More Value per Key: Asymmetric Sparse Attention for Faster LLM Decoding

Noam Elata[*]
Technion – Haifa, Israel
noamelata@technion.ac.il

Itay Lamprecht[*†]
Technion – Haifa, Israel
Crusoe AI
itayl@campus.technion.ac.il

Mikey Shechter[*]
Technion – Haifa, Israel
m.sachter@technion.ac.il

Daniel Ohayon
Corma
Technion – Haifa, Israel
ohayon.daniel4@gmail.com

Itay Hubara
Stealth Startup
itayhubara@gmail.com

Daniel Soudry
Technion – Haifa, Israel
daniel.soudry@gmail.com

Abstract

Autoregressive generation in Large Language Models (LLMs) is constrained by the memory and computational demands of attention mechanisms. Sparse attention methods mitigate this cost by selecting only high-probability entries of the attention matrix. We observe that in many such methods, this renders the probability-value multiplication negligible, shifting the bottleneck to the query-key step. Key heads can therefore be reduced to accelerate inference, while retaining more value heads preserves capacity with limited additional decoding cost. We introduce Sparse Asymmetric Group-Query Attention (SAGA), which decouples key and value head counts to exploit this principle, and pair it with approximate top-N (Atop-N) attention, a simple sparse attention method designed to study the interaction between sparsity and head-count asymmetry. We formalize the benefits of this asymmetry theoretically and validate them empirically through latency measurements and quality evaluations on models up to 1.5B parameters. Together, SAGA and Atop-N achieve end-to-end decoding speedups exceeding $2\times$ over our full-attention GQA baseline at long contexts. Models trained from scratch with SAGA nearly match the quality of comparable GQA variants on the evaluated benchmarks. To facilitate adoption, we introduce an efficient fine-tuning method that converts pretrained models to the SAGA architecture, enabling practitioners to benefit from our approach without costly retraining.[1]

1 Introduction

Attention-based architectures have become dominant across AI applications [1], with large language models such as GPT [2, 3], Llama [4], and Qwen [5, 6] demonstrating strong capabilities in natural language understanding [7], generation [8], and reasoning [9]. However, the quadratic scaling of attention with sequence length [10, 11] remains a significant bottleneck, constraining context window sizes and limiting the ability of models to reason over lengthy documents.

For autoregressive generation, KV caching [12] provides substantial speedups by reusing key and value computations across generation steps. This shifts the bottleneck from compute capacity to

[*]Equal contribution.
[†]Work was partially done while the author was at Intel, Israel.
[1]Code is available at https://github.com/noamelata/SAGA.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

memory capacity and bandwidth [13]. Group-Query Attention (GQA) [14] further exploits this setting by reducing the number of key-value heads, accelerating inference for long sequences. Yet even with these optimizations, bounded context windows remain a practical necessity.

Sparse attention methods [15–19] offer a complementary direction, accelerating attention by computing only a subset of query-key interactions. Such methods span both the prefill and decoding phases, and range from zero-shot approximations to approaches integrated into pre-training. In this work, we focus on the decoding phase and use a generic sparsity method, as primary examples of the effects of sparsity on the computational profile of generation. To this end, we adopt top-N attention as a representative sparsity method. Building on previous observations [20, 21] that models assign significant attention weight to only a small subset of keys, top-N attention has the advantage of simplicity: it requires no training to apply and admits a portable implementation, a property uncommon among sparse attention methods. Our efficient approximate top-N attention (Atop-N) implementation gathers only the most relevant attention entries, reducing the probability-value multiplication to a fraction of its original cost.

Analyzing the resulting computational profile reveals a fundamental asymmetry: under sparse attention, the bottleneck shifts from the probability-value multiplication to the query-key multiplication. This insight motivates Sparse Asymmetric Group-Query Attention (SAGA), an architectural modification that decouples the number of key heads h_k from value heads h_v. While GQA ties these together, SAGA reduces h_k independently, directly accelerating the query-key multiplication. The number of value heads remains large to preserve model capacity, incurring no additional cost since only the top-N values are accessed. We formalize this asymmetry theoretically and validate it empirically, demonstrating end-to-end speedups exceeding $2\times$ in long contexts while maintaining the majority of baseline quality.

To facilitate practical adoption, we introduce an efficient fine-tuning method for converting existing GQA models to the SAGA architecture. We merge teacher key heads using a closed-form initialization derived from minimum-weight perfect matching on attention logit reconstruction error (Sec. 4), followed by a short distillation phase. This enables the benefits of SAGA without costly retraining.

To summarize, our contributions are:

- We present a novel analysis of LLM inference trade-offs with accelerated sparse attention.
- We introduce Atop-N, a simple and portable sparse attention algorithm for decoding.
- Motivated by our finding, we propose SAGA, an extension of the GQA architecture tailored for sparse attention inference. We show that SAGA is competitive both when trained from scratch and when finetuned from existing LLM transformers, while demonstrating more than $2\times$ runtime speedups end-to-end on long sequence generation.

2 Background and related work

2.1 Multi-head self-attention

The Transformer architecture, introduced by Vaswani *et al*. [1], revolutionized natural language processing. Its core mechanism, multi-head self-attention, enables parallel processing of sequences and captures long-range dependencies by allowing the model to attend to all positions simultaneously.

Formally, given a hidden state $\mathbf{X} \in \mathbb{R}^{l_q \times D}$, we apply three projections $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{D \times (h \cdot d)}$ to obtain queries $\mathbf{Q} \in \mathbb{R}^{h \times l_q \times d}$, keys $\mathbf{K} \in \mathbb{R}^{h \times l_{kv} \times d}$, and values $\mathbf{V} \in \mathbb{R}^{h \times l_{kv} \times d}$:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V, \tag{1}$$

where h is the number of heads, D and d are the hidden and head dimension sizes respectively, and l_q and l_{kv} are the query and key-value sequence lengths. The attention output is computed as:

$$\mathbf{A} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} \tag{2} \qquad \mathbf{P} = \text{softmax}(\mathbf{A}) \tag{3} \qquad \mathbf{O} = \mathbf{P}\mathbf{V}, \tag{4}$$

where $\mathbf{A}, \mathbf{P} \in \mathbb{R}^{h \times l_q \times l_{kv}}$ denote the attention and probability matrix respectively. The output $\mathbf{O}$ is then multiplied by $\mathbf{W}_O \in \mathbb{R}^{(h \cdot d) \times D}$. Modern LLMs stack many such layers in a decoder-only design, modeling complex relationships between all sequence elements regardless of distance.

2.2 KV cache

Key-Value (KV) caching [12] is an optimization technique that significantly improves the inference efficiency of autoregressive Transformer models. During the generation of each new token, Transformer models would typically recompute the key and value projections for all previous tokens in the sequence. The KV cache stores projections from previous steps, eliminating redundant computations.

For an autoregressive model generating the l_{kv}-th token, the attention computation with KV caching can be expressed as:

$$\mathbf{O}_{l_{kv}} = \text{softmax}\left(\frac{\mathbf{Q}_{l_q}[\mathbf{K}_{1:(l_{kv}-1)}, \mathbf{K}_{l_{kv}}]^\top}{\sqrt{d}}\right)[\mathbf{V}_{1:(l_{kv}-1)}, \mathbf{V}_{l_{kv}}], \tag{5}$$

where $\mathbf{K}_{1:(l_{kv}-1)}$ and $\mathbf{V}_{1:(l_{kv}-1)}$ are retrieved from the cache, and only $\mathbf{Q}_{l_q}$, $\mathbf{K}_{l_{kv}}$, and $\mathbf{V}_{l_{kv}}$ need to be computed for the new token.

By caching key and value vectors from previous time steps, the model only needs to compute new projections for the most recently generated token, leading to substantial speedups during inference. This optimization is particularly valuable for long-form text generation, where the model must generate many tokens sequentially. The trade-off is increased memory usage proportional to sequence length, as the cache must store key-value pairs for all tokens in the context window.

2.3 Group-query attention and head reduction

Group-Query Attention (GQA) [14] addresses the memory overhead of KV caching by recognizing an asymmetry in the attention computation during decoding: there are significantly more cached keys and values than queries ($l_{kv} \gg l_q$). In this context, it is sensible to allocate more of the model's capacity to the queries, which are computed fresh at each decoding step, rather than to the keys and values, which are retrieved from the cache.

GQA achieves this by decoupling the number of key-value heads h_{kv} from the number of query heads h_q, such that $h_q > h_{kv}$. Multiple query heads share a single key-value head, reducing the size of the KV cache while preserving the expressiveness of the query projections. This approach yields a significant improvement in inference efficiency with negligible degradation in model quality, making it a widely adopted technique in modern LLM architectures.

Beyond reducing the total number of KV heads, recent work has explored making key and value head counts asymmetric. DHA [22] fuses similar attention heads in MHA architecture into decoupled per-layer key and value configurations to reduce memory footprint. Sigma [23] performs differential rescaling of query, key, and value dimensions during pretraining and finds that reducing key capacity is preferable to reducing value capacity—an observation consistent with our analysis. While conceptually related, SAGA differs in both motivation and applicability, exploiting the shifted computational bottleneck under sparse attention to accelerate decoding. Multi-head latent attention with DeepSeek Sparse Attention [24, 25] is related in spirit but must be trained from scratch with a shared latent KV and a learned indexer; it reduces the number of key *positions* scored per query, whereas SAGA reduces the number of key *heads*, so the two are orthogonal rather than competing.

2.4 Attention sparsity and KV cache acceleration

Recent efforts to mitigate the quadratic complexity of the attention mechanism have largely focused on attention sparsity and KV cache compression. Sparse attention methods, such as BigBird [17], Longformer [16], and Sparse Transformer [15], utilize predefined patterns like sliding windows or global tokens to restrict the number of query-key pairs computed. While effective for the prefill phase, these static patterns often struggle to capture the dynamic, data-dependent importance of tokens during autoregressive decoding. NSA [19], SpargeAttention [26], BLASST [27], and BSFA [28] also aim to bridge the gap between algorithmic sparsity and hardware efficiency, mostly focusing on the more compute-intensive pre-fill phase. Our work, SAGA, focuses instead on the decode phase, and offers a new framework by combining a simple attention sparsity method with a dedicated neural architecture.

Other methods accelerate decoding by evicting low-importance tokens from the KV cache [29, 30] or selectively loading cache pages using query-aware metadata [31, 32]. Complementary techniques

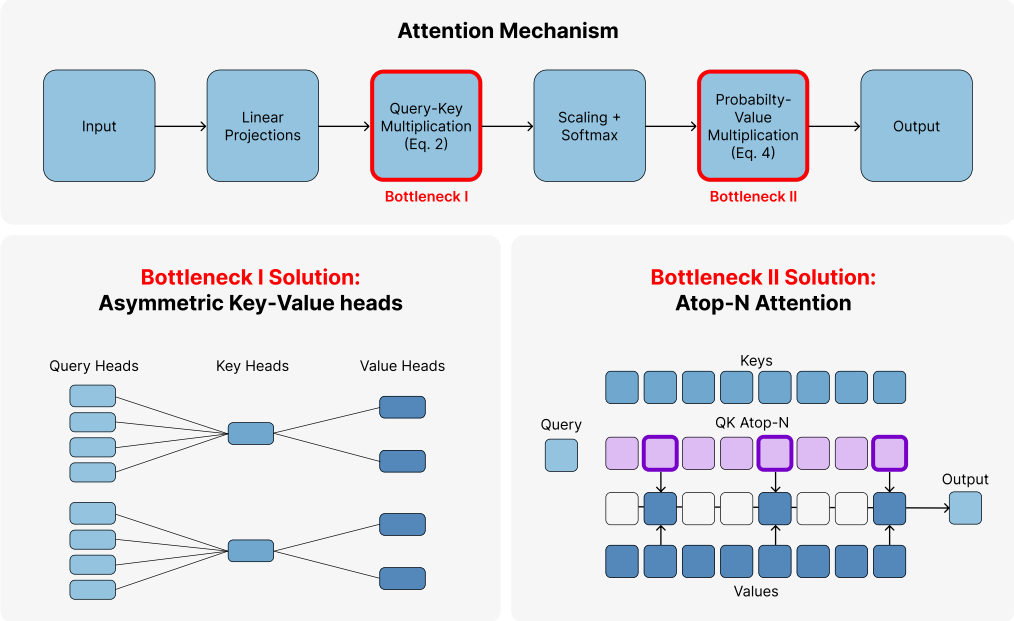


Figure 1: Diagram of SAGA. There are two significant decoding bottlenecks in attention mechanisms: key-query multiplications and probability-value multiplications (top). SAGA addresses the query-key bottleneck by reducing the number of key heads (left). The probability-value bottleneck is mitigated by utilizing Atop-N to select important values (right).

reduce cache memory via quantization [33, 34]. SAGA is orthogonal to these approaches; it optimizes the head architecture itself and can be readily combined with cache compression methods.

2.5 Top-N attention

Top-N attention [20, 35] (also known as top-K attention or kNN attention) is based on the observation that only a small subset of attention keys meaningfully influence the attention output. This sparsity arises from the softmax operation, which effectively learns to identify the best query-key matches while suppressing mismatched pairs.

Formally, top-N attention identifies the indices of the N largest values in each row of $\mathbf{A}$:

$$\mathbf{i}_{j,k,1:N} = \text{TopN}(\mathbf{A}_{j,k,1:l_{kv}}), \tag{6}$$

and masks all other entries of $\mathbf{A}$ not in $\mathbf{i}$ before applying the softmax. This formulation has been shown to enable memory-efficient implementations [20, 35], though it does not yield computational speedups since the full attention matrix must still be computed. Several recent works have extended top-N attention in various directions [36, 37] or strengthened its theoretical foundations [38].

3 Method

We begin by introducing the attention sparsity method underlying SAGA. For clarity, we present an architecture-agnostic formulation of top-N attention to illustrate our approach and the trade-offs many online sparse attention method induce. In the following analysis, we show that under sparse attention, the computational bottleneck shifts from the probability-value multiplication to the query-key multiplication—the property that SAGA is designed to exploit.

3.1 Efficient approximate top-N attention for LLM decoding

We propose an efficient framework for top-N attention designed for LLM decoding. Using the indices $\mathbf{i} \in \mathbb{Z}^{h \times l_q \times N}$ from Eq. (6), we define the **Top-N Attention Matrix** $\mathbf{A}[\mathbf{i}] \in \mathbb{R}^{h \times l_q \times N}$ and corresponding **Top-N Values** $\mathbf{V}[\mathbf{i}] \in \mathbb{R}^{h \times l_q \times N \times d}$, and compute attention over only the gathered entries:

$$\mathbf{O} = \mathrm{softmax}(\mathbf{A}[\mathbf{i}])\mathbf{V}[\mathbf{i}]. \tag{7}$$

In autoregressive generation ($l_q = 1$), this reduces the probability-value cost from $O(h\, l_{kv}\, d)$ to $O(h\, N\, d)$, potentially saving nearly 50% of computation and bandwidth when $N \ll l_{kv}$. However, naïvely finding the true top-N entries incurs lengthy hardware runtimes, negating the benefit.

We address this with an approximate threshold approach: given the N-th largest attention value as a threshold, top-N entries can be identified by a simple per-element comparison — a parallel, portable operation much faster than sorting. Since attention distributions shift only marginally between consecutive decoding steps, we compute exact thresholds once every n tokens and reuse them, amortizing the cost over the full text generation. We term this method Atop-N attention; full implementation details are in Sec. C.

For long sequences, Atop-N yields an overall speedup of nearly $2\times$ for the full attention operation. Section 5 shows that for practical values of N, Atop-N retains 83–97% of baseline RULER accuracy.

3.2 Sparse Asymmetric Group-Query Attention

The analysis in the previous section reveals that under Atop-N attention, the probability-value multiplication accesses only $h_q \times l_q \times N$ value vectors, N per query head, regardless of the number of key-value heads h_{kv}. Consequently, h_{kv} affects computational complexity only through the query-key multiplication in Eq. (2). This asymmetry suggests that the number of key heads and value heads should be disentangled.

We propose Sparse Asymmetric Group-Query Attention (SAGA), which decouples the key and value head counts into separate hyperparameters h_k and h_v. Reducing h_k directly reduces the memory bandwidth required to load keys during the query-key multiplication step (Eq. (2)), the dominant latency bottleneck in LLM decoding under Atop-N attention. The number of value heads h_v can remain large to preserve model capacity, since only the top-N values are accessed regardless of h_v. A diagram of SAGA is shown in Fig. 1.

Beyond the bandwidth reduction, fewer key heads yield a smaller key projection $\mathbf{W}_K$ (reducing per-token compute) and a smaller KV cache footprint, enabling longer context lengths or larger batch sizes within the same memory budget.

SAGA can be applied directly when training from scratch, as detailed in Sec. 5.2. To leverage existing pretrained models, we describe a procedure for converting GQA models to SAGA in Sec. 4 below.

4 Converting GQA models to SAGA

We describe a practical procedure for converting a pretrained GQA model into a SAGA model. Our goal is to transform the attention module of an existing GQA model T, with equal numbers of key and value heads

$$h_k^T = h_v^T, \tag{8}$$

into an attention module of a SAGA model S with fewer key heads than value heads,

$$h_k^S < h_v^S, \tag{9}$$

while preserving the attention behavior as closely as possible. The conversion consists of two stages:

1. Choosing an initialization for the SAGA key projection matrices $\mathbf{W}_K^S$ that minimizes deviation from the original GQA attention, as described below.
2. Distillation-based fine-tuning, using the original GQA model T as a teacher and the SAGA model S as a student, as implemented in Sec. 5.3.

Our GQA to SAGA conversion recipe naturally extends to multi-head attention (MHA) models.

4.1 Initializing SAGA from GQA

We consider a GQA model with h_k^T key heads, $h_v^T = h_k^T$ value heads, and h_q query heads. We convert it to a SAGA model with fewer key heads h_k^S while keeping the same number of value heads $h_v^S = h_v^T$. We assume the reduction factor is a positive integer $m \geq 2$

$$h_k^T = m \cdot h_k^S \qquad (m \in \mathbb{N}), \tag{10}$$

so that each SAGA key head is shared by exactly m value heads. For simplicity we describe the case $m = 2$, i.e. $h_k^T = 2h_k^S$, and note that the construction extends directly to any $m \in \mathbb{N}$ by replacing the minimum-weight perfect matching in Theorem 4.2 with a minimum-weight perfect m-partitioning of G. Let $\mathbf{W}_K^1, \ldots, \mathbf{W}_K^{2h_k^S}$ denote the key projection matrices, and $\mathbf{W}_Q^1, \ldots, \mathbf{W}_Q^{h_q}$ the query projection matrices of an attention module of the teacher model T.

Let $\mathbf{X} \in \mathbb{R}^{L \times D}$ denote a matrix of hidden states sampled from a representative dataset. Our objective is to construct new key projections $\hat{\mathbf{W}}_K^1, \ldots, \hat{\mathbf{W}}_K^{h_k^S}$ for the SAGA model such that the resulting attention logits

$$\frac{(\mathbf{X}\mathbf{W}_Q^j)(\mathbf{X}\hat{\mathbf{W}}_K^i)^\top}{\sqrt{d}} \tag{11}$$

closely approximate the corresponding logits produced by the original GQA model for every query head j that originally shared one of the two merged teacher key heads. We choose to optimize only the merged key projections, and to match the attention logits while ignoring rotary embeddings because this is the broadest objective that enables us to derive a closed-form solution. Matching attention probabilities, or jointly optimizing the query and key projections, leads to a nonlinear problem with no closed-form minimizer.

Reducing the number of key heads requires deciding which original key heads should be merged together. We formalize this choice using a permutation $\sigma = (\sigma_1, \ldots, \sigma_{2h_k^S})$ of $\{1, \ldots, 2h_k^S\}$. The permutation specifies a partition of the original key heads into h_k^S disjoint pairs:

$$(\sigma_{2i-1}, \sigma_{2i}), \quad i = 1, \ldots, h_k^S. \tag{12}$$

Each pair is merged into a single SAGA key head. Once σ is determined, we apply the induced permutations to the query, value, and output projection matrices so that they remain aligned with the reordered key heads and each merged key is shared by its intended value-head pair. The induced permutations definitions are given in Sec. B. Now we explain how we choose the permutation and merge the chosen key heads. For each teacher key head $r \in \{1, \ldots, 2h_k^S\}$, let $\mathcal{Q}_r$ denote the query heads that share $\mathbf{W}_K^r$, and let $\mathbf{Q}^j := \mathbf{X}\mathbf{W}_Q^j$ denote the realized query matrices. For any pair of teacher key heads $a, b \in \{1, \ldots, 2h_k^S\}$ and candidate merged projection $\mathbf{M}$, define the pairwise logit-matching loss

$$\begin{aligned} \mathcal{L}_{a,b}(\mathbf{M}) := &\sum_{j \in \mathcal{Q}_a} \left\| \mathbf{Q}^j(\mathbf{X}\mathbf{M})^\top - \mathbf{Q}^j(\mathbf{X}\mathbf{W}_K^a)^\top \right\|_{\mathrm{F}}^2 \\ &+ \sum_{j \in \mathcal{Q}_b} \left\| \mathbf{Q}^j(\mathbf{X}\mathbf{M})^\top - \mathbf{Q}^j(\mathbf{X}\mathbf{W}_K^b)^\top \right\|_{\mathrm{F}}^2 \end{aligned} \tag{13}$$

where $\|A\|_{\mathrm{F}} = \sqrt{\sum_{i,j} A_{ij}^2}$ denotes the Frobenius norm of a matrix A. We omit the attention scaling factor $1/\sqrt{d}$, since it multiplies the entire objective by the constant $1/d$ and therefore does not affect the minimizer. Our objective is to jointly choose the pairing σ and the merged key projections $\mathbf{M}^{[1:h_k^S]}$ so as to minimize the total squared logit reconstruction error over the dataset:

$$\min_{\sigma} \min_{\mathbf{M}^{[1:h_k^S]}} \sum_{i=1}^{h_k^S} \mathcal{L}_{\sigma_{2i-1}, \sigma_{2i}}(\mathbf{M}^i). \tag{14}$$

Lemma 4.1. *For any pair of teacher key heads $a, b \in \{1, \ldots, 2h_k^S\}$, an optimal merged key projection minimizing $\mathcal{L}_{a,b}(\mathbf{M})$ is*

$$\hat{\mathbf{W}}_K(a, b) = \left(\mathbf{W}_K^a \mathbf{S}_a + \mathbf{W}_K^b \mathbf{S}_b\right)(\mathbf{S}_a + \mathbf{S}_b)^\dagger. \tag{15}$$

where $\mathbf{S}_r := \sum_{j \in \mathcal{Q}_r} (\mathbf{Q}^j)^\top (\mathbf{Q}^j)$, and $\dagger$ denotes the Moore–Penrose pseudoinverse.

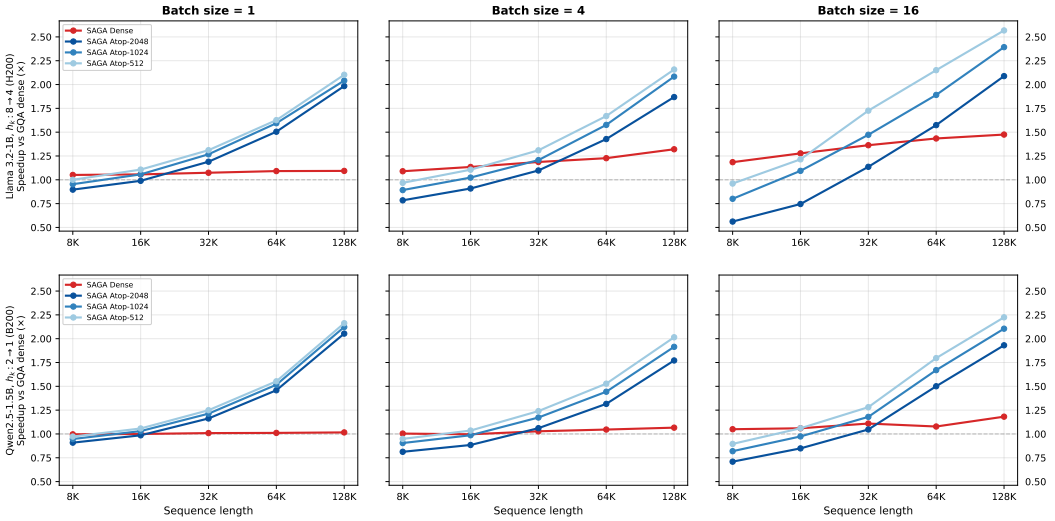


Figure 2: End-to-End speedup over baseline full attention as a function of sequence length, for Llama 3.2-1B ($h_k : 8 \to 4$, H200) and Qwen2.5-1.5B ($h_k : 2 \to 1$, B200) at various batch sizes. Merging alone (SAGA full) gives a consistent speedup across all configurations. Combining merging with Atop-N attention adds overhead at short sequences but provides substantial gains at long contexts.

The proof of Lemma 4.1 is given in Sec. A. Let $G = (V, E, w)$ be the complete weighted graph with vertex set $V = \{1, \ldots, 2h_k^S\}$, where for each pair of teacher key heads $\{a, b\}$, E contains an edge with weight

$$w(a, b) := \mathcal{L}_{a,b}(\hat{\mathbf{W}}_K(a, b)) = \min_{\mathbf{M}} \mathcal{L}_{a,b}(\mathbf{M}). \tag{16}$$

where the second equality follows from Lemma 4.1.

Theorem 4.2. *A minimum-weight perfect matching of G yields an optimal solution to Equation* (14)*, with σ^* being a permutation whose consecutive pairs are the matched edges, and for each matched pair $(\sigma^*_{2i-1}, \sigma^*_{2i})$ the corresponding optimal merged key projection is*

$$\hat{\mathbf{W}}_K^i = \hat{\mathbf{W}}_K(\sigma^*_{2i-1}, \sigma^*_{2i}). \tag{17}$$

Proof. For a fixed permutation σ, we look at the inner objective in Eq. (14). This objective is a sum of h_k^S independent pairwise terms, one for each pair $(\sigma_{2i-1}, \sigma_{2i})$. Therefore

$$\min_{\mathbf{M}^{[1:h_k^S]}} \sum_{i=1}^{h_k^S} \mathcal{L}_{\sigma_{2i-1},\sigma_{2i}}(\mathbf{M}^i) = \sum_{i=1}^{h_k^S} \min_{\mathbf{M}^i} \mathcal{L}_{\sigma_{2i-1},\sigma_{2i}}(\mathbf{M}^i) = \sum_{i=1}^{h_k^S} w(\sigma_{2i-1}, \sigma_{2i}), \tag{18}$$

Moreover, for each i, the minimizing choice is $\hat{\mathbf{W}}_K^i = \hat{\mathbf{W}}_K(\sigma_{2i-1}, \sigma_{2i})$. Thus the value of Eq. (14) for a given permutation σ is exactly the total weight of the perfect matching encoded by its consecutive pairs. Any perfect matching of G can be written as such a permutation, and conversely any such permutation defines a perfect matching. Minimizing Eq. (14) over σ is therefore exactly the minimum-weight perfect matching problem on G. □

5 Experiments

5.1 Inference decoding speedups

We measure *end-to-end* per-token decode latency on Llama 3.2-1B [39] ($h_k : 8 \to 4$) and Qwen2.5-1.5B [6] ($h_k : 2 \to 1$) using compiled CUDA graphs. We compare the baseline model (original GQA head configuration) against the SAGA model (merged key heads), each evaluated with full attention and with Atop-N attention at $N \in \{512, 1024, 2048\}$. We sweep sequence lengths from 8K to 128K at different batch sizes, reporting mean per-token latency over 10 runs of 32 generated tokens each.

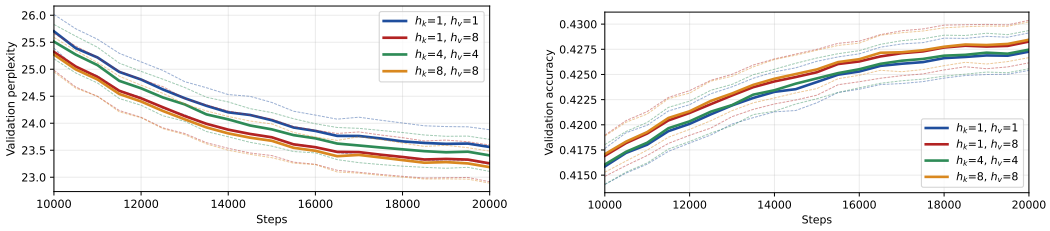


Figure 3: Training from scratch on SmolLM2-360M with $h_q = 16$ query heads. **Left:** validation perplexity (↓). **Right:** validation accuracy (↑). SAGA $(h_k, h_v) = (1, 8)$ closely tracks the high-capacity baseline $(8, 8)$ while outperforming both the matched-key-budget $(1, 1)$ baseline, and the parameter-matched baseline $(4, 4)$. Solid curves are means of 3 seeds; dotted lines indicate ± 1 std.

Fig. 2 presents end-to-end speedup over baseline full attention. To showcase different GPU architectures, Llama is evaluated on H200 while Qwen is evaluated on the newer B200 (complementary results in Sec. D). Merging key heads alone (SAGA full) provides a modest but consistent speedup across all sequence lengths and batch sizes, reflecting the reduced query-key cost from fewer key heads (see Sec. D for a kernel-level profile, larger models and additional ablations). This benefit grows with batch size as queries compete for memory bandwidth: on Llama at batch size 16, SAGA full attention is up to $1.47\times$ faster than baseline full at 128K context.

Combining merged heads with Atop-N attention introduces overhead at short sequences due to the threshold calibration and sparse gather, but yields substantial speedups at long contexts. At 128K with batch size 1, SAGA Atop-N achieves up to a $2.2\times$ end-to-end speedup on Qwen and $2.1\times$ on Llama over baseline full attention. Even at batch size 16, where the Atop-N overhead is larger, SAGA Atop-N reaches $2.2\times$ and $2.6\times$ speedups at 128K on Qwen and Llama respectively.

We note that flash attention [40] provides little benefit during autoregressive decode, where the attention computation is a matrix-vector product with no quadratic attention matrix to tile. Our full-attention baseline uses faster standard compiled attention, enabling portable support for the asymmetric GQA and ensuring a fair comparison when applying Atop-N attention.

To assess the quality impact of the Atop-N approximation, we evaluate all configurations on the full RULER benchmark [41] at context lengths of 8K, 16K, and 32K in Sec. 5.3 below. Fig. 4 reports per-task accuracy. On Qwen2.5-1.5B, Atop-N retains 83–97% of full attention quality across all context lengths, with most tasks virtually unaffected. The degradation is concentrated in multi-key retrieval tasks, which require the model to simultaneously attend to many dispersed keys – a setting where aggressive sparsity discards relevant entries.

5.2 Training from scratch

We train SAGA from scratch on the SmolLM2-360M [42] backbone, leaving the model unchanged except for the head counts. The number of query heads is fixed to $h_q = 16$, while h_k and h_v are varied. SAGA with $(h_k, h_v) = (1, 8)$ is compared against three GQA baselines: high-capacity with $(h_k, h_v) = (8, 8)$, low-capacity with $(h_k, h_v) = (1, 1)$ matching the key budget of SAGA, and parameter-matched with $(h_k, h_v) = (4, 4)$ roughly matching the parameter count of SAGA. This isolates the contribution of decoupling h_v from h_k. Each configuration is run with three seeds. Sec. D.2 shows that a larger-scale Qwen3-0.6B experiment trained on 50B tokens exhibits the same trend.

Fig. 3 shows that SAGA $(1, 8)$ closely matches the quality of the $(8, 8)$ baseline despite using a little over half the KV cache of $(8, 8)$. It also clearly outperforms the $(1, 1)$ baseline as well as the $(4, 4)$ baseline despite having a similar number of parameters. This result suggests that the improvement comes from allocating capacity asymmetrically. The same holds for how the key budget itself is spent: a Sigma-style variant [23] that reduces the key-head dimension instead of reducing the number of heads, $(h_k, d_k) = (2, 32)$, matches SAGA's $(1, 64)$ in KV-cache size and nearly in parameter count (both with $h_v = 8$, $d_v = 64$), yet reaches 41.7% final validation accuracy versus 42.7% for SAGA. This is consistent with the results of the SIGMA paper, who favored reducing the number of key heads while leaving the key-head dimension unchanged. Combined with Atop-N attention at

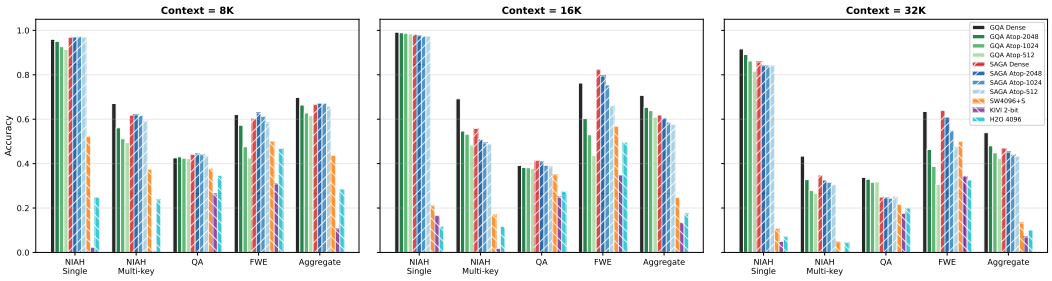


Figure 4: RULER accuracy for full attention and Atop-N attention at $N \in \{512, 1024, 2048\}$, across context lengths 8K, 16K, and 32K on Qwen2.5-1.5B, for different task categories. SAGA with Atop-N remains near baseline quality, while alternative eviction-based methods degrade substantially at longer contexts.

inference, this translates to the substantial decoding speedups reported in Sec. 5.1 at a small cost in quality. Our from-scratch experiments are necessarily limited in scale, and we hope future work will validate SAGA's benefits in a large-scale pretraining setting.

5.3 Application to pretrained models

To evaluate SAGA on models with realistic capabilities, we convert pretrained models to the SAGA architecture via knowledge distillation, avoiding the compute and data requirements of training a competitive LLM from scratch. We start from Qwen2.5-1.5B [6], a highly capable model with only $h_k = 2$ key heads, where the matching from Sec. 4 is trivial (a single pair) and reduces to averaging ($h_k : 2 \to 1$). We then follow up with Llama 3.2-1B [39], using the full matching-based initialization from Sec. 4 to merge 8 key heads into 4 (see Sec. B.1 for a comparison of merging strategies).

Ideally, a fair comparison would increase the student's capacity to compensate for the reduced key heads, for example by increasing the head dimension or adding value heads. However, as teacher and student representations diverge, a short distillation cannot effectively leverage the teacher's signal. For simplicity we keep the student architecture identical to the teacher except for the reduced h_k, and accept the slight quality degradation from this conservative approach.

The student is initialized by merging the teacher's key projection heads and trained with a combination of KL divergence from the teacher and hard-label cross-entropy. We train in several stages to expose the student to increasing context lengths while keeping total training short at only around 23K iterations (architecture dependent). We use a mix of SlimPajama [43], QA [44, 45] and long-context data [46, 47], as well as synthetic needle-in-a-haystack examples. We note that the student is heavily weighted to match the teacher's output probabilities on these sequences, so its retained performance reflects architectural capacity rather than task-specific adaptation. Full hyperparameters and data mix details are provided in Sec. C.3.

The distilled student retains 96.3% of the teacher's aggregate RULER score across 8K, 16K, and 32K context lengths, with per-task results shown in Fig. 4. A complementary Llama 3.2-1B comparison is shown in Sec. D. As shown in Tab. 1, the SAGA student remains close to the teacher on general-capability benchmarks despite the reduced key head capacity. For comparison, we evaluate alternative decode-acceleration methods: sliding window with attention sinks (SW+S) [30], H2O [29], and KIVI [48]. These methods fall below 50% aggregate RULER accuracy at 8K context and degrade rapidly at longer contexts, despite the 4096-token budget for sliding window and H2O. We note that SAGA could readily combine with KV cache quantization and eviction methods, such as those used for comparison above.

Furthermore, to experiment at a larger scale, we evaluate Llama3.1-70B-Instruct [39] with AtopN only. Since AtopN requires no retraining, these experiments remain feasible with limited computational resources. To highlight that the AtopN approximation holds over long generations, we evaluate on tasks that elicit extended outputs before a final answer: GPQA Main, limited to 8K output tokens, and GPQA Diamond, limited to 16K output tokens [49]. Both consist of graduate-level questions spanning multiple domains, however Diamond is a harder subset. In practice, the model produces long generations on these tasks, averaging over 1.2K output tokens on GPQA Main and 2.1K output

Table 1: General-capability benchmarks for the teacher and SAGA student models.

	Qwen2.5-1.5B ($h_k : 2 \to 1$)		Llama 3.2-1B ($h_k : 8 \to 4$)	
	Baseline	SAGA	Baseline	SAGA
HellaSwag (acc_norm ↑)	0.678	0.644	0.643	0.583
WikiText (word ppl ↓)	12.10	12.43	11.98	13.09

Table 2: Llama3.1-70B-Instruct accuracy on GPQA with AtopN for varying N. GPQA Main is limited to 8K output tokens and GPQA Diamond to 16K.

N	GPQA Diamond @16K (↑)	GPQA Main @8K (↑)
Baseline	0.4343	0.4241
2048	0.3939	0.4129
1024	0.3838	0.4330
512	0.3586	0.4241

tokens on GPQA Diamond. We experiment with $N \in \{2048, 1024, 512\}$ and find that GPQA Main accuracy is unharmed even at $N = 512$, whereas GPQA Diamond incurs some degradation. Results are available in Table 2

6 Limitations, discussion and conclusion

We introduced SAGA, an attention architecture designed around the computational trade-offs of sparse decoding. By decoupling key and value head counts, SAGA directly addresses the query-key bottleneck that dominates under sparse attention, while preserving model capacity through additional value heads. Our experiments show that SAGA nearly matches equivalent GQA models when trained from scratch and retains 96.3% of the teacher's aggregate RULER score when distilled from pretrained checkpoints, while delivering substantial inference end-to-end speedups of more than $2\times$ at long contexts.

Several limitations point to promising future directions. First, our current implementation relies on the simple Atop-N method; integrating hardware-aware sparse attention kernels could yield further speedup gains. Second, our distilled SAGA models maintain the same architecture as the teacher aside from the reduced key projections — future work could compensate for this slight capacity reduction by reallocating parameters, for example by increasing the value head count or dimension. Finally, our quality experiments are limited to models of up to 1.5B parameters; validating SAGA's benefits in a full-scale pretraining setting remains an important open question. More broadly, we believe that co-designing attention architectures with their inference-time sparsity patterns is a fertile direction, and hope that SAGA serves as a step toward making this a standard consideration in LLM design.

Acknowledgments and Disclosure of Funding

The research of DS was funded by the European Union (ERC, A-B-C-Deep, 101039436). Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency (ERCEA). Neither the European Union nor the granting authority can be held responsible for them. The research of DS was also supported by the Council for Higher Education in Israel under the Moonshot Project.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[5] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[6] Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

[7] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

[8] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[10] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. Efficient attention: Attention with linear complexities. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 3531–3539, 2021.

[11] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In *International conference on algorithmic learning theory*, pages 597–619. PMLR, 2023.

[12] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

[13] Michael Davies, Neal Crago, Karthikeyan Sankaralingam, and Christos Kozyrakis. Liminal: Exploring the frontiers of llm decode performance, 2025. URL `https://arxiv.org/abs/2507.14397`.

[14] 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. *arXiv preprint arXiv:2305.13245*, 2023.

[15] Rewon Child. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

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

[17] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33: 17283–17297, 2020.

[18] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020. URL `https://arxiv.org/abs/2001.04451`.

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

[20] Ankit Gupta, Guy Dar, Shaya Goodman, David Ciprut, and Jonathan Berant. Memory-efficient transformers via top-k attention. *arXiv preprint arXiv:2106.06899*, 2021.

[21] Pichao Wang, Xue Wang, Fan Wang, Ming Lin, Shuning Chang, Hao Li, and Rong Jin. Kvt: k-nn attention for boosting vision transformers. In *European conference on computer vision*, pages 285–302. Springer, 2022.

[22] Yilong Chen, Linhao Zhang, Junyuan Shang, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, and Yu Sun. DHA: Learning decoupled-head attention from transformer checkpoints via adaptive heads fusion. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[23] Zhenghao Lin, Zihao Tang, Xiao Liu, Yeyun Gong, Yi Chang, Jian Jia, Jian Li, Yelong Shen, Chen Lin, Nan Duan, and Weizhu Chen. Sigma: Differential rescaling of query, key and value for efficient language models, 2025. URL `https://arxiv.org/abs/2501.13629`.

[24] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[25] DeepSeek-AI. DeepSeek-V3.2-Exp: Boosting long-context efficiency with DeepSeek sparse attention. `https://github.com/deepseek-ai/DeepSeek-V3.2-Exp`, 2025.

[26] Jintao Zhang, Chendong Xiang, Haofeng Huang, Haocheng Xi, Jun Zhu, Jianfei Chen, et al. Spargeattention: Accurate and training-free sparse attention accelerating any model inference. In *Forty-second International Conference on Machine Learning*, 2025.

[27] Jiayi Yuan, Cameron Shinn, Kai Xu, Jingze Cui, George Klimiashvili, Guangxuan Xiao, Perkz Zheng, Bo Li, Yuxin Zhou, Zhouhai Ye, et al. Blasst: Dynamic blocked attention sparsity via softmax thresholding. *arXiv preprint arXiv:2512.12087*, 2025.

[28] Daniel Ohayon, Itay Lamprecht, Itay Hubara, Israel Cohen, Daniel Soudry, and Noam Elata. Block sparse flash attention. *arXiv preprint arXiv:2512.07011*, 2025.

[29] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

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

[31] 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. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

[32] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

[33] Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024.

[34] 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*, pages 31094–31116. PMLR, 2023.

[35] Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022.

[36] Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew Gormley. Unlimiformer: Long-range transformers with unlimited length input. *Advances in Neural Information Processing Systems*, 36:35522–35543, 2023.

[37] Chao Lou, Zixia Jia, Zilong Zheng, and Kewei Tu. Sparser is faster and less is more: Efficient sparse attention for long-range transformers. *arXiv preprint arXiv:2406.16747*, 2024.

[38] Themistoklis Haris. k nn attention demystified: A theoretical exploration for scalable transformers. *arXiv preprint arXiv:2411.04013*, 2024.

[39] Aaron Grattafiori, Abhimanyu Dubey, et al. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

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

[41] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[42] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Guilherme Penedo, Nouamane Tazi, Lewis Tunstall, et al. SmolLM2: When smol goes big – data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

[43] Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R. Steeves, Joel Hestness, and Nolan Dey. Slimpajama: A 627b token cleaned and deduplicated version of redpajama. `https://www.cerebras.ai/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama`, June 2023. URL `https://huggingface.co/datasets/cerebras/SlimPajama-627B`.

[44] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, 2018. URL `https://arxiv.org/abs/1809.09600`.

[45] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 784–789, 2018. URL `https://arxiv.org/abs/1806.03822`.

[46] Yushi Bai, Xin Lv, Jiajie Zhang, Yuze He, Ji Qi, Lei Hou, Jie Tang, Yuxiao Dong, and Juanzi Li. Longalign: A recipe for long context alignment of large language models, 2024. URL `https://arxiv.org/abs/2401.18058`.

[47] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models, 2024. URL `https://arxiv.org/abs/2309.12307`.

[48] 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 *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[49] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

[50] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

A Proof of Lemma 4.1

Recall that $\mathcal{Q}_r$ denotes the query heads that share $\mathbf{W}_K^r$. Define the realized query matrices $\mathbf{Q}^j := \mathbf{X}\mathbf{W}_Q^j$, and fix a pair of teacher key heads $a, b \in \{1, \dots, 2h_k^S\}$. Recall the pairwise loss

$$\begin{aligned}\mathcal{L}_{a,b}(\mathbf{M}) := &\sum_{j\in\mathcal{Q}_a} \left\|\mathbf{Q}^j(\mathbf{X}\mathbf{M})^\top - \mathbf{Q}^j(\mathbf{X}\mathbf{W}_K^a)^\top\right\|_{\mathrm{F}}^2 \\ &+ \sum_{j\in\mathcal{Q}_b} \left\|\mathbf{Q}^j(\mathbf{X}\mathbf{M})^\top - \mathbf{Q}^j(\mathbf{X}\mathbf{W}_K^b)^\top\right\|_{\mathrm{F}}^2 .\end{aligned} \tag{19}$$

Proof of Lemma 4.1. For any fixed $j \in \mathcal{Q}_a$, we have

$$\begin{aligned}&\left\|\mathbf{Q}^j(\mathbf{X}\mathbf{M})^\top - \mathbf{Q}^j(\mathbf{X}\mathbf{W}_K^a)^\top\right\|_{\mathrm{F}}^2 = \left\|\mathbf{Q}^j\big(\mathbf{X}(\mathbf{M}-\mathbf{W}_K^a)\big)^\top\right\|_{\mathrm{F}}^2 \\ &= \mathrm{tr}\Big(\mathbf{Q}^j\big(\mathbf{X}(\mathbf{M}-\mathbf{W}_K^a)\big)^\top\mathbf{X}(\mathbf{M}-\mathbf{W}_K^a)(\mathbf{Q}^j)^\top\Big) \\ &= \mathrm{tr}\big((\mathbf{M}-\mathbf{W}_K^a)^\top\mathbf{X}^\top\mathbf{X}(\mathbf{M}-\mathbf{W}_K^a)(\mathbf{Q}^j)^\top(\mathbf{Q}^j)\big),\end{aligned} \tag{20}$$

where we used the trace form of the Frobenius norm, $\|A\|_{\mathrm{F}}^2 = \mathrm{tr}(AA^\top)$, and the cyclic property of the trace. Denoting

$$\mathbf{S}_a := \sum_{j\in\mathcal{Q}_a}(\mathbf{Q}^j)^\top(\mathbf{Q}^j), \qquad \mathbf{S}_b := \sum_{j\in\mathcal{Q}_b}(\mathbf{Q}^j)^\top(\mathbf{Q}^j) \tag{21}$$

and summing over $j \in \mathcal{Q}_a$ gives

$$\sum_{j\in\mathcal{Q}_a}\left\|\mathbf{Q}^j(\mathbf{X}\mathbf{M})^\top - \mathbf{Q}^j(\mathbf{X}\mathbf{W}_K^a)^\top\right\|_{\mathrm{F}}^2 = \mathrm{tr}\big((\mathbf{M}-\mathbf{W}_K^a)^\top\mathbf{X}^\top\mathbf{X}(\mathbf{M}-\mathbf{W}_K^a)\mathbf{S}_a\big). \tag{22}$$

Applying the same argument to the second summation term, $\mathcal{L}_{a,b}(\mathbf{M})$ is equivalent to minimizing

$$\mathrm{tr}\big((\mathbf{M}-\mathbf{W}_K^a)^\top\mathbf{X}^\top\mathbf{X}(\mathbf{M}-\mathbf{W}_K^a)\mathbf{S}_a\big) + \mathrm{tr}\big((\mathbf{M}-\mathbf{W}_K^b)^\top\mathbf{X}^\top\mathbf{X}(\mathbf{M}-\mathbf{W}_K^b)\mathbf{S}_b\big). \tag{23}$$

This is a convex quadratic function of $\mathbf{M}$. We find the minimizer by taking the derivative with respect to $\mathbf{M}$. Using the identity $\frac{\partial}{\partial\mathbf{M}}\mathrm{tr}\big(\mathbf{M}^\top\mathbf{B}\mathbf{M}\mathbf{C}\big) = 2\mathbf{B}\mathbf{M}\mathbf{C}$ for symmetric positive semidefinite $\mathbf{B}$ and $\mathbf{C}$, and applying it with $\mathbf{M} \leftarrow \mathbf{M}-\mathbf{W}_K^a$, $\mathbf{B} \leftarrow \mathbf{X}^\top\mathbf{X}$, and $\mathbf{C} \leftarrow \mathbf{S}_a$ (and analogously for the second term), the gradient of the objective with respect to $\mathbf{M}$ is

$$2\mathbf{X}^\top\mathbf{X}(\mathbf{M}-\mathbf{W}_K^a)\mathbf{S}_a + 2\mathbf{X}^\top\mathbf{X}(\mathbf{M}-\mathbf{W}_K^b)\mathbf{S}_b. \tag{24}$$

Setting this to zero gives

$$\mathbf{X}^\top\mathbf{X}\Big(\mathbf{M}(\mathbf{S}_a+\mathbf{S}_b) - \mathbf{W}_K^a\mathbf{S}_a - \mathbf{W}_K^b\mathbf{S}_b\Big) = 0. \tag{25}$$

Hence any $\mathbf{M}$ satisfying

$$\mathbf{M}(\mathbf{S}_a+\mathbf{S}_b) = \mathbf{W}_K^a\mathbf{S}_a + \mathbf{W}_K^b\mathbf{S}_b \tag{26}$$

is optimal. One such solution is

$$\mathbf{M} = \big(\mathbf{W}_K^a\mathbf{S}_a + \mathbf{W}_K^b\mathbf{S}_b\big)(\mathbf{S}_a+\mathbf{S}_b)^\dagger . \tag{27}$$

Since $\mathbf{S}_a$ and $\mathbf{S}_b$ are positive semidefinite, the rows of $\mathbf{W}_K^a\mathbf{S}_a + \mathbf{W}_K^b\mathbf{S}_b$ lie in the row space of $\mathbf{S}_a + \mathbf{S}_b$, so this choice indeed satisfies the linear system above. □

B Induced permutations for query, value, and output projections

Once a matching has been chosen, let $\sigma = (\sigma_1, \dots, \sigma_{2h_k^S})$ be any permutation of $\{1, \dots, 2h_k^S\}$ whose consecutive pairs are exactly the matched pairs, i.e. $(\sigma_{2i-1}, \sigma_{2i})$ for $i = 1, \dots, h_k^S$. The purpose of the induced permutations is to re-index the teacher heads before we merge its two key heads so that each matched pair becomes consecutive.

We first permute the key and value heads by σ:

$$\widetilde{\mathbf{W}}_K := \big[\mathbf{W}_K^{\sigma_1} \ \cdots \ \mathbf{W}_K^{\sigma_{2h_k^S}}\big], \qquad \widetilde{\mathbf{W}}_V := \big[\mathbf{W}_V^{\sigma_1} \ \cdots \ \mathbf{W}_V^{\sigma_{2h_k^S}}\big]. \tag{28}$$

For the query heads, let

$$g := \frac{h_q}{h_k^T} = \frac{h_q}{2h_k^S}, \qquad B_r := \big((r-1)g+1, \ldots, rg\big), \quad r = 1, \ldots, 2h_k^S, \tag{29}$$

so B_r is the block of query heads that share the teacher key head r. The associated query-head permutation π_Q is the concatenation of these blocks into a single sequence of length h_q:

$$\pi_Q := \big(B_{\sigma_1} \| B_{\sigma_2} \| \cdots \| B_{\sigma_{2h_k^S}}\big) \in \{1, \ldots, h_q\}^{h_q}, \tag{30}$$

where $\|$ denotes concatenation, so that $\pi_Q(k)$ for $k = 1, \ldots, h_q$ refers to the k-th entry of this flattened sequence. Writing the query projection in head blocks,

$$\mathbf{W}_Q = \big[\mathbf{W}_Q^1 \ \cdots \ \mathbf{W}_Q^{h_q}\big], \tag{31}$$

the permuted query projection is

$$\widetilde{\mathbf{W}}_Q := \big[\mathbf{W}_Q^{\pi_Q(1)} \ \cdots \ \mathbf{W}_Q^{\pi_Q(h_q)}\big]. \tag{32}$$

For the output projection, write $\mathbf{W}_O$ as a vertical stack of query-head input blocks,

$$\mathbf{W}_O = \begin{bmatrix} \mathbf{W}_{O,1} \\ \vdots \\ \mathbf{W}_{O,h_q} \end{bmatrix}, \qquad \mathbf{W}_{O,r} \in \mathbb{R}^{d \times D}. \tag{33}$$

We permute these input blocks by the same query-head permutation:

$$\widetilde{\mathbf{W}}_O := \begin{bmatrix} \mathbf{W}_{O,\pi_Q(1)} \\ \vdots \\ \mathbf{W}_{O,\pi_Q(h_q)} \end{bmatrix}. \tag{34}$$

Before merging, this re-indexing leaves the attention module unchanged. After merging, the two value heads at positions $2i-1$ and $2i$ and their associated $2g$ query heads are exactly the heads that should share the merged SAGA key head i.

B.1 Merging strategies ablation

We compare three strategies for initializing the merged SAGA key projections on Llama 3.2-1B, measuring cross-entropy loss and per-token KL divergence from the teacher immediately after initialization (before distillation):

- **Sequential pairing**: merges consecutive head pairs in their original tensor order (effectively arbitrary pairing) by averaging their key projection matrices.
- **Key matching**: minimum-weight perfect matching with edge weight

  $$w(a, b) := \big\|\mathbf{X}\mathbf{W}_K^a - \mathbf{X}\mathbf{W}_K^b\big\|_{\mathrm{F}}^2, \tag{35}$$

  i.e. the same matching procedure as ours, but the graph weights are based only on the realized key matrices instead of attention logits. Here, the minimizer is the average $\frac{1}{2}(\mathbf{W}_K^a + \mathbf{W}_K^b)$.
- **Logit matching** (ours): the minimum-weight perfect matching on attention logit reconstruction error described in Sec. 4.1.

Logit-matching initialization consistently achieves the lowest reconstruction error across both reduction ratios, with the advantage growing as the reduction becomes more aggressive ($8 \to 2$). The two other merging strategies do not have a clear superior method. Sequential pairing is better at $8 \to 4$, and key matching at $8 \to 2$.

Table 3: Initialization quality before distillation (Llama 3.2-1B). Lower is better for both metrics.

Reduction	Method	CE	KL
$8 \to 4$	Sequential pairing	7.231	4.754
$8 \to 4$	Key matching	7.727	5.254
$8 \to 4$	Logit matching (ours)	**6.748**	**4.271**
$8 \to 2$	Sequential pairing	10.306	7.819
$8 \to 2$	Key matching	9.077	6.596
$8 \to 2$	Logit matching (ours)	**7.597**	**5.125**

C Implementation details

Code implementation of our experiments is included in `https://github.com/noamelata/SAGA`.

C.1 Atop-N attention

Our Atop-N attention is an efficient implementation of the algorithm described in Sec. 3.1. Specifically, it uses a pre-calibrated per-head threshold to collect all entries that exceed it, greatly improving runtimes. This comparison operation uses a custom portable Triton[2] kernel, replacing pytorch's CUDA-optimized `torch.topk` used for true top-N values.

During the first decode step after prefill, we compute exact attention probabilities via softmax and extract the N-th largest value per head using `torch.topk`. This value serves as the threshold for the subsequent n decode steps. Because the attention distribution changes slowly across consecutive decode steps, we use $n = 32$, thus amortizing the computational cost of `torch.topk` across 32 decode steps.

Alongside with the reduced probability-value multiplication, computing softmax only on the top-N values is a significant source of speedup. Yet, we find that thresholds in logit space tend to perform poorly. Instead, we make use of a probability-space threshold τ transformed into the logit space, a much cheaper but mathematically equivalent operation. Given raw attention logits $\mathbf{z}$, we approximate the max logit $\hat{m}$ from a small tail of positions and compute the log-sum-exp denominator $\hat{s} = \sum_i \exp(z_i - \hat{m})$. The logit-space threshold is then $\hat{m} + \log(\tau \cdot \hat{s})$, which requires only a single reduction pass over the full sequence rather than a full softmax.

The selected indices are used to gather the corresponding value vectors, after which the logit-space weights are normalized via softmax (the $-\infty$ padding receives zero probability) and multiplied with the gathered values via regular batched matrix multiplication (for batch < 16) or sparse batched matrix multiplication (otherwise) to produce the attention output.

Certain layers can be designated as exempt from Top-N selection and always use full attention. In our experiments, we exempt the first and 14th attention layers for Qwen2.5-1.5B[3] and the first layer for Llama 3.2-1B[4], as these layers tend to have more diffuse attention patterns.

The threshold comparison and index collection are implemented as a single Triton kernel. Each row of the attention weight matrix is partitioned into blocks (autotuned over 512, 1024, and 2048 elements). Within each block, elements exceeding the threshold are identified via a comparison mask, their positions computed via an intra-block cumulative sum, and results written to an output buffer using an atomic counter to coordinate across blocks. The kernel outputs both the selected values and their indices into a fixed-size buffer of capacity $2N$, with unused slots initialized to $-\infty$. If the number of entries exceeding the threshold exceeds $2N$, the atomic counter saturates and excess entries are silently dropped, preserving memory safety. The buffer is allocated once and reused across steps, ensuring compatibility with CUDA graph capture.

[2] `https://github.com/triton-lang/triton`, MIT license.

[3] `https://huggingface.co/Qwen/Qwen2.5-1.5B`, Apache 2.0 license.

[4] `https://huggingface.co/meta-llama/Llama-3.2-1B`, Llama 3.2 Community License.

C.2 Training SAGA from scratch

We use the SmolLM2-360M[5] backbone. All models are trained on SlimPajama[6] for 20K steps with batch size 256 and sequence length 1024, totaling ∼5.24B tokens. We use the AdamW optimizer [50] with $(\beta_1, \beta_2) = (0.9, 0.999)$ and weight decay 0.1 and a cosine learning rate schedule with 500 warmup steps, peak learning rate 3×10^{-4}, and minimum learning rate 7.5×10^{-5}. Training is performed in bfloat16, and each configuration is run with 3 seeds.

C.3 Distillation to SAGA

The student model is initialized from the teacher by merging the key projection heads according to the procedure described in Section 4.1. We then reorder the teacher's query, value, and output projection heads using the induced permutations described in Section B so that they remain aligned with the merged key heads, and copy all parameters unchanged from the teacher. The student is trained with a weighted combination of KL divergence from the teacher's output distribution (weight $\alpha = 0.7$, temperature $T = 3$) and hard-label cross-entropy (weight $1 - \alpha = 0.3$), using AdamW with fused implementation.

Training proceeds in three stages with increasing context length. In Stage 1, the student trains for 15K steps at context length 4096 with a learning rate of 2×10^{-4} and global batch size 64. The learning rate schedule follows a warmup-stable-decay (WSD) pattern: 200 warmup steps, 9800 stable steps, and 5000 decay steps. The data mix is 75% SlimPajama (packed), 15% QA data (HotpotQA[7] and SQuAD[8], packed), and 10% synthetic needle-in-a-haystack examples.

Stage 2 starts from the Stage 1 checkpoint with a fresh optimizer and trains for 5K steps at context length 20480 with a learning rate of 1×10^{-4} and global batch size 32. The WSD schedule uses 200 warmup, 3300 stable, and 1500 decay steps. The data mix shifts to 75% SlimPajama, 5% QA, 10% long-context documents (LongAlign-10k[9], LongAlpaca-12k[10], individually padded), and 10% synthetic NIAH.

Stage 3 continues from the Stage 2 checkpoint and trains at context length 36864 with a learning rate of 5×10^{-5} and global batch size 32. For Qwen2.5-1.5B: 2K steps, $\alpha = 0.7$, $T = 3$, WSD schedule with 100 warmup and 500 decay steps. For Llama 3.2-1B: 4K steps, $\alpha = 0.85$, $T = 4$, WSD schedule with 200 warmup and 800 decay steps. Stage 3 interleaves 20K-length microbatches (2 per gradient step) with the 36K-length data to prevent forgetting of shorter-range attention patterns. Training at 36864 tokens provides headroom over the 32K evaluation target. Total training time is approximately 60 hours on 8×H200 GPUs.

C.4 Evaluation

Quality is evaluated using the RULER benchmark[11] for long-context tasks, and HellaSwag[12] and WikiText[13] for general capabilities. All evaluations use the lm-eval-harness framework[14].

D Additional results

D.1 Qwen2.5-1.5B speedups on H200

Figure 2 reports Qwen2.5-1.5B latency measurements on B200. Fig. 5 shows the corresponding results on H200, confirming that SAGA's speedup gains are consistent across GPU generations.

[5] `https://huggingface.co/HuggingFaceTB/SmolLM2-360M`, Apache 2.0 license.
[6] `https://huggingface.co/datasets/cerebras/SlimPajama-627B`, Apache 2.0 license.
[7] `https://huggingface.co/datasets/hotpotqa/hotpot_qa`, CC BY-SA 4.0 license.
[8] `https://huggingface.co/datasets/rajpurkar/squad_v2`, CC BY-SA 4.0 license.
[9] `https://huggingface.co/datasets/THUDM/LongAlign-10k`
[10] `https://huggingface.co/datasets/Yukang/LongAlpaca-12k`
[11] `https://huggingface.co/datasets/THUDM/RULER`, Apache 2.0 license.
[12] `https://huggingface.co/datasets/Rowan/hellaswag`, MIT license.
[13] `https://huggingface.co/datasets/Salesforce/wikitext`, CC BY-SA 3.0 license.
[14] `https://github.com/EleutherAI/lm-evaluation-harness`, MIT license.

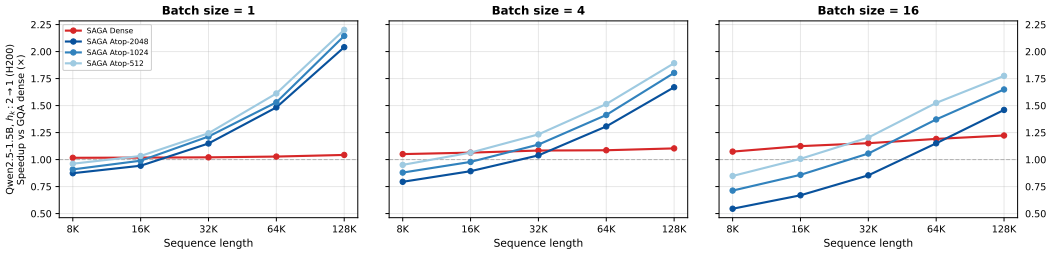


Figure 5: Speedup over baseline full attention on Qwen2.5-1.5B ($h_k : 2 \to 1$) measured on H200 at various batch sizes and sequence lengths.

D.2 Qwen3-0.6B trained from scratch on 50B tokens

We train a Qwen3-0.6B-based architecture from scratch on 50B tokens. This model is approximately $1.7\times$ larger than the SmolLM2-360M model evaluated in Sec. 5.2, and its training budget is $9.54\times$ larger. We run a single seed for each configuration due to computational limitations. Tab. 4 shows that SAGA with $(h_k, h_v) = (1, 8)$ is closer to the high-capacity $(8, 8)$ baseline than to the matched-key-budget $(1, 1)$ baseline on HellaSwag accuracy, WikiText perplexity, and validation accuracy. Thus, the same trend holds at this larger model and training scale.

Table 4: From-scratch results for a Qwen3-0.6B-based architecture trained on 50B tokens.

Configuration	HellaSwag (acc_norm ↑)	WikiText (word ppl ↓)	Validation accuracy (↑)
$(h_k, h_v) = (1, 1)$	0.3310 ± 0.0047	34.9129	0.440
$(h_k, h_v) = (1, 8)$	0.3343 ± 0.0047	33.8604	0.443
$(h_k, h_v) = (8, 8)$	0.3368 ± 0.0047	33.3382	0.444

D.3 Llama 3.2-1B RULER accuracy

Figure 6 presents the per-task RULER accuracy breakdown for Llama 3.2-1B ($h_k : 8 \to 4$), complementing the Qwen2.5-1.5B results in the main text. The SAGA student retains the majority of baseline quality, with degradation again concentrated in multi-key retrieval tasks.

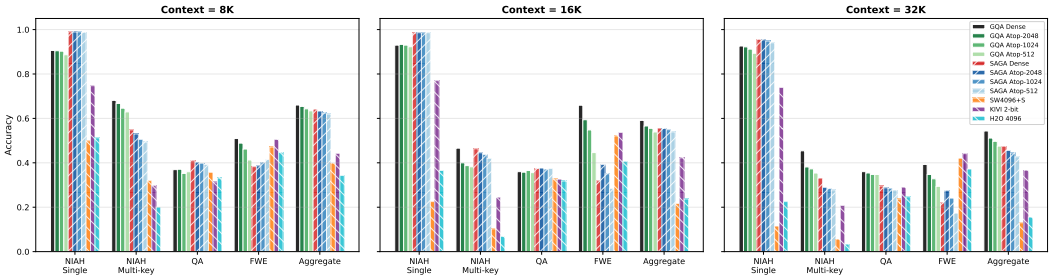


Figure 6: Per-task RULER accuracy for Llama 3.2-1B ($h_k : 8 \to 4$) with full attention and Atop-N attention at $N \in \{512, 1024, 2048\}$, across context lengths 8K, 16K, and 32K.

D.4 Kernel-level decode profile

Tab. 5 profiles one decode step of Llama 3.2-1B at 64K context and batch size 16 on an H100. Under full attention the query-key and probability-value multiplications cost a comparable amount (870 vs. 1091 μs). Merging key heads $8 \to 4$ halves the query-key term while leaving the probability-value term unchanged, and Atop-N reduces the probability-value term to 88 μs, leaving the query-key multiplication as the dominant remaining cost. The two components thus address the two bottlenecks of Fig. 1.

Table 5: Decode-step profile of Llama 3.2-1B ($h_k : 8 \to 4$) at 64K context, batch size 16, on H100. Kernel times are μs per non-exempt layer per decode step; the last rows give end-to-end per-token latency.

Kernel	GQA (full)	SAGA (full)	SAGA + Atop-N 1024
Query-key multiplication	870.1	466.0	446.0
Probability-value multiplication	1090.9	1091.1	87.5
Softmax / threshold	249.8	257.3	107.7
Gather	1.2	1.2	88.8
Q/K/V/O projections	22.8	23.3	22.8
MLP	42.7	43.0	43.3
Layer total	2277.6	1881.9	796.0
End-to-end (ms / token)	40.57	31.08	18.99
Speedup	1.00×	1.31×	2.14×

D.5 Decode latency at larger model scale

Decode latency depends on tensor shapes and kernels rather than on weight values, so the efficiency of SAGA can be measured on larger architectures without training them. Tab. 6 reports per-token decode latency of the Qwen2.5-32B architecture ($h_k : 8 \to 4$) with untrained weights at 128K context on H100. The speedup of SAGA with Atop-N persists at this scale and grows with batch size, as in the smaller models, though at a slower rate. Quality at this scale is not evaluated.

Table 6: Per-token decode latency (ms) of the Qwen2.5-32B architecture ($h_k : 8 \to 4$) at 128K context on H100.

	Batch 1	Batch 4	Batch 16
GQA + full attention	46.90	84.99	263.06
SAGA + Atop-N 1024	39.43	68.93	175.06
Speedup	1.19×	1.23×	1.50×

D.6 Separating the head configuration from the selector

Tab. 7 reports Qwen2.5-1.5B decode speedups for GQA and SAGA under full attention, Atop-N, and *exact* top-N, which recomputes the true top-N indices with `torch.topk` at every step. Comparing GQA with and without Atop-N isolates the selector; comparing SAGA against GQA under the same attention method isolates the head configuration. SAGA is faster under all three, e.g. 1.23× over GQA with Atop-N and 1.15× with exact top-N at batch size 4 and 128K, so the architectural gain does not depend on the threshold approximation of Atop-N. Exact top-N is slower than dense attention except at the longest contexts, which is what motivates Atop-N.

Table 7: Decode speedup of Qwen2.5-1.5B relative to GQA with full attention at the same batch size (H100), $N = 1024$.

Batch	Heads	Attention	8K	16K	32K	64K	128K
1	GQA	full	1.00	1.00	1.00	1.00	1.00
1	GQA	Atop-N	0.93	1.00	1.17	1.43	1.88
1	GQA	exact top-N	0.68	0.63	0.89	1.08	1.39
1	SAGA	full	1.00	1.02	1.05	1.05	1.04
1	SAGA	Atop-N	0.95	1.02	1.23	1.53	2.06
1	SAGA	exact top-N	0.68	0.64	0.92	1.14	1.48
4	GQA	full	1.00	1.00	1.00	1.00	1.00
4	GQA	Atop-N	0.85	0.93	1.06	1.24	1.51
4	GQA	exact top-N	0.67	0.73	0.82	0.96	1.10
4	SAGA	full	1.05	1.05	1.08	1.11	1.14
4	SAGA	Atop-N	0.89	0.98	1.16	1.41	1.86
4	SAGA	exact top-N	0.69	0.75	0.87	1.05	1.27

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The paper claims, as described in the abstract, are reflected by the method and experiment sections.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in the limitation section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Theoretical results include complete proofs as well as assumptions.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Full implementation details are provided in the paper and supplementary material. Data used for evaluation and training is publicly available.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code implementation of our experiment is available in the supplementary material.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See implementation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: For the speedup experiments, error bars are not reported because the std is small enough to be ignored while making the graphs harder to follow. For the pretraining section, error bars are provided and defined.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide these implementation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This research conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The paper is on the topic of LLM architecture and efficiency. It has no societal impacts beyond the general in use of LLM technology.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Datasets, models and code is properly cited and credited. Terms of used are properly respected.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Documentation is provided.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The paper is on the topic of LLM architecture and efficency, and thus uses LLMs.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.