOCOMO's adversarial category, whose expected answer is an abstention, is excluded because answer matching cannot score it. The sample counts n in the tables are the number of (needle, depth, repeat) instances per cell for the retrieval benchmarks, the first 200 tasks of REPOQA, 2,255 of the 2,556 MULTIHOP_RAG queries, and the 1,540 LOCOMO questions of the four scored categories.

For each cell of Tables 2, 5 and 6 the prompts are materialized once with the model's own tokenizer and scored under every attention mode, so that all methods are evaluated on identical inputs; the modes run in separate engine instances so that no state can be shared between them. The share of the cache MoG reads exactly is determined by N, K_1 and K_2 alone, 5.3%, 2.7% and 1.2% at 8k, 32k and 128k tokens, and is therefore the same for every benchmark and model at a given length. Tables 5 and 6 give the results for Qwen3-4B-Instruct-2507 and OLMo-3-7B.

		8k			32k			100k		
Benchmark	n	exact	MoG	Retro.	exact	MoG	Retro.	exact	MoG	Retro.
NIAH	5	1.000	1.000	1.000	1.000	1.000	1.000	0.600	0.400	0.000
NEEDLEBENCH	20	1.000	1.000	1.000	1.000	1.000	1.000	0.467	0.333	0.000
NOLIMA	50	0.420	0.440	0.400	0.160	0.160	0.180	0.180	0.200	0.100
REPOQA	200	0.444	0.445	0.438	0.271	0.270	0.272	0.159	0.127	0.105
BICS	100	0.200	0.250	0.340	0.020	0.010	0.110	0.000	0.000	0.000
MULTIHOP_RAG	2255	0.832	0.830	0.836	0.772	0.772	0.800	0.543	0.542	0.235
LOCOMO	1540	0.478	0.475	0.461	0.430	0.413	0.397	0.143	0.126	0.058

Table 5: Qwen3-4B-Instruct-2507: mean score of exact attention, MoG and RetroInfer (Retro.).

		8k			32k			128k		
Benchmark	n	exact	MoG	Retro.	exact	MoG	Retro.	exact	MoG	Retro.
NIAH	5	1.000	0.800	1.000	0.600	0.600	*coll.*	0.200	0.000	–
NEEDLEBENCH	20	0.833	0.833	0.867	0.700	0.533	*coll.*	0.533	0.233	–
NOLIMA	50	0.160	0.140	0.140	0.060	0.060	*coll.*	0.120	0.120	–
REPOQA	200	0.288	0.285	0.285	0.199	0.174	*coll.*	0.111	0.069	–
BICS	100	0.010	0.010	0.010	0.000	0.000	*coll.*	0.000	0.010	–
MULTIHOP_RAG	2255	0.811	0.810	0.801†	0.794	0.795	*coll.*	0.748	0.744	–
LOCOMO	1540	0.415	0.414	0.177†	0.366	0.364	*coll.*	0.266	0.258	–

Table 6: OLMo-3-7B-Instruct: mean score of exact attention, MoG ($K_1 = 5$, $K_2 = 20$) and RetroInfer (Retro.), with the recipe of the Qwen3 tables. The 128k cells run beyond OLMo-3's native 65,536-token window with its global layers' YaRN factor raised to reach a 148k window; at that length the model often summarizes the context instead of answering, under every attention mode. RetroInfer builds its retrieval index only for prompts of at least 16,452 tokens, so its 8k column is its dense runtime, except for LOCOMO, three quarters of whose 8k prompts exceed that length (the conversation itself is 12k–22k tokens), and 56 of the 2,255 MULTIHOP_RAG prompts. †RetroInfer produced collapsed generations (token 0 repeated) on part of the samples, which score zero. *coll.*: every generation collapsed (probes of 2–20 prompts per benchmark): RetroInfer's k-means leaves some heads with fewer non-empty clusters than it reads per decode step (Appendix B.3). –: not reported; RetroInfer's index build crashes on OLMo-3 at 128k.

At 8k and 32k tokens, MoG and exact attention are indistinguishable. On the two largest benchmarks, MULTIHOP_RAG with 2,255 questions and LOCOMO with 1,540, the two modes agree to within 0.003 at 8k and within 0.02 at 32k for both models; a paired bootstrap over samples puts the 95% interval of the difference at $[-0.005, +0.002]$ and $[-0.006, +0.006]$ for MULTIHOP_RAG at 8k and 32k on the 4B model, $[-0.006, +0.004]$ and $[-0.007, +0.004]$ on the 14B, and $[-0.008, +0.004]$ for LOCOMO at 8k on both, so none of these differences is distinguishable from zero. Decoding is greedy, so the only source of variation is the sample of prompts, and on the retrieval benchmarks MoG recovers every needle that exact attention recovers. In this regime MoG reads between 2.7% and 5.3% of the cache.

At the longest context the agreement largely persists, with two exceptions. REPOQA is the only needle benchmark on which MoG is consistently below exact attention at long context on both Qwen3 models (0.127 against 0.159 for the 4B model and 0.344 against 0.430 for the 14B model; paired bootstrap intervals $[-0.053, -0.013]$ and $[-0.118, -0.058]$, so the gap is real). Its answers are entire functions that must be reproduced nearly verbatim (sentence BLEU of at least 0.8) from a repository located anywhere in the prompt, so a single missed sub-cluster invalidates the whole sample. The second exception is a matter of sample size: NIAH and NEEDLEBENCH comprise 5 and 20 samples, so a difference of one or two samples shifts the score by 0.2 and 0.05 respectively. MoG misses one NIAH needle on each model and recovers one additional NEEDLEBENCH needle on the 14B model, which is consistent with sampling noise at this size. BICS is close to zero for exact attention itself at every length and therefore provides little discriminative signal.

RetroInfer runs in its authors' implementation, which serves the model through HuggingFace Transformers rather than vLLM, with the same prompts, decoding settings and YaRN policy as our runs; its exact-attention baseline in that framework differs from the vLLM one in the tables by at most 0.02 on every benchmark except BICS, whose hundred samples give 0.34 against 0.20 at 8k. Judged against its own baseline, RetroInfer matches exact attention at 8k on every benchmark and for both models, and at 32k trails it by 0.05 on LOCOMO and 0.02 on NOLIMA for the 4B model while matching it elsewhere; MoG shows no comparable loss at 32k. At 100k and 128k tokens RetroInfer's retrieval path fails on part of the samples through the two mechanisms of Appendix B.3; collapsed generations score zero, so its long-context cells understate it, and we do not draw a comparison at these lengths.

The long-context cells are also those in which the budget matters. At 100k–128k tokens the default setting reads less than 2% of the cache. The fact that the end-task scores are maintained as well as they are at this coverage is the contribution of the APPROX tier, which keeps the softmax normalized even when the exact tiers miss relevant keys.

B.3 LONG-CONTEXT REASONING BENCHMARKS (SECTION 4.3)

Benchmarks. LONGREASON (Ling et al., 2025) consists of 794 multiple-choice reasoning questions synthetically expanded across several context-length targets, with a distinct split provided per length. Because the underlying questions remain constant across splits, evaluating across the 8k, 32k, and 128k context tiers isolates the effect of sequence length while controlling for task difficulty—a property unique among the evaluated benchmarks. LOCOBENCH (Qiu et al., 2025) evaluates software engineering capabilities (e.g., code comprehension, architecture mapping, bug localization, and security analysis) over complete, multi-file repositories across ten programming languages. Because context sizes vary by repository, length-specific evaluation bins are constructed by selecting repositories whose assembled prompts closely match the target token length. LONGBENCH-V2 (Bai et al., 2025) (503 four-way multiple-choice questions across real-world documents in six domains) and AA-LCR (Artificial Analysis, 2025) (100 multi-document reasoning questions averaging $\approx$100k tokens) feature unmanipulated documents where context length is an intrinsic property of each instance rather than a controllable parameter. Accordingly, these two benchmarks are evaluated at their native sequence lengths, applying middle-truncation only when a sample exceeds the model's context window.

Scoring. LONGREASON and LONGBENCH-V2 are evaluated using standard multiple-choice accuracy based on the model's chosen letter option. Because AA-LCR and LOCOBENCH elicit free-form responses typically graded via LLM-as-a-judge, we implement deterministic scoring proxies to avoid judge-model dependencies. AA-LCR is scored by the recall of salient reference atoms (e.g.,

numerical figures, percentages, and named entities) present in the generated output. LOCOBENCH is scored by the fraction of published scenario evaluation criteria whose named symbols are generated. While these deterministic metrics serve as proxies—and their absolute scores are not directly comparable to published judge-based results—they provide an objective basis for comparing relative performance across different attention mechanisms under identical prompts and scoring functions.

Models and Baselines. In addition to the two Qwen3 variants, we evaluate OLMo-3-7B-Instruct (OLMo Team et al., 2025). OLMo-3 employs a 65,536-token native context window using standard multi-head attention (32 key/value heads), where three out of every four layers use a 4,096-token sliding window and the remaining quarter attend globally. Because MoG preserves exact attention on layers with local sliding windows—where standard windowing already caps cache growth—MoG selectively replaces full attention in only 8 of OLMo-3's 32 layers (specifically, those whose key/value cache expands with sequence length). The RetroInfer baseline is applied to the exact same subset of layers using a modified runtime that provides bounded key/value buffering for windowed attention. RetroInfer's runtime provides both CPU–GPU cooperative caching (as originally proposed) and a GPU-only cache mode. It constructs its retrieval index only for prompts exceeding 16,452 tokens (a 16,384-token segment threshold plus 68 static tokens); shorter contexts default to full dense attention, making its 8k evaluations a measure of its dense implementation rather than retrieval. Beyond this length, the reference RetroInfer implementation exhibits two critical failure modes:

1. **Empty cluster allocation:** The k-means algorithm occasionally produces non-empty clusters well below the fixed allocation queried during decoding. For example, at 100k tokens on Qwen3-4B, layer 1 yielded only 83 and 102 valid clusters across two key/value groups compared to the expected 1,560. On OLMo-3 at 32k tokens, a head yielded 7 valid clusters against a target of 500. Attending over these unpopulated clusters produces non-finite values, causing the model to generate repetitive token 0 sequences for the remainder of the generation. This failure rate increases with prompt length on Qwen3-4B (occurring in a few percent of runs at 32k and ≈66% at 100k), while affecting 100% of generations on OLMo-3 at 32k.

2. **Index construction crashes:** On the longest inputs, index construction triggers illegal memory accesses. This occurred across five of the 160 LONGBENCH-V2 prompts on Qwen3-4B (≈130k tokens each, across three retries) and universally on OLMo-3 at 128k.

As a result, RetroInfer results on OLMo-3 are omitted beyond 32k tokens. These failures stem specifically from the retrieval mechanism rather than the underlying dense kernel; when executing dense attention on the same inputs, RetroInfer achieves normal accuracy (e.g., 0.700 on Qwen3-4B at 100k context on LONGREASON, compared to 0.275 under retrieval where 62% of outputs collapsed, and 0.692 for both exact attention and MoG). We report RetroInfer performance wherever valid outputs are produced, denote fully collapsed evaluation cells with *coll.*, and detail cell-wise collapse rates in Appendix J. Exact attention and MoG never emit token 0; the at most 3% counted for them are repetition loops. Full benchmarks are provided in Tables 3 and 4 (Section 4.3).

C Per-layer Numerical Match with Exact

To measure how closely MoG reproduces exact attention independently of any benchmark, the production decode path is instrumented so that, at every layer and decode step, it also computes exact attention from the same keys and values and compares the two outputs per head. Table 7 reports the cosine similarity between the outputs, averaged over heads and layers, its median over layers and the mean relative L_2 error, for Qwen3-4B and Qwen3-14B at 8k, 32k and 128k tokens.

MoG's output is very close to that of exact attention. At 8k and 32k tokens the default budget reads 2.7–5.3% of the cache and reproduces exact attention with a mean cosine similarity of 0.98–0.99 for both models, with 1–4% of heads below 0.9. At 128k tokens it reads only 1.2% of the cache and the mean falls to 0.94.

Model	N	cosine similarity mean	cosine similarity median	rel. L_2 mean
Qwen3-4B-Instruct-2507	8k	0.992	0.999	0.055
Qwen3-4B-Instruct-2507	32k	0.984	0.994	0.107
Qwen3-4B-Instruct-2507	128k	0.939	0.960	0.339
Qwen3-14B	8k	0.993	0.996	0.069
Qwen3-14B	32k	0.983	0.992	0.121
Qwen3-14B	128k	0.938	0.958	0.266

Table 7: Fidelity of MoG's attention output to exact attention on the real decode path, $K_1 = 5$, $K_2 = 20$, all layers of the model (36 for the 4B, 40 for the 14B). Cosine similarity is averaged over heads within a layer; "mean" and "median" are then taken over layers.

D On the covariance model behind the cluster score

The cluster score of Equation 3 is the log moment-generating function of the attention logit $q \cdot k$ under a Gaussian model of the keys in a cluster, and the *principled* form of that model is a full covariance, not a diagonal one. If the real keys of cluster c are modeled as $k \sim \mathcal{N}(\mu_c, \Sigma_c)$ with $\Sigma_c \in \mathbb{R}^{d\times d}$, then

$$\log \mathbb{E}\big[e^{\,q\cdot k}\big] \;=\; q\cdot\mu_c \;+\; \tfrac{1}{2}\, q^\top \Sigma_c\, q \;=\; q\cdot\mu_c \;+\; \tfrac{1}{2}\sum_i q_i^2\,\Sigma_{c,ii} \;+\; \tfrac{1}{2}\sum_{i\neq j} q_i q_j\,\Sigma_{c,ij}, \qquad (7)$$

so Equation 3 is exactly Equation 7 with the off-diagonal (cross-dimension) terms dropped, i.e. a mean-field approximation $\Sigma_c \approx \mathrm{diag}(\sigma_c^2)$. Two natural refinements sit between the diagonal and the full model. A *partial* covariance $\Sigma_c \approx \mathrm{diag}(\sigma_c^2) + U_c U_c^\top$ with $U_c \in \mathbb{R}^{d\times r}$ captures the r dominant correlated directions of each cluster at a cost of $O(dr)$ per score instead of $O(d^2)$; and, since real keys are not Gaussian, the log-MGF can be carried beyond second order through its cumulant expansion $\sum_{n\geq 1} \kappa_n(q)/n!$, whose third- and fourth-order terms require the skewness and kurtosis tensors of the cluster. Under the Gaussian model the Approx tier's weight $\exp s(c,m)$ is the exact expectation of the summed softmax numerator of the sub-cluster's members, so any of these refinements would tighten Approx as well as the Hot/Warm ranking.

We use the diagonal form for two reasons. The first is cost. A full covariance stores $d(d{+}1)/2$ values per (sub-)cluster – 8,256 for $d = 128$ – against $2d = 256$ for a mean and diagonal variance; with sub-clusters of roughly $N/(C\,M) \approx \sqrt{N}/\log_2 N$ keys (about 15 keys at $N = 128$k), the summary would be several times *larger* than the real keys it summarises, and scoring all $C \cdot M$ candidates would cost $O(C\,M\,d^2)$ per query per head, comparable to attending to the entire cache exactly. Even a rank-r correction multiplies the scoring traffic, which is already the critical path of the decode step (Section H), by $(1 + r)$. The second reason is that, in practice, the diagonal score is already accurate for what it is used for. The score has to *rank* clusters and sub-clusters, and to weight the residual mass; the $q \cdot \mu$ term dominates both, the diagonal variance term supplies the per-cluster spread correction that separates tight clusters from diffuse ones, and the omitted cross terms $q_i q_j \Sigma_{ij}$ carry mixed signs that largely cancel within the k-means clusters, whose members are close in key space to begin with. In our ablations, replacing the diagonal by a full or low-rank covariance, or adding the third-order cumulant, changed the selected Hot/Warm sets and the per-layer agreement with exact attention only marginally, while every variant paid the storage and scoring cost above. We attribute this to the dimensions being strongly correlated only where there is genuine signal, in which case the underlying tokens are among the top clusters and sub-clusters and are attended exactly.

E Why MoG needs QK-normalization

MoG is designed for models that normalize queries and keys per head before the rotary embedding (i.e. the query and keys are normalized to the surface of a d-dimensional sphere). All models in this paper have it. The method runs on models without QK-norm, but its accuracy at long context is noticeably worse there and this appendix explains why.

Everything goes through a Gaussian summary. MoG never looks at the keys of a cluster after the fit. Selection, and the weight of every collapsed pseudo-token, come from the diagonal Gaussian summary (μ, σ^2, n) recorded at prefill (Section 3.1). The summary is only as good as the keys are well behaved, and a per-head RMS normalization is what makes them well behaved: after it, every key has the same RMS up to a fixed per-channel gain, so key scale is identical across tokens and bounded across heads. Without it, the scale of a key is whatever the projection (and its bias, if the model has one) happens to produce. On such models a few heads end up with keys tens of times larger than the rest, and inside those heads a handful of channels carry most of the energy. These are the well-known outlier heads and channels of unnormalized attention, and they hurt MoG in two places.

The partition. k-means partitions by Euclidean distance. In a head whose keys have a few dominant channels, the partition is driven by those channels rather than by the directions the query cares about, and the clusters stop being groups of keys that respond alike to a query. Sub-clustering inherits the same problem. QK-norm removes the per-token magnitude differences that would otherwise pull the partition toward outlier tokens, and leaves only direction, which is what attention responds to.

The score. The more serious effect is on the score of Equation 3. It is a log moment-generating function, and its variance term $\frac{1}{2}\sum_i q_i^2 \sigma_{c,i}^2$ grows with the square of the key scale. In an outlier head a modest misfit of σ, which a diagonal model of a few dominant channels will always have, turns into a score error of many nats. The HOT and WARM tiers only rank clusters by $s(c)$, and a ranking tolerates a shared additive error reasonably well. The APPROX tier does not. It uses $\exp s(c, m)$ directly as the weight of a pseudo-token, so an error of ten nats is a factor of 2×10^4, enough to either drown the exact tiers or make the collapsed mass vanish. With QK-norm the variance term stays $O(1)$ and the score error stays small in every head: on the Qwen3 models used here we measured it at well under one nat for the typical head and below 2.5 nats for the worst head, with the same keys and queries the model uses at decode time. A single head per layer whose pseudo-token weights are wrong by e^{10} is enough to derail generation, and at 128k tokens the APPROX tier represents about 98% of every head's cache, so there is no slack for such heads.

Per-head versus whole-projection normalization. "QK-norm" names two different operations, and the difference matters here. Qwen3 applies RMS normalization to each head's own d-dimensional slice independently, so every head's keys are separately put on a common scale. OLMo-2 and OLMo-3 apply one RMS normalization to the *entire* concatenated projection, over heads $\times\, d$ dimensions at once; the per-token total is then fixed, but nothing equalizes one head against another, and a head that carries more of the energy keeps larger keys. Since the argument above is about per-head key scale, only the first form supplies what it asks for. We measured both. Table 8 reports the key statistics of Qwen3-4B-Instruct-2507 and OLMo-3-7B-Instruct over the same 8,192-token prompt, across every (layer, key/value head) pair.

	Qwen3-4B (per-head)	**OLMo-3-7B** (whole-projection)
key norm, max/median within a head (median)	1.20	1.74
key norm, max/median within a head (worst)	1.79	24.69
coefficient of variation of key norm (median)	0.055	0.177
(layer, head) pairs with outlier keys	0/288	186/1024
worst share of outlier keys in one head	0.0%	20.8%
score error σ, nats (median)	0.68	0.20
score error σ, nats (worst head)	1.61	3.81
heads above the 2.5-nat threshold	0/288	1/1024

Table 8: Key statistics under the two forms of QK-norm, measured on the models' own keys at 8k tokens. "Outlier keys" are keys whose norm exceeds three times their head's median. The score error is the standard deviation of the difference between Equation 3 and the true log-sum-exp of a cluster's members, the quantity the APPROX tier exponentiates.

The measurement separates the two claims this appendix makes. The structural one holds: whole-projection normalization does *not* equalize per-head key scale, and OLMo-3 shows exactly the dispersion Qwen3 lacks – 186 of its 1,024 (layer, head) pairs contain keys more than three times their head's median norm, up to a fifth of a head's keys in the worst case, against not one such pair anywhere in Qwen3. The consequential one does not follow from it: the score error that this dispersion is supposed to produce stays small, with a *lower* median than Qwen3's and exactly one head of 1,024 above the 2.5-nat threshold at which the APPROX tier's weights would be meaningfully wrong. What MoG actually needs is therefore narrower than per-head normalization: it needs the per-head log-MGF score to be accurate, and a model can deliver that without normalizing per head. Per-head QK-norm remains the sufficient condition we can check by inspection; the score error of Table 8, which MoG can compute at fit time from the statistics it already builds, is the condition that actually matters.

Practical consequence. We treat a bounded per-head score error as the requirement, and per-head QK-norm as the easiest way to be confident of it in advance. For a model that fails it the options are to raise K_1 and K_2, which costs bandwidth, or to detect the offending heads at fit time from the statistics MoG already computes and route those heads to exact attention. We have not evaluated the second option.

F STANDALONE KERNEL SPEED

Figure 5 reports the time of a single decode step of the attention kernel alone, for MoG and for vLLM's exact flash-attention decode kernel, on a single request with 32 query heads, 8 key/value heads and head size 128. The context length is increased from 8k tokens until one of the two kernels exhausts device memory. Both kernels are replayed from a CUDA graph, as in production; the MoG measurement includes selection, the four tiers and the merge, and the cluster shape follows Section 3.1 ($C \approx \sqrt{N}$, $M \approx \log_2 N$).

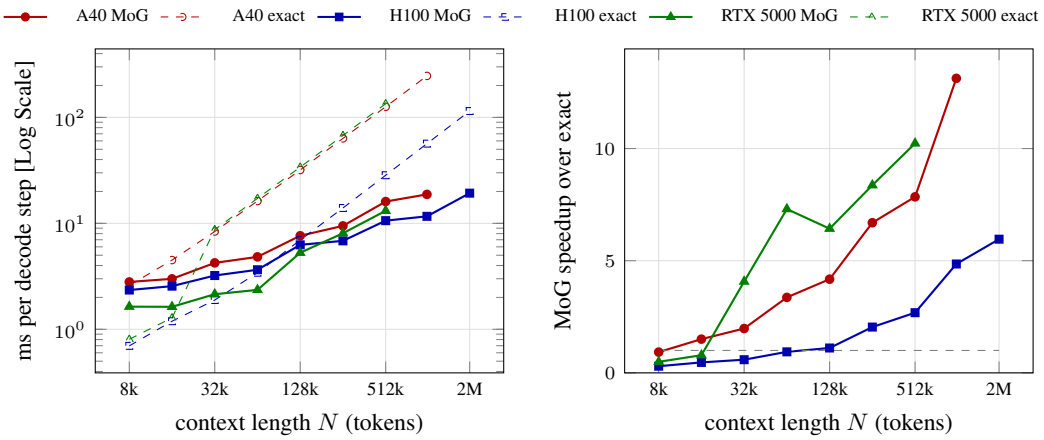


Figure 5: Decode-step time of the attention kernel alone, MoG vs. exact flash-attention decode, as a function of context length. Left: absolute time (solid: MoG, dashed: exact). Right: the ratio exact/MoG; the dashed line marks parity. Each sweep runs to the largest context length tested on that GPU (A40 1M tokens, H100 2M, RTX 5000 512k), beyond which MoG's fit buffers no longer fit in device memory while exact attention, which allocates nothing beyond the cache itself, still runs.

The context length where MoG step overtakes the $O(N)$ exact step depends on the GPU. To elaborate, MoG is faster than exact attention from 16k tokens on the A40, from 32k on the RTX 5000, and from approximately 128k on the H100, whose memory bandwidth keeps the exact kernel competitive over a wider range. These crossovers are later than those of Section 4.1, which times the full 36-layer step in the offloading harness rather than the attention kernel alone. Beyond the intersection the ratio grows with every doubling of N, as the asymptotic gap in Table 10 predicts. The absolute times in the left panel show the same effect: the cost of exact attention grows linearly with

N, whereas the MoG step remains below 20 ms on the A40 up to one million tokens, where exact attention requires roughly 250 ms.

G INCREMENTAL CONTRIBUTION OF EACH TIER TO ACCURACY

Having specified all four tiers, we can ask why the pipeline needs the APPROX tier and not HOT alone. Table 9 quantifies the answer: we replay 32 greedy decode queries on four real 32k-token prompts of Qwen3-4B through the production fit and kernel, build the pipeline up one component at a time – HOT alone, then +WARM, then +APPROX, renormalizing without APPROX – and report the mean over layers of the relative L_2 error of each layer's attention output. TAIL is exact by construction and is omitted. HOT alone is not viable. Every cluster it does not pick is simply dropped, the softmax normalization is broken, and the error reflects this directly. WARM accounts for most of the improvement. APPROX then conserves the mass of every unread subcluster instead of discarding it, lowering the remaining error by about a quarter, to about 13%. The overall time and memory complexity of our approach is $O(\sqrt{N}\log N\, d)$ versus exact's $O(Nd)$, which is broken down in Appendix H.

Pipeline	**Rel. L_2 (mean)**
HOT only (renormalized)	62%
+ WARM (renormalized)	17%
+ APPROX *(full pipeline)*	13%

Table 9: Per-tier accuracy build-up: relative L_2 error of each layer's attention output against exact attention, mean over layers, on real decode queries (Qwen3-4B, 32k). Each row adds one tier to the row above it.

H COMPUTATIONAL COMPLEXITY

The hierarchical partitioning scheme described in Section 3.1 admits a compact asymptotic characterization across computational time, memory bandwidth (bytes transferred per decode step), and overall memory footprint. Throughout this analysis, N denotes the context length, d the head dimension, and T the number of tokens generated post-fit. All quantities are expressed per layer and per key/value head, with parameters K_1 and K_2 treated as constants.The cluster topology is parameterized by $C = \Theta(\sqrt{N})$ clusters and $M = \Theta(\log N)$ sub-clusters per cluster. Consequently, an average cluster contains $N/C = \Theta(\sqrt{N})$ tokens, while an average sub-cluster contains $N/(CM) = \Theta(\sqrt{N}/\log N)$ tokens.

H.1 TIME COMPLEXITY PER DECODE STEP

Table 10 summarizes the per-step operational costs. Cluster selection requires scoring C clusters alongside CM sub-clusters (for the WARM tier and APPROX weight computation). Each score comprises a dot product and a weighted sum of squares against a statistic vector, yielding an aggregate operational cost of $O(CMd) = O(\sqrt{N}\log N\, d)$. The subsequent top-$k$ selections over C and CM candidates contribute an additional $O(\sqrt{N}\log N)$ operations, independent of d.Exact attention is computed over K_1 clusters and K_2 sub-clusters, corresponding to $K_1\sqrt{N} + K_2\sqrt{N}/\log N$ tokens, at $O(d)$ cost per token. The APPROX tier evaluates a single moment-generating-function weight for each residual sub-cluster at cost $O(CMd)$, whereas the TAIL tier computes exact attention over the T newly generated tokens. Finally, merging the four candidate triples via online softmax requires $O(d)$ operations.The total per-step time complexity is $O(\sqrt{N}\log N\, d + Td)$, compared to $O(Nd)$ for standard exact attention. The logarithmic factor stems from sub-cluster scoring; the real-attention term is asymptotically dominated as soon as $K_1\sqrt{N} \ll \sqrt{N}\log N$ (i.e., when $K_1 < \log N$), a condition satisfied across all evaluated sequence lengths. Under grouped-query attention (GQA), key/value statistics are read once per group, whereas candidate scores and selected slices are evaluated per query head; query-side terms are thus scaled by the group size.

	Exact attention	**MoG**
Cluster and sub-cluster scoring	—	$O(\sqrt{N}\log N\, d)$
Top-k selection	—	$O(\sqrt{N}\log N)$
Exact attention (HOT, WARM)	$O(N\, d)$	$O\big((K_1\sqrt{N} + K_2\frac{\sqrt{N}}{\log N})\, d\big)$
Analytic residual (APPROX)	—	$O(\sqrt{N}\log N\, d)$
Generated tokens (TAIL)	(in the $O(Nd)$ above)	$O(T\, d)$
Total	$O(N\, d)$	$O(\sqrt{N}\log N\, d + T\, d)$

Table 10: Per-decode-step computational complexity per layer and key/value head. For fixed K_1, K_2, the exact-attention overhead of MoG becomes negligible relative to the scoring and residual evaluation terms as N grows, yielding sub-linear complexity in N.

H.2 MEMORY BANDWIDTH PER STEP

Because autoregressive decoding is memory-bandwidth bound, memory transfer volume dictates step latency. The byte transfer requirements directly track the operational terms (Table 11). Standard exact attention requires reading the entire key-value cache ($2Nd$ elements).Conversely, MoG reads only the required statistics—comprising $(2C + 3CM)d$ elements (a mean and variance per cluster, plus a mean, variance, and mean value per sub-cluster)—alongside the selected attention slices, totaling $2(K_1\sqrt{N}+K_2\sqrt{N}/\log N)d$ elements. Write overhead for scores and indices is $O(CM)$ and statistically negligible. The memory-bandwidth reduction factor scales as $\Theta(\sqrt{N}/\log N)$, yielding empirical speedups of $18\times$ at $N = 10^6$ and $49\times$ at $N = 10^7$ (Section 4.1).

Memory Footprint and Offloading. The storage layout described in Section 3.1 permutes rather than compresses the KV cache. It retains $2Nd$ elements, supplemented by zero-padding to align sub-clusters within each cluster to a common dimension (incurring a 0.4%–8% capacity overhead across evaluated configurations). Auxiliary statistics contribute $(2C + 3CM)d = O(\sqrt{N}\log N\, d)$ elements. As a fraction of total memory, this overhead diminishes as $O(\log N/\sqrt{N})$, dropping from 33% at $N = 8$k to 1.5% at $N = 10^7$.Consequently, MoG does not substantially compress total cache memory; rather, it drastically reduces the active working set required per decode step. Only the statistical vectors and a localized slot cache (holding recently selected slices) must remain resident in VRAM ($O(\sqrt{N}\log N\, d)$ elements). The primary store can be offloaded to host RAM and streamed asynchronously on demand, whereas standard attention requires either hosting all $2Nd$ elements in GPU memory or transferring the complete cache at every step (Section 4.1).The tail buffer requires an additional $2Td$ elements, identical to standard attention implementations. Fitting the structure post-prefill requires transient memory for one layer of gathered keys ($O(Nd)$) and assignments ($O(N)$). The fitting step incurs $O(INCd) = O(N^{3/2}d)$ time for I iterations of k-means with C centroids, alongside $O(N\log N)$ for intra-cluster rank sorting and $O(Nd)$ to populate the store. This constitutes the sole super-linear operation of the method and is incurred once per request. In our vLLM integration, the primary store is constructed alongside the engine's native paged cache, doubling memory usage during decoding; standalone and offloaded implementations maintain only the primary store.

	Exact attention	**MoG**
Store (resident or offloaded)	$2Nd$	$2Nd + O(\sqrt{N}\log N\, d)$
Read per decode step	$2Nd$	$O(\sqrt{N}\log N\, d)$
Minimum resident on the GPU	$2Nd$, or streamed per step	$O(\sqrt{N}\log N\, d)$ statistics and slot cache
Transient at the fit	—	$O(Nd)$ keys, $O(N)$ assignments
Generated tokens	$2Td$	$2Td$

Table 11: Memory requirements (in elements) per layer and key/value head, where sub-cluster alignment padding adds 0.4–8% to the store. Although MoG preserves overall storage capacity, it significantly reduces per-step memory bandwidth and active GPU memory requirements.

I DECODE COST FOR OTHER MODELS

Figures 6 to 9 repeat the measurements of Section 4.1 for the two other model shapes of the paper, under the setup of Appendix B.1: Qwen3-14B, whose 40 layers all attend globally, and OLMo-3-7B, where only the 8 full-attention layers of 32 carry a cache that grows with the context and are the only layers MoG replaces. The curves show the same behavior as Qwen3-4B, with the crossover and the offloading cliff at the same context lengths.

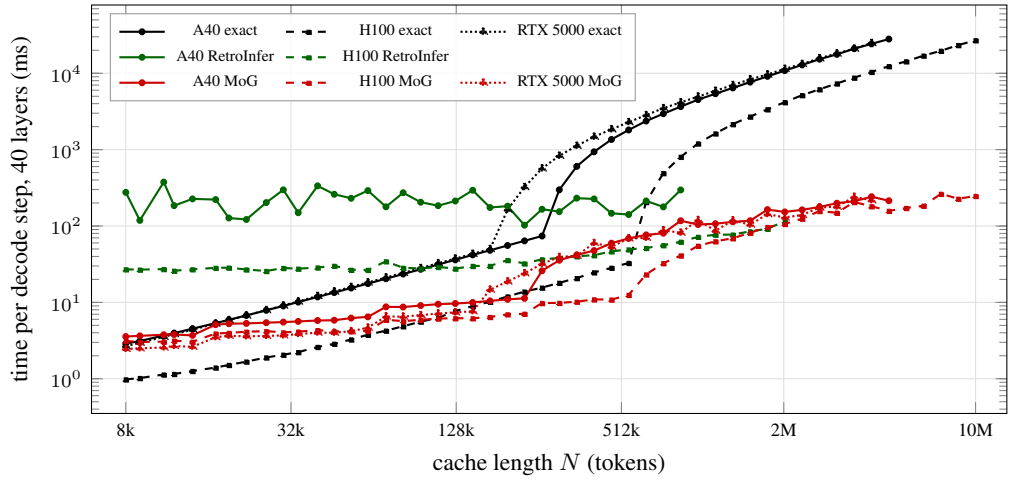


Figure 6: Qwen3-14B (40 query heads, 8 key/value heads, head size 128): time of one steady-state decode step through its 40 attention layers, as in Figure 3.

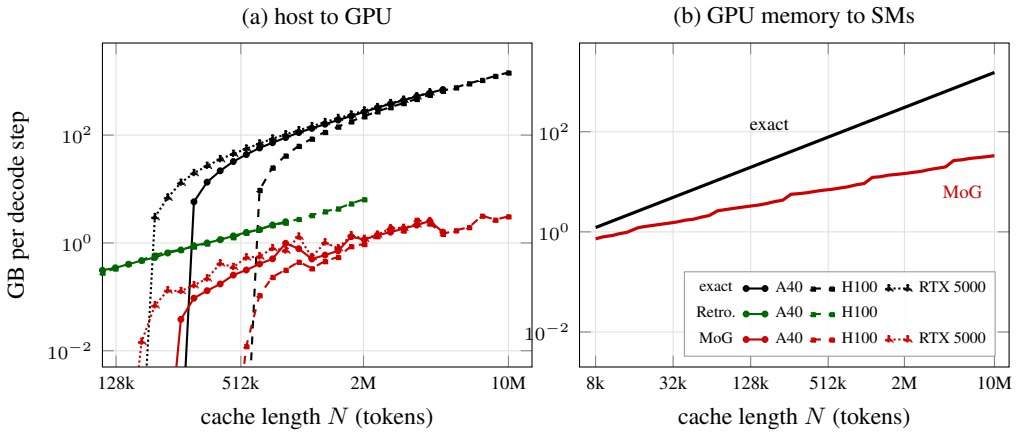


Figure 7: Qwen3-14B (40 query heads, 8 key/value heads, head size 128): data movement per decode step through its 40 attention layers, as in Figure 4.

J COLLAPSED GENERATIONS BY CELL

Table 12 gives, for every cell of the reasoning grid, the share of generations that collapsed – the model emitting token 0 ("!") for the remainder of the output, which follows non-finite logits in the attention path. Exact attention and MoG never emit token 0 (their 1–3% are repetition loops). RetroInfer does, on both of its cache implementations, at a rate that rises with context length on Qwen3-4B and is total on OLMo-3 at 32k (Appendix B.3 gives the cause); RetroInfer is not reported on OLMo-3 beyond 32k.

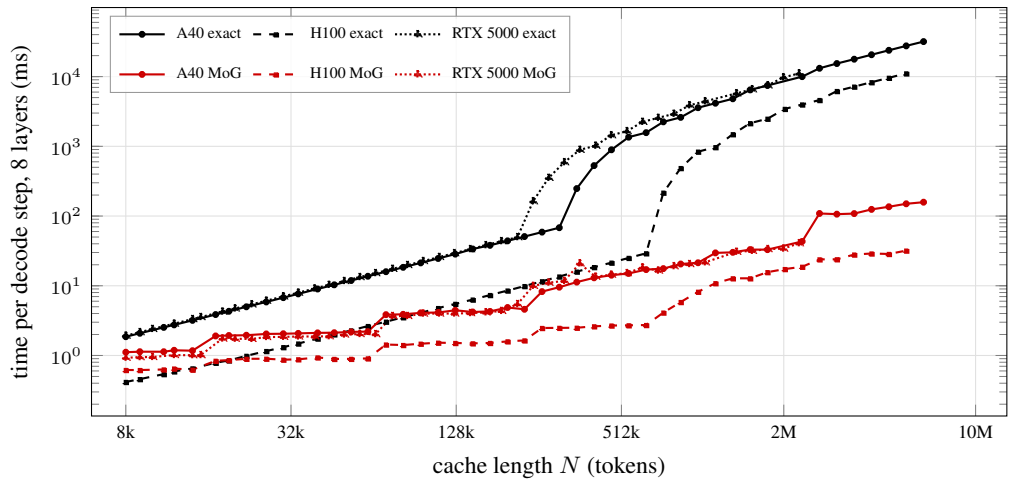


Figure 8: OLMo-3-7B (32 query heads, 32 key/value heads, head size 128; the 8 full-attention layers of its 32, the other 24 attend over a 4k sliding window): time of one steady-state decode step through its 8 attention layers, as in Figure 3.

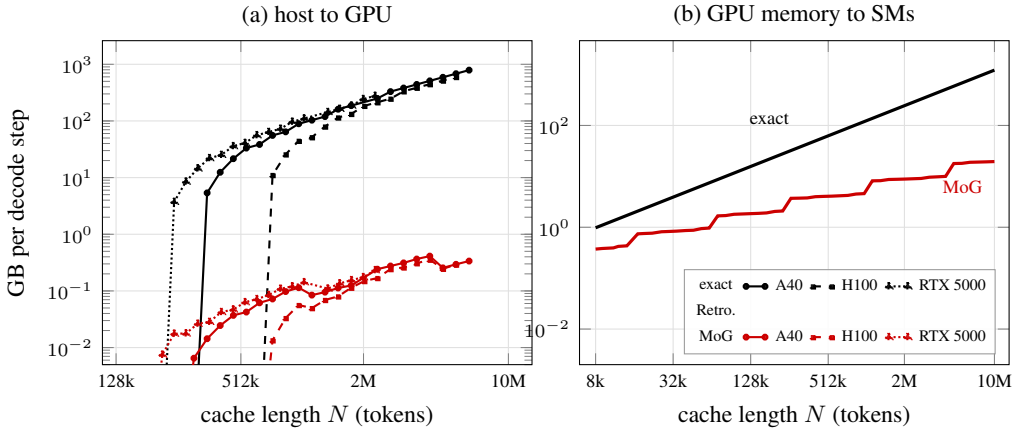


Figure 9: OLMo-3-7B (32 query heads, 32 key/value heads, head size 128; the 8 full-attention layers of its 32, the other 24 attend over a 4k sliding window): data movement per decode step through its 8 attention layers, as in Figure 4.

Model	Benchmark	Length	exact	MoG	Retro.
Qwen3-4B-Instruct-2507	LONGREASON	8k	0%	0%	0%
Qwen3-4B-Instruct-2507	LONGREASON	32k	0%	0%	3%
Qwen3-4B-Instruct-2507	LONGREASON	100k	1%	0%	62%
Qwen3-4B-Instruct-2507	LOCOBENCH	8k	0%	0%	0%
Qwen3-4B-Instruct-2507	LOCOBENCH	32k	0%	0%	12%
Qwen3-4B-Instruct-2507	LOCOBENCH	100k	0%	0%	65%
Qwen3-4B-Instruct-2507	LONGBENCH-V2	natural	1%	3%	31%$^{\dagger 5}$
Qwen3-4B-Instruct-2507	AA-LCR	natural	0%	0%	37%
OLMo-3-7B-Instruct	LONGREASON	8k	0%	0%	0%
OLMo-3-7B-Instruct	LONGREASON	32k	0%	0%	*coll.*
OLMo-3-7B-Instruct	LONGREASON	128k	0%	0%	–
OLMo-3-7B-Instruct	LOCOBENCH	8k	0%	0%	0%
OLMo-3-7B-Instruct	LOCOBENCH	32k	0%	0%	*coll.*
OLMo-3-7B-Instruct	LOCOBENCH	128k	0%	0%	–
OLMo-3-7B-Instruct	LONGBENCH-V2	natural	0%	0%	–
OLMo-3-7B-Instruct	AA-LCR	natural	0%	0%	–
Qwen3-14B	LONGREASON	8k	0%	0%	0%
Qwen3-14B	LONGREASON	32k	0%	0%	0%
Qwen3-14B	LONGREASON	128k	0%	0%	0%
Qwen3-14B	LOCOBENCH	8k	0%	0%	0%
Qwen3-14B	LOCOBENCH	32k	0%	0%	0%
Qwen3-14B	LOCOBENCH	128k	0%	0%	0%
Qwen3-14B	LONGBENCH-V2	natural	1%	1%	1%
Qwen3-14B	AA-LCR	natural	0%	0%	0%

Table 12: Share of generations that collapsed to a run of token 0, per cell. *coll.*: every generation collapsed and the cell was not scored; – : cell not run or not applicable. $^{\dagger k}$: k prompts on which RetroInfer's index build itself crashed (an illegal memory access in the upstream kernel on ≈130k-token prompts, reproduced on three attempts); they are counted as unanswered (score 0) in both this rate and the accuracy tables.