SCALABLE ATTENTION: THE INFERENCE COMPUTE-CACHE SCALING LAW

Anonymous authors
Paper under double-blind review

ABSTRACT

We present Scalable Attention (SA), which scales attention compute without enlarging the KV cache, adding marginal parameters. Based on SA, we propose the compute-cache scaling law, which models loss as separate power laws in compute and cache. Given unit prefill and decode costs, set by prefix caching, workload, speculative decoding, and hardware, the law returns the optimal compute and cache for any target loss. At the 3.3B scale, SA with twice the attention compute outperforms GQA in validation loss while using only 25% of its KV cache.

1 INTRODUCTION

Advancements in modern hardware and large-scale clusters have made it possible to train increasingly massive language models. Scaling parameters and compute (Kaplan et al., 2020; Hoffmann et al., 2022) remains the most reliable path to improving model intelligence. Recent scaling laws target pretraining, which involves only the prefill stage. They identify the architecture that minimizes pretraining loss under a given FLOPs budget. Inference, however, has two stages with different bottlenecks. Prefill is compute-bound and determined by model FLOPs, whereas decode is memory-bound and dominated by moving key-value (KV) cache and parameters from high-bandwidth memory to on-chip memory. Pretraining scaling laws therefore predict the optimal design under a prefill FLOPs budget but do not account for decode cost.

We build Scalable Attention (SA) on Grouped-Query Attention (GQA) (Ainslie et al., 2023), where h_q query heads attend to h_{kv} KV heads, each serving as both key and value. Each query head is projected into s_a sub-heads, where s_a is the scaling attention factor. Within a group, all $s_a \times (h_q/h_{kv})$ sub-heads, derived from its h_q/h_{kv} query heads, attend to the same KV head. Each computes attention scores against this KV head and takes a weighted sum over it, giving $s_a \times (h_q/h_{kv})$ attention outputs per group. A separate learned value projection is then applied to each output, and the s_a outputs of each query head are summed to form its final output. This design scales attention compute by raising the effective head count while adding marginal attention parameters.

Building on SA, we propose the first inference attention law, the compute-cache scaling law. Following Hoffmann et al. (2022), we measure attention expressivity by the validation loss, where a lower loss indicates higher expressivity, and model it as

$$\mathcal{L}(C, M) \;=\; E + \frac{A}{C^{\alpha}} + \frac{B}{M^{\beta}}, \qquad (1)$$

where C denotes attention compute, M denotes KV cache, and E is the loss neither resource can remove. The two remaining terms are the excess loss caused by limited compute and by limited cache, with compute determining the prefill cost and cache determining the decode cost.

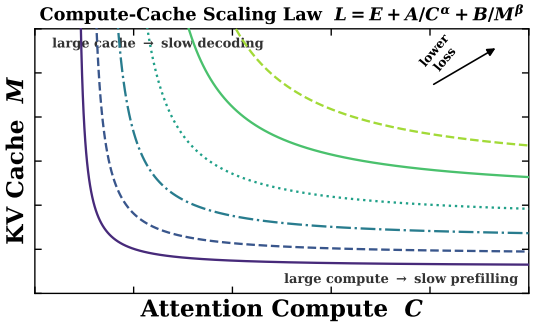


Figure 1: Iso-loss curves.

For a target loss, the feasible configurations form an iso-loss curve, shown in Figure 1. All configurations on a given curve attain the same loss, so compute and cache can be traded against each other. Moving along the curve toward larger M reduces compute, while moving toward larger C does the reverse, and curves farther from the origin correspond to lower loss. The trade is not unlimited. Each

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.

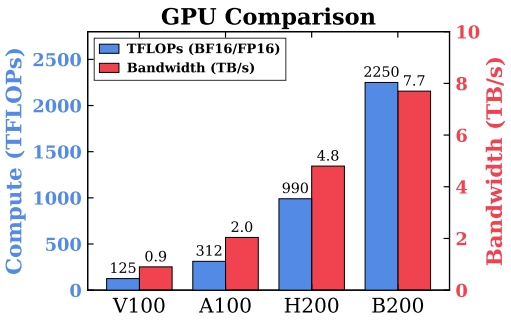


Figure 2: Peak BF16/FP16 compute throughput and HBM memory bandwidth across four NVIDIA GPU generations (V100, A100, H200, B200). From V100 to B200, peak compute has grown by 18× while memory bandwidth has only grown by 8.6×.

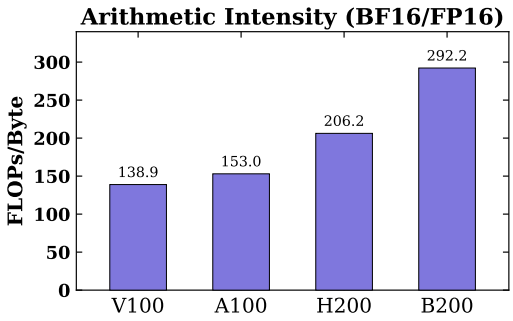


Figure 3: Arithmetic intensity ceiling of each GPU, defined as the ratio of peak compute to peak memory bandwidth. The ridge point rises from 139 on V100 to 292 on B200, so memory-bound operators such as decode attention leave more compute idle on newer hardware.

curve approaches an asymptote in both directions, so every loss level demands a minimum amount of each resource that no quantity of the other can replace. The two extremes also differ in latency. A configuration with large M incurs slow decoding, since decoding is memory-bound, whereas a configuration with large C incurs slow prefill, since prefill is compute-bound.

Since serving time is linear in each resource, we write it as $\tau_{\text{prefill}}\, C + \tau_{\text{decode}}\, M$, where τ_{prefill} is the prefill time per unit of attention compute and τ_{decode} is the decode time per unit of KV cache. Both are set by the deployment rather than by the model, and either can be measured on any running configuration by dividing its prefill or decode time by its C or M. For a target loss $\bar{\mathcal{L}}$, the optimal configuration minimizes this time subject to $\mathcal{L}(C, M) = \bar{\mathcal{L}}$, and the Lagrangian conditions give

$$\frac{\tau_{\text{prefill}}\,(C^\star)^{1+\alpha}}{\alpha A} = \frac{\tau_{\text{decode}}\,(M^\star)^{1+\beta}}{\beta B}, \qquad \frac{A}{(C^\star)^\alpha} + \frac{B}{(M^\star)^\beta} = \bar{\mathcal{L}} - E. \tag{2}$$

Since $AC^{-\alpha} + BM^{-\beta}$ is strictly convex, the feasible region $\{\mathcal{L}(C, M) \le \bar{\mathcal{L}}\}$ is strictly convex, so for any $\bar{\mathcal{L}} > E$ the pair $(C^\star, M^\star)$ is unique. A deployment with costly prefill therefore favors less compute and more cache, and one with costly decode favors the opposite.

2 Preliminary

2.1 Modern Hardware

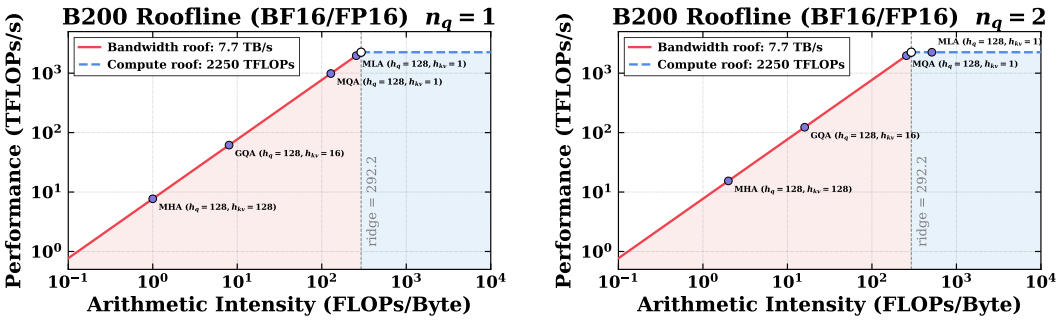
The evolution of modern computing hardware shows an increasing gap between computational power and memory bandwidth, as seen in the shift from NVIDIA V100 to B200. During this period, tensor core peak compute has expanded by 18×, whereas HBM bandwidth has only grown by 8.6× (Figure 2). As GPU peak compute outpaces memory bandwidth growth, the arithmetic intensity ceiling (the ratio of the two) continues to rise (Figure 3), which makes arithmetic intensity the quantity an architecture has to be designed against.

2.2 Decode

The two phases of inference sit on opposite sides of this ceiling. Prefill is typically compute-bound, while decode is heavily memory-bound. To optimize for modern hardware, we must increase compute and reduce data movement during decoding to raise the overall arithmetic intensity. Following Zadouri et al. (2025), the arithmetic intensity of the decode phase can be quantified as

$$\text{Arithmetic Intensity} \approx \frac{h_q}{h_{kv}} \cdot n_q \cdot \alpha_{kv} \quad (n_{kv} \gg n_q), \tag{3}$$

where h_q and h_{kv} denote the number of query and KV heads, respectively; n_q represents the number of query tokens processed per decode step, n_{kv} the number of cached positions they attend to; and α_{kv} equals 2 when keys and values are tied into one cached tensor and 1 otherwise.



(a) Standard autoregressive decode ($n_q = 1$).

(b) Speculative decode ($n_q = 2$).

Figure 4: Arithmetic intensity of attention variants on the B200 roofline. We evaluate the variants with $h_q = 128$, setting $h_{kv} = 128$ for MHA, $h_{kv} = 16$ for GQA (the standard $h_q/h_{kv} = 8$ ratio), and $h_{kv} = 1$ for MQA and MLA. Under standard autoregressive decode ($n_q = 1$, left), all variants are memory-bound, where throughput is bottlenecked by HBM bandwidth rather than peak FLOPs. Under speculative decode ($n_q = 2$, right), the arithmetic intensity doubles as the computational load per KV cache access scales linearly with n_q. This shift pushes MLA across the ridge into the compute-bound region, while other variants remain memory-bound.

At $n_q = 1$, Multi-Head Attention (MHA) (Vaswani et al., 2017), Grouped-Query Attention (GQA), Multi-Query Attention (MQA) (Shazeer, 2019), and Multi-Head Latent Attention (MLA) (DeepSeek et al., 2024) reach arithmetic intensities of 1, 8, 128, and 256 (Figure 4a), all below the B200 ridge of 292, so decoding is bound by HBM bandwidth and leaves most peak FLOPs idle while the hardware waits on KV cache reads for each token. Speculative decoding (Leviathan et al., 2023) amortizes each KV load over n_q queries, and at $n_q = 2$ these values double to 2, 16, 256, and 512 (Figure 4b). Only MLA crosses the ridge, but crossing is not free, since decode latency is then set by FLOPs rather than bandwidth. The useful target is therefore an arithmetic intensity that approaches the ridge without exceeding it.

3 Scalable Attention

3.1 Methodology

For a sequence of n tokens with hidden states $\boldsymbol{H} \in \mathbb{R}^{n \times d}$, SA first derives the query heads $\boldsymbol{Q}_i \in \mathbb{R}^{n \times d_h}$ and KV heads $\boldsymbol{C}_j \in \mathbb{R}^{n \times d_h}$ by per-head projection,

$$\boldsymbol{Q}_i = \boldsymbol{H}\boldsymbol{W}_i^{\mathrm{Q}}, \qquad \boldsymbol{C}_j = \boldsymbol{H}\boldsymbol{W}_j^{\mathrm{KV}}, \tag{4}$$

where $i \in \{1, \ldots, h_q\}$, $j \in \{1, \ldots, h_{kv}\}$, and $\boldsymbol{W}_i^{\mathrm{Q}}, \boldsymbol{W}_j^{\mathrm{KV}} \in \mathbb{R}^{d \times d_h}$. Following GQA, each $\boldsymbol{C}_j$ is shared by h_q/h_{kv} query heads via the mapping $g(i) = \lceil i \cdot h_{kv}/h_q \rceil$. To scale attention compute by a factor of s_a, a natural approach is to apply s_a independent linear projections to each query and key head, indexed by $m \in \{1, \ldots, s_a\}$, defining $\boldsymbol{Q}_{i,m} = \boldsymbol{Q}_i \boldsymbol{W}_{i,m}^{\mathrm{SQ}}$ and $\boldsymbol{K}_{i,m} = \boldsymbol{C}_{g(i)} \boldsymbol{W}_{i,m}^{\mathrm{SK}}$. $\boldsymbol{W}_{i,m}^{\mathrm{SK}}$ can be absorbed into the query side, since the resulting attention logits expand as

$$\boldsymbol{Q}_{i,m}\boldsymbol{K}_{i,m}^{\top} \;=\; \boldsymbol{Q}_i \, \boldsymbol{W}_{i,m}^{\mathrm{SQ}} \big(\boldsymbol{W}_{i,m}^{\mathrm{SK}}\big)^{\top} \boldsymbol{C}_{g(i)}^{\top}. \tag{5}$$

Both $\boldsymbol{W}_{i,m}^{\mathrm{SQ}}$ and $(\boldsymbol{W}_{i,m}^{\mathrm{SK}})^{\top}$ lie in $\mathbb{R}^{d_h \times d_h}$, so their product is itself a single matrix of the same shape. We therefore parameterize this product directly as $\boldsymbol{W}_{i,m}^{\mathrm{SQ}}$ from the outset, eliminating $\boldsymbol{W}_{i,m}^{\mathrm{SK}}$ entirely.

Building on the merged query linear projection, we now describe the SA forward pass in detail. The query head $\boldsymbol{Q}_i$ is first normalized by a row-wise RMSNorm (Zhang & Sennrich, 2019) with a learnable per-channel scale $\gamma^{\mathrm{Q}} \in \mathbb{R}^{d_h}$ before the projection,

$$\bar{\boldsymbol{Q}}_i \;=\; \gamma^{\mathrm{Q}} \odot \boldsymbol{Q}_i \Big/ \sqrt{\tfrac{1}{d_h}\|\boldsymbol{Q}_i\|^2 + \epsilon}, \tag{6}$$

and then mapped through $\boldsymbol{W}_{i,m}^{\mathrm{SQ}}$ to yield the sub-head query $\tilde{\boldsymbol{Q}}_{i,m} = \bar{\boldsymbol{Q}}_i \boldsymbol{W}_{i,m}^{\mathrm{SQ}}$. Following Henry et al. (2020), we apply QKNorm before computing attention scores. The sub-head query is rescaled

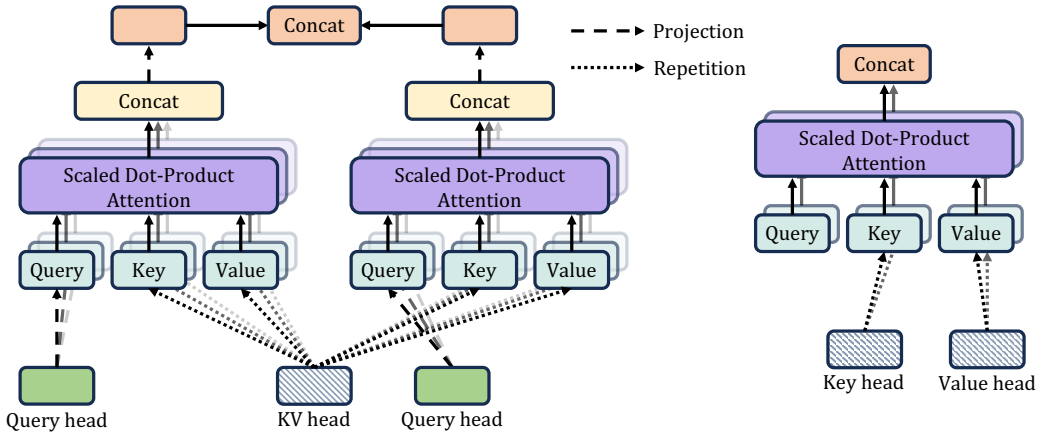


Figure 5: Comparison between SA (left) and GQA (right). While GQA shares a single key-value head pair across h_q/h_{kv} query heads by replication, SA introduces a scaling factor s_a on top of this grouping mechanism. Each query head is independently projected into s_a sub-heads, while the KV head serves as both the key and the value in attention. The s_a attention outputs of each query head are then concatenated and passed through a learned value projection, allowing SA to increase attention compute across s_a subspaces without materializing key or value heads.

by a parameter-free row-wise RMS normalization (DeepSeek et al., 2026), and the KV head $\boldsymbol{C}_{g(i)}$ is normalized by a row-wise RMSNorm with learnable scale $\gamma^{\mathrm{KV}} \in \mathbb{R}^{d_h}$,

$$\hat{\boldsymbol{Q}}_{i,m} = \tilde{\boldsymbol{Q}}_{i,m} \Big/ \sqrt{\tfrac{1}{d_h}\|\tilde{\boldsymbol{Q}}_{i,m}\|^2 + \epsilon}, \qquad \bar{\boldsymbol{C}}_{g(i)} = \gamma^{\mathrm{KV}} \odot \boldsymbol{C}_{g(i)} \Big/ \sqrt{\tfrac{1}{d_h}\|\boldsymbol{C}_{g(i)}\|^2 + \epsilon}. \tag{7}$$

Positional information is then injected by partial RoPE (Su et al., 2024), where the rotation at token position t is given by the block-diagonal matrix $\boldsymbol{R}_t \in \mathbb{R}^{d_h \times d_h}$, with an identity block on the first $d_h - d_r$ dimensions and a rotation block on the last d_r dimensions. At each token position t, both $\hat{\boldsymbol{Q}}_{i,m}$ and $\bar{\boldsymbol{C}}_{g(i)}$ are rotated by $\boldsymbol{R}_t$. For each sub-head query (i, m), the attention weight from query position t to source position s is then given by the causal softmax

$$a_{i,m,t,s} = \frac{\exp(\ell_{i,m,t,s})}{\sum_{s' \le t} \exp(\ell_{i,m,t,s'})}, \qquad \ell_{i,m,t,s} = \frac{1}{\sqrt{d_h}}\, \hat{\boldsymbol{Q}}_{i,m,t} \boldsymbol{R}_t \left(\bar{\boldsymbol{C}}_{g(i),s} \boldsymbol{R}_s\right)^{\top}, \tag{8}$$

and the output at position t is the weighted sum of the rotated KV head across source positions,

$$\boldsymbol{O}_{i,m,t} = \sum_{s \le t} a_{i,m,t,s}\, \bar{\boldsymbol{C}}_{g(i),s} \boldsymbol{R}_s. \tag{9}$$

The rotated KV head $\bar{\boldsymbol{C}}_{g(i),s}\boldsymbol{R}_s$ is cached during inference and serves as both the key and the value in attention, so its source position rotation $\boldsymbol{R}_s$ enters $\boldsymbol{O}_{i,m,t}$ through value-side weighted sum. Applying an inverse RoPE $\boldsymbol{R}_t^{-1}$ (Harik & Jayasiri, 2022; Azabou et al., 2023; Miyato et al., 2024) on the right gives

$$\bar{\boldsymbol{O}}_{i,m,t} = \boldsymbol{O}_{i,m,t}\boldsymbol{R}_t^{-1} = \sum_{s \le t} a_{i,m,t,s}\, \bar{\boldsymbol{C}}_{g(i),s}\, \boldsymbol{R}_{s-t}, \tag{10}$$

where $\boldsymbol{R}_s\boldsymbol{R}_t^{-1} = \boldsymbol{R}_{s-t}$ depends only on the relative offset $s - t$, ensuring translation invariance. Finally, the s_a attention outputs for each query head i are concatenated and projected by a learned value projection,

$$\tilde{\boldsymbol{O}}_{i,t} = \mathrm{Concat}\big(\bar{\boldsymbol{O}}_{i,1,t}, \ldots, \bar{\boldsymbol{O}}_{i,s_a,t}\big)\, \boldsymbol{W}_i^{\mathrm{SV}}, \tag{11}$$

where $\boldsymbol{W}_i^{\mathrm{SV}}$ is the vertical concatenation of $\{\boldsymbol{W}_{i,m}^{\mathrm{SV}}\}_{m=1}^{s_a}$, with each $\boldsymbol{W}_{i,m}^{\mathrm{SV}} \in \mathbb{R}^{d_h \times d_h}$. The per-head outputs are then concatenated across all h_q query heads and passed through the output projection,

$$\boldsymbol{O}_t = \mathrm{Concat}\Big(\tilde{\boldsymbol{O}}_{1,t}, \ldots, \tilde{\boldsymbol{O}}_{h_q,t}\Big)\, \boldsymbol{W}^{\mathrm{O}}, \tag{12}$$

where $\boldsymbol{W}^{\mathrm{O}} \in \mathbb{R}^{h_q d_h \times d}$ and $\boldsymbol{O}_t \in \mathbb{R}^d$ is the attention layer output at position t. This completes the SA forward pass, contrasted with GQA in Figure 5. We next show why projection scales attention compute while GQA's repetition does not, quantify SA's parameter overhead, identify what determines the unit costs, and give the resulting design space.

3.2 Projection Instead of Repetition

We first motivate why SA derives its sub-heads by projection rather than repetition. Suppose each query head $\boldsymbol{Q}_i$ were replicated s_a times, so all sub-heads attend to $\bar{\boldsymbol{C}}_{g(i)}$ with the same query and produce the same attention output $\bar{\boldsymbol{O}}_{i,t}$. The value projection then gives

$$\tilde{\boldsymbol{O}}_{i,t} = \mathrm{Concat}\big(\underbrace{\bar{\boldsymbol{O}}_{i,t}, \ldots, \bar{\boldsymbol{O}}_{i,t}}_{s_a}\big)\boldsymbol{W}_i^{\mathrm{SV}} = \bar{\boldsymbol{O}}_{i,t}\sum_{m=1}^{s_a}\boldsymbol{W}_{i,m}^{\mathrm{SV}}, \tag{13}$$

which collapses to a single head with the reparameterized projection $\sum_m \boldsymbol{W}_{i,m}^{\mathrm{SV}}$. Repetition therefore adds no representational capacity, even with distinct value projections, and merely incurs an $s_a\times$ overhead in attention compute. SA avoids this degeneracy by giving each sub-head its own query projection $\boldsymbol{W}_{i,m}^{\mathrm{SQ}}$, so that the s_a attention distributions $a_{i,m,t,s}$ differ and their outputs span s_a distinct subspaces. Given distinct attention distributions, the value projections $\boldsymbol{W}_{i,m}^{\mathrm{SV}}$ then combine the s_a outputs into a single head.

3.3 Attention Parameter Overhead

A natural concern with scalable attention is whether increasing the scaling factor s_a inflates the attention parameter budget. We show that the growth is negligible. Counting the trainable parameters in a single SA layer gives

$$P_{\mathrm{SA}}(s_a) = \underbrace{d\cdot h_q d_h}_{\boldsymbol{W}^{\mathrm{Q}}} + \underbrace{2d\cdot h_{kv} d_h}_{\boldsymbol{W}^{\mathrm{KV}}} + \underbrace{2h_q d_h^2\cdot s_a}_{\boldsymbol{W}^{\mathrm{SQ}},\,\boldsymbol{W}^{\mathrm{SV}}} + \underbrace{h_q d_h\cdot d}_{\boldsymbol{W}^{\mathrm{O}}}. \tag{14}$$

Among these terms, only $\boldsymbol{W}^{\mathrm{SQ}}$ and $\boldsymbol{W}^{\mathrm{SV}}$ depend on s_a, and they do so linearly. Every other term is independent of s_a. Crucially, these s_a-dependent terms scale with the head dimension d_h rather than the hidden size d. Since $d_h \ll d$ in standard configurations, the parameter overhead introduced by s_a is marginal. For comparison, a standard GQA layer contains $d\cdot h_q d_h + 2d\cdot h_{kv} d_h + h_q d_h\cdot d$ parameters. To compare the two under a common attention compute budget, we adopt the Kimi K2 (Kimi et al., 2025) configuration ($d = 7168$, $L = 61$, $d_h = 128$, $h_q = 64$, and $h_{kv} = 8$) and use s_a as an axis. For SA, s_a is the scaling factor introduced earlier, while for GQA we scale the query heads to $h_q^{\mathrm{GQA}} = 64 s_a$ and hold h_{kv} fixed, so that both reach the same attention compute. Figure 6 plots the resulting attention parameter counts across $s_a \in \{1, 2, 4, 8, 16\}$.

At the baseline ($s_a = 1$), SA and GQA use almost identical parameter budgets, 8.19B and 8.06B, respectively. As s_a increases, however, the two parameter trajectories diverge sharply. GQA's parameter count scales almost linearly, reaching 15.22B, 29.55B, 58.21B, and 115.52B at $s_a = 2, 4, 8, 16$, respectively. SA grows only marginally over the same range, reaching 8.32B, 8.57B, 9.08B, and 10.11B. Concretely, a $16\times$ increase in attention compute adds only 1.9B parameters to SA ($\sim 1.23\times$ its baseline), whereas the same increase adds 107.5B to GQA ($\sim 14.3\times$ its baseline). This confirms that SA scales attention compute at a marginal parameter cost, since the projection overhead, $2h_q d_h^2$ per unit of s_a, grows far more slowly than GQA's query and output projections, which scale linearly in the much larger hidden size d instead of within a single head. As a result, SA preserves HBM capacity for the KV cache and MoE experts rather than spending it on attention parameters. This is what makes C a free variable in the inference scaling law, since scaling attention compute no longer competes with the rest of the model for memory.

3.4 What Determines the Unit Costs

Once the unit costs τ_{prefill} and τ_{decode} are known, the compute-cache scaling law (Section 1) returns the optimal configuration $(C^\star, M^\star)$ for any target loss. Both costs are set by how a model is served rather than by the model itself, and we identify four factors that shape them.

The first factor is the prefix cache hit rate (Zhu et al., 2026). Serving systems reuse the KV cache of shared prefixes across requests, which skips the KV projections for the cached tokens and the attention among them. The uncached tokens still attend over the full context, so prefill cost falls with the hit rate. A deployment with a large persistent cache, shared system prompts, and multi-turn traffic therefore has a smaller τ_{prefill}, while a cold cache with no cross-request sharing requires a full prefill on every request and keeps τ_{prefill} high. This factor is a property of the serving infrastructure.

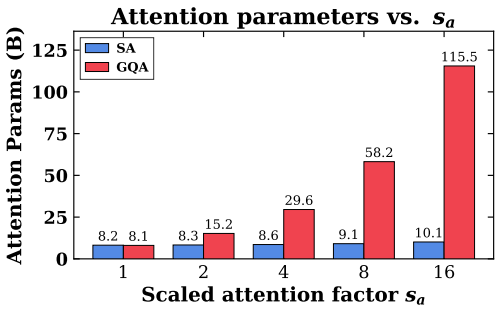


Figure 6: Attention parameter count vs. scaling factor s_a. GQA's parameters grow almost linearly with s_a, reaching 115.5B at $s_a = 16$, while SA stays nearly flat at 10.1B.

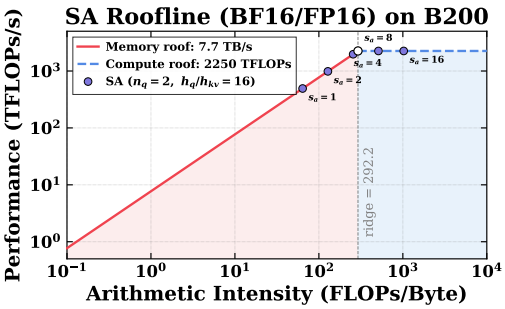


Figure 7: Arithmetic intensity of SA on the B200 roofline. Increasing s_a raises arithmetic intensity linearly; at $s_a = 8$, SA crosses the ridge into the compute-bound region.

The second factor is the workload shape, namely the ratio of prompt length to output length. Workloads with short prompts and long outputs, such as reasoning (Wei et al., 2022), spend most of their time in decoding and raise τ_{decode} relative to τ_{prefill}. Workloads with long prompts and short outputs, such as agentic serving (Yuan et al., 2026), are prefill-dominated and do the reverse.

The third factor is decode acceleration. Speculative decoding verifies several draft tokens in one forward pass, reducing the number of memory-bound decode steps per generated token. This lowers τ_{decode} while leaving prefill unchanged.

The fourth factor is the hardware. Prefill is compute-bound, so τ_{prefill} falls with peak FLOPs, whereas decode is memory-bound, so τ_{decode} falls with HBM bandwidth. Since peak compute has grown faster than bandwidth across GPU generations (Figure 2), newer hardware lowers τ_{prefill} relative to τ_{decode} and shifts the optimum toward more compute and less cache.

3.5 Design Space

Eq. (2) gives the optimal configuration $(C^\star, M^\star)$ for a target loss, and this section turns it into a concrete choice of s_a. Three practical constraints narrow the space. First, the GQA group size in modern LLMs is typically $h_q/h_{kv} \in \{8, 16\}$. Second, speculative decoding, now standard in LLM serving, verifies $n_q = 2$ draft tokens per step in most deployments, so a single KV read serves n_q query tokens and τ_{decode} falls accordingly. Third, our design ties keys and values by default, contributing a factor $\alpha_{kv} = 2$. Combining these with the scaling factor s_a, the decoding arithmetic intensity of SA follows from Eq. (3) as

$$\text{AI}_{\text{SA}} \approx \frac{h_q}{h_{kv}} \cdot n_q \cdot s_a \cdot \alpha_{kv} \qquad (n_{kv} \gg n_q). \tag{15}$$

The group size fixes M through h_{kv}, and n_q and α_{kv} are fixed by the serving stack and our design, leaving s_a, which sets C, as the free parameter. Speculative decoding acts on both sides, raising arithmetic intensity through n_q and lowering τ_{decode}, so a larger n_q leaves less room for s_a.

The hardware supplies an upper bound. The ridge point of modern accelerators continues to climb, with the B200 already at 292 FLOPs/Byte (Figure 3). Raising s_a increases AI_{SA}, and once it exceeds the ridge, decoding tips into the compute-bound region and slows down (Figure 7), so s_a can grow only until AI_{SA} reaches it. On the B200, with $n_q = 2$ and $\alpha_{kv} = 2$, the largest usable value is $s_a = 4$ for $h_q/h_{kv} = 16$ and $s_a = 8$ for $h_q/h_{kv} = 8$, both reaching AI $= 256$ just under the ridge. In general, the bound is $s_a \le \text{ridge} \cdot h_{kv}/(h_q\, n_q\, \alpha_{kv})$, so any accelerator and group size can be plugged in directly.

Within this bound, the unit costs select the operating point. A deployment with costly decode sits close to the largest s_a the hardware allows, since decoding has spare compute at every step and s_a converts it into expressivity at no extra cache, whereas a deployment with costly prefill settles for a smaller s_a, since attention compute is already the binding cost. The bound itself loosens as hardware advances. Since the ridge has risen across recent GPU generations while group sizes have stayed fixed, s_a should grow correspondingly so that AI_{SA} continues to track the hardware.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Baseline Selection. Several attention mechanisms are rarely used in current LLMs. MHA stores a full per-head KV cache, which is prohibitive for long context. MLA and Grouped Latent Attention (GLA) (Zadouri et al., 2025) handle incremental prefill poorly, since neither of their two computation paths is efficient. The absorbed form inflates prefill FLOPs, while the unabsorbed form materializes full keys and values and forfeits the memory savings that motivated MLA. DeepSeek-V4 (DeepSeek et al., 2026) no longer adopts MLA. Tensor Product Attention (TPA) (Zhang et al., 2025) and Simulated Attention Score (SAS) (Zheng et al., 2025) also scale attention compute, but must materialize the scaled KV cache in full at every decoding step, which gives up the same savings and blocks reuse of existing decoding kernels. SA keeps the expansion on the query side instead, so the KV cache stays at its base size and decoding proceeds as in GQA. We therefore compare against GQA (Appendix B.2) and attempt DeepSeek-V4's MQA (Shazeer, 2019) (Appendix B.1).

Model Configuration. Our experiments are built on the Llama-3 (Llama et al., 2024) architecture. All models share a common backbone of $L = 28$ layers, hidden size $d = 3072$, and $h_q = 24$ query heads. Where applicable, the per-head dimension is $d_h = 128$ and partial RoPE is applied to the last $d_r = 64$ dimensions. For GQA, we use $h_{kv} = 8$ with separate K and V heads and apply RoPE only to Q and K. For SA, we sweep $s_a \in \{1, 2, 4, 8\}$ over $h_{kv} \in \{2, 4, 8\}$ and train eleven of the twelve combinations, omitting $s_a = 1$ at $h_{kv} = 2$. Under Eq. (15) with $n_q = 2$ and $\alpha_{kv} = 2$, all but $s_a = 8$ at $h_{kv} = 2$ keep the decoding arithmetic intensity below the B200 ridge point of 292 (Figure 3). For DeepSeek-V4's MQA baseline, we use $h_{kv} = 1$, per-head dimension $d_h = 512$, query LoRA (Hu et al., 2022) rank $r_q = 1024$, and $G = 4$ value head groups. Following Zadouri et al. (2025), the FFN intermediate dimension of every configuration is retuned so that the total parameter count matches a fixed target. Complete architectural hyperparameters are deferred to Appendix C. All models are implemented on top of the `nanoGPT` codebase (Karpathy, 2022).

Pretraining Configuration. Each model is pretrained from scratch at the 3.3B scale on FineWeb-Edu-100B (Penedo et al., 2024), with 98.3B tokens for training and 0.1B held out for validation, using the GPT-2 tokenizer (50,257 tokens). We optimize with AdamW (Loshchilov & Hutter, 2019), setting $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.1, and gradient clipping at 1.0. The learning rate warms up linearly over 500 steps to 3.0×10^{-4}, then follows a cosine schedule (Loshchilov & Hutter, 2017) down to 10% of that peak. Training runs for 50,000 steps on 8 NVIDIA H100 GPUs, with a context length of 4,096 and a global batch of 480 sequences (≈2.0M tokens per step).

Evaluation Benchmark. Following pretraining scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022), we use the validation loss on FineWeb-Edu as the measure of attention expressivity and fit the inference compute-cache scaling law to it. We additionally report perplexity on six out-of-domain corpora, namely Wikipedia, C4 (Raffel et al., 2020), the Pile (Gao et al., 2020), RefinedWeb (Penedo et al., 2023), Cosmopedia (Ben Allal et al., 2024), and FineWeb (Penedo et al., 2024), using 0.1B tokens per corpus. For downstream evaluation, we report zero-shot accuracy on seven common-sense reasoning benchmarks via `lm-evaluation-harness` (Gao et al., 2024), namely ARC-Easy (ARC-E) (Clark et al., 2018), ARC-Challenge (ARC-C), OpenBookQA (Mihaylov et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), Winogrande (Sakaguchi et al., 2021), and PIQA (Bisk et al., 2020). We report length-normalized accuracy for ARC-E/C, OpenBookQA, HellaSwag, and PIQA, and standard accuracy for the remaining tasks.

4.2 EXPERIMENT ANALYSIS

Our DeepSeek-V4 MQA baseline diverged to NaN. Unlike DeepSeek-V4, which pairs MQA with sparse attention, we use full attention and could not obtain a stable run, so we omit MQA. Table 4 in Appendix D reports perplexity and downstream accuracy for all SA configurations and GQA.

4.2.1 THE INFERENCE COMPUTE-CACHE SCALING LAW

We fit Eq. (1) to the eleven SA configurations, taking $C = s_a$ and $M = h_{kv}$ so that C counts sub-heads per query head and M counts cached heads per layer. The five parameters are estimated

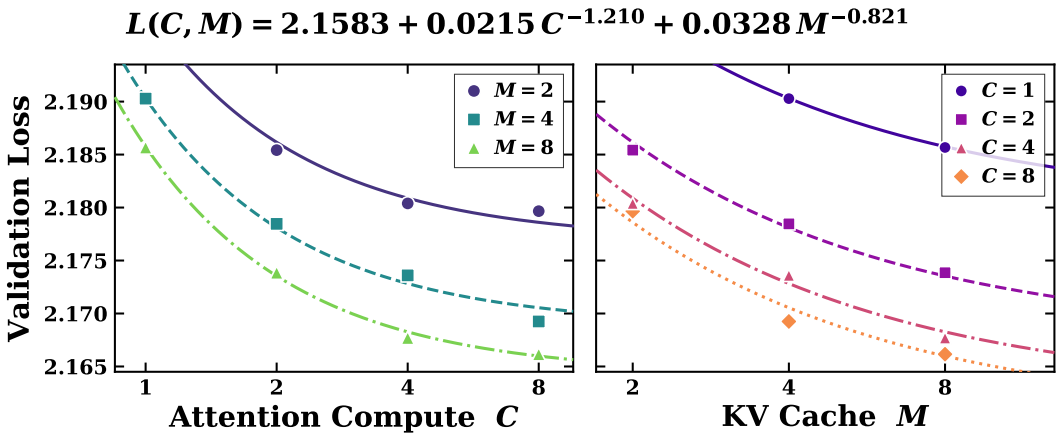


Figure 8: The compute-cache scaling law fitted to the eleven SA configurations, with $C = s_a$ and $M = h_{kv}$. Markers are measured validation losses and curves are Eq. (16). Left, loss against compute at each cache level. Right, loss against cache at each compute level.

by nonlinear least squares from a grid of initializations, giving

$$\mathcal{L}(C, M) \;=\; 2.1583 \;+\; 0.0215 C^{-1.210} \;+\; 0.0328 M^{-0.821}. \tag{16}$$

Figure 8 plots the two slices of the fit, loss against compute at each cache level and against cache at each compute level, with the error of every configuration deferred to Table 5 in Appendix D. All eleven markers sit close to their curves, with a largest error of 0.00125 and root mean squared error 0.00066, about 3% of the 0.024 spread across configurations.

The two resources act independently. In Eq. (16), compute and cache enter as separate terms, so adding compute lowers the loss by the same amount no matter how much cache is present. The curves in each panel of Figure 8 are therefore the same shape shifted up or down. The data largely agree. Doubling s_a from 1 to 2 lowers the loss by 0.0118 at both $h_{kv} = 4$ and $h_{kv} = 8$, close to the predicted 0.0122, and doubling h_{kv} from 4 to 8 lowers it by 0.0046 at both $s_a = 1$ and $s_a = 2$, matching the predicted 0.0046. The match is looser at the largest configurations. Doubling s_a from 4 to 8 lowers the loss by 0.0007, 0.0044, and 0.0015 at $h_{kv} = 2$, 4, and 8, against a predicted 0.0023 for all three. These gains show no consistent trend with h_{kv}, so they do not suggest that the benefit of compute depends on cache, and every configuration stays within 0.0013 of the fit.

Compute saturates faster than cache. The curves in the left panel flatten faster than those in the right, reflecting $\alpha = 1.210$ against $\beta = 0.821$. Each time C doubles, the loss drop is 0.43 of the previous one, whereas each time M doubles, it is still 0.57 of the previous one. Neither resource is therefore always more important. Differentiating Eq. (16) gives how much loss each removes when it grows by a small fraction,

$$-\frac{\partial \mathcal{L}}{\partial \ln C} = \alpha A\, C^{-\alpha}, \qquad -\frac{\partial \mathcal{L}}{\partial \ln M} = \beta B\, M^{-\beta}, \tag{17}$$

and setting the two equal gives the break-even curve $C^{\alpha} = (\alpha A/\beta B)\, M^{\beta}$, or $C^{1.210} = 0.97\, M^{0.821}$ for the fitted values. With less compute than this curve requires, adding compute helps more, and with more, adding cache helps more. At $h_{kv} = 4$ the break-even point is $s_a = 2.5$, and the data confirm the switch. From $s_a = 1$, doubling compute lowers the loss by 0.0118 while doubling cache lowers it by only 0.0046. From $s_a = h_{kv} = 4$ the order flips, with doubling cache worth 0.0059 against 0.0044 for compute, and the law predicts that further doublings of compute give only 0.0010 and 0.0004 against 0.0026 and 0.0015 for cache.

The optimum drifts toward cache. Because compute runs out sooner, the best split between compute and cache changes as we aim for lower loss. Solving the first condition in Section 1 for $M^{\star}$ gives the optimal path

$$\frac{M^{\star}}{C^{\star}} \;=\; \left(\frac{\beta B\, \tau_{\text{prefill}}}{\alpha A\, \tau_{\text{decode}}}\right)^{\frac{1}{1+\beta}} (C^{\star})^{\frac{\alpha-\beta}{1+\beta}}, \tag{18}$$

where the unit costs enter only the prefactor and the exponent, 0.21 for the fitted values, depends on α and β alone. The cache-to-compute ratio therefore grows with scale at a rate that no property of the deployment can change. With equal unit costs, the optimum is $(C^\star, M^\star) = (1.6, 1.8)$ at a loss of 2.190 and $(4.1, 5.7)$ at 2.170, so $M^\star/C^\star$ grows from 1.13 to 1.38, and limited cache causes a growing share of the loss above E, from 63% to 67%. A ratio that is optimal at one loss level is therefore not optimal at another. The unit costs shift the optimum in a different way, scaling the prefactor and sliding it along a fixed iso-loss curve. At the loss of the $C = M = 4$ configuration, setting $\tau_{\text{prefill}}/\tau_{\text{decode}}$ to 0.25, 1, and 4 moves the optimum to $(5.2, 3.5)$, $(3.4, 4.5)$, and $(2.4, 6.4)$, so $M^\star/C^\star$ ranges from 0.68 to 2.63. Cheaper prefill favors more compute, and cheaper decode favors more cache. Target loss and serving cost both change the best ratio, and the law separates them. The trade has limits, since each target loss requires a minimum of both resources that the other cannot replace, $C \geq 1.38$ and $M \geq 2.70$ at the same target loss. Halving compute to $C = 2$ requires raising M from 4 to 9.4, whereas doubling it to $C = 8$ saves only 0.9 units of cache, which is why the iso-loss curves in Figure 1 bend sharply.

4.2.2 SA-2 OUTPERFORMS GQA WITH ONLY 25% OF ITS KV CACHE AT 2× THE ATTENTION FLOPS

We next compare SA against standard GQA to test whether scaling attention compute can buy a smaller KV cache without losing quality. All models are matched in parameter count. GQA uses $h_{kv} = 8$ with separate key and value heads, so it caches $16\,d_h$ values per token per layer. SA-2 with $h_{kv} = 4$ ties keys and values and caches only $4\,d_h$, 25% of GQA's cache, while its $s_a = 2$ sub-heads double the FLOPs of attention scores and aggregation. Under this setting, SA-2 reaches a validation loss of 2.1785 against 2.1809 for GQA. The advantage holds across the design space. SA with $h_{kv} = 2$ and $s_a = 4$ matches GQA (2.1804 against 2.1809) with 12.5% of its cache, and SA with $h_{kv} = 4$ and $s_a = 4$ lowers the loss to 2.1736 with 25% of it. Scaling attention compute through learned projections thus trades prefill FLOPs for a much smaller KV cache and a lower loss.

4.3 DECODING LATENCY AGAINST THE ROOFLINE

We build SA's decoding kernel for NVIDIA Blackwell GPUs on top of TokenSpeed (TokenSpeed Team, 2026) and benchmark its latency on B200. The configuration uses $h_q = 16$ and $h_{kv} = 1$ with $n_q = 2$ for speculative decoding, and we sweep the scaling factor $s_a \in \{1, 2, 4, 8\}$. We evaluate a standard scenario with context lengths of 2K, 4K, 8K, and 16K at batch size 128, and a long-context scenario with 256K, 512K, 1M, and 2M at batch size 1. The arithmetic intensity of SA's decoding attention is $\text{AI} = 64\,s_a$. B200 has a BF16 ridge point of 292, so SA is firmly memory-bound for $s_a \leq 2$ ($\text{AI} \leq 128$), approaches the ridge at $s_a = 4$ ($\text{AI} = 256$), and becomes compute-bound at $s_a = 8$ ($\text{AI} = 512$). As shown in Figure 9, measured latency follows this prediction. It stays within 5% of $s_a = 1$ for $s_a = 2$, rises by 20–30% at $s_a = 4$, and grows by a further 1.5–2.0× at $s_a = 8$. Moderate scaling factors thus incur little decoding latency cost on B200.

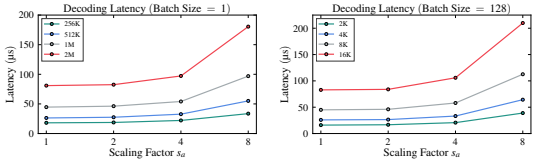


Figure 9: Decoding latency on B200 across scaling factors s_a, in the long-context scenario (left, batch size 1, 256K–2M) and the standard scenario (right, batch size 128, 2K–16K). Latency stays nearly flat for $s_a \leq 2$, rises moderately at $s_a = 4$, and grows sharply at $s_a = 8$ once compute-bound.

5 CONCLUSION

We present Scalable Attention, which scales attention compute via lightweight projections at base KV cache size and only marginal parameter cost. We propose the compute-cache scaling law, showing that compute saturates faster than cache, so the optimum shifts toward cache as target loss falls and toward compute as prefill cheapens, which lets serving costs guide attention design. At the 3.3B scale, the law fits eleven configurations, and SA with twice the attention compute outperforms GQA while caching only 25% as much. Our B200 kernel confirms decoding latency stays nearly flat up to twice the attention compute. Since the ridge point keeps rising across GPU generations, the optimal scaling factor should grow with it, making SA increasingly attractive on future hardware.

AI USE STATEMENT

We use LLMs to polish the writing and reorganize tables and figures.

ETHICS STATEMENT

This work proposes an efficiency-oriented attention mechanism and does not involve human subjects or the collection of new datasets. Experiments use established corpora and benchmarks under their licenses.

REPRODUCIBILITY STATEMENT

Our implementation builds on public GitHub codebases. Upon acceptance, we will release the full code and datasets in a public GitHub repository.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Empirical Methods in Natural Language Processing*, 2023.

Mehdi Azabou, Vinam Arora, Venkataramana Ganesh, Ximeng Mao, Santosh B Nachimuthu, Michael Jacob Mendelson, Blake Aaron Richards, Matthew G Perich, Guillaume Lajoie, and Eva L Dyer. A unified, scalable framework for neural population decoding. In *Advances in Neural Information Processing Systems*, 2023.

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Cosmopedia. `https://huggingface.co/datasets/HuggingFaceTB/cosmopedia`, 2024.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2020.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *North American Association for Computational Linguistics*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

DeepSeek et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

DeepSeek et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2024.

Georges Harik and Varuna Jayasiri. Rotary positional embeddings with relative distances, 2022. URL `http://research.labml.ai/RoPER.html`.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. *arXiv preprint arXiv:2010.04245*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, 2022.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Andrej Karpathy. Nanogpt. `https://github.com/karpathy/nanoGPT`, 2022.

Kimi et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, 2023.

Songtao Liu, Hongwu Peng, Zhiwei Zhang, Zhengyu Chen, and Yue Guo. Multi-head low-rank attention. In *International Conference on Learning Representations*, 2026.

Llama et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Empirical Methods in Natural Language Processing*, 2018.

Takeru Miyato, Bernhard Jaeger, Max Welling, and Andreas Geiger. GTA: A geometry-aware attention mechanism for multi-view transformers. In *International Conference on Learning Representations*, 2024.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon LLM: Outperforming curated corpora with web data only. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 2021.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

TokenSpeed Team. TokenSpeed: A speed-of-light LLM inference engine. https://github.com/lightseekorg/tokenspeed, 2026.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Yichao Yuan, Ankita Nayak, Souvik Kundu, and Nishil Talati. Agentic ai workload characteristics. *arXiv preprint arXiv:2605.26297*, 2026.

Ted Zadouri, Hubert Strauss, and Tri Dao. Hardware-efficient attention for fast decoding. In *Conference on Language Modeling*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Association for Computational Linguistics*, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019.

Yifan Zhang, Yifeng Liu, Huizhuo Yuan, Zhen Qin, Yang Yuan, Quanquan Gu, and Andrew Chi-Chih Yao. Tensor product attention is all you need. In *Advances in Neural Information Processing Systems*, 2025.

Chuanyang Zheng, Jiankai Sun, Yihang Gao, Yuehao Wang, Peihao Wang, Jing Xiong, Liliang Ren, Hao Cheng, Janardhan Kulkarni, Yelong Shen, Atlas Wang, Mac Schwager, Anderson Schneider, Xiaodong Liu, and Jianfeng Gao. Sas: Simulated attention score. In *Advances in Neural Information Processing Systems*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

Kan Zhu, Mathew Jacob, Chenxi Ma, Yi Pan, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. Tracelab: Characterizing coding agent workloads for llm serving. *arXiv preprint arXiv:2606.30560*, 2026.

A Related Work

The pursuit of decoding efficiency has driven a series of changes to the attention mechanism (Shazeer, 2019; Zhang et al., 2025; Zadouri et al., 2025). Multi-Query Attention (MQA) (Shazeer, 2019) shares a single key and value head across all query heads, which shrinks the KV cache and the bandwidth each decode step must consume, but the compression is aggressive enough to cost model quality. Grouped-Query Attention (GQA) (Ainslie et al., 2023) softens this by giving each group of query heads its own KV head, recovering much of the lost expressivity while keeping most of the efficiency.

Multi-Head Latent Attention (MLA) (DeepSeek et al., 2024) pushes further by compressing the KV cache into a single latent vector. During decoding the up-projection matrices are absorbed into the queries, so full keys and values are never materialized, giving an MQA-style computation over a more expressive latent state. The single latent vector, however, cannot be sharded across tensor-parallel (TP) ranks, so every rank must hold the entire cache and the per-rank memory saving that TP normally provides is lost. Inference frameworks such as SGLang (Zheng et al., 2024) therefore fall back on data parallelism for MLA decoding.

Scaling attention capacity without paying for it elsewhere remains difficult. Simulated Attention Score (SAS) (Zheng et al., 2025) and Multi-Head Low-Rank Attention (MLRA) (Liu et al., 2026) both expand attention capacity at small parameter cost, yet each runs into a different efficiency wall. SAS relies on convolutional operators that cannot be absorbed analytically at decode time, so the expanded heads must be written out in full, and the traffic between off-chip High-Bandwidth Memory (HBM) and on-chip SRAM grows accordingly.

MLRA instead splits the single latent head into several independent branches, which restores shardability but multiplies communication. Each branch produces its own partial output that must be summed across TP ranks, so a four-branch configuration issues four All-Reduce calls per attention layer where GQA issues one, and the volume scales with the branch count. Adding branches to scale further would also enlarge the KV cache, since each branch keeps its own latent. Neither line of work therefore delivers capacity that scales without a matching cost in memory traffic or communication, which is the gap SA is designed to close.

B Attention Mechanism

B.1 DeepSeek-V4's MQA

We follow the MQA variant from DeepSeek-V4 (DeepSeek et al., 2026), which differs from SA in three respects. First, $h_{kv} = 1$, so a single shared head $\boldsymbol{C} = \boldsymbol{H}\boldsymbol{W}^{\mathrm{KV}} \in \mathbb{R}^{n \times d_h}$ serves all h_q query heads and, as in SA, acts as both key and value.

Second, the query is first down-projected into a single latent head $\boldsymbol{C}_q \in \mathbb{R}^{n \times r_q}$ and then up-projected into the h_q query heads,

$$\boldsymbol{C}_q \;=\; \boldsymbol{H}\boldsymbol{W}^{\mathrm{DQ}}, \qquad \bar{\boldsymbol{C}}_q \;=\; \gamma^{\mathrm{DQ}} \odot \boldsymbol{C}_q \big/ \sqrt{\tfrac{1}{r_q}\|\boldsymbol{C}_q\|^2 + \epsilon}, \qquad \boldsymbol{Q} \;=\; \bar{\boldsymbol{C}}_q\, \boldsymbol{W}^{\mathrm{UQ}}, \tag{19}$$

with $\boldsymbol{W}^{\mathrm{DQ}} \in \mathbb{R}^{d \times r_q}$, $\boldsymbol{W}^{\mathrm{UQ}} \in \mathbb{R}^{r_q \times h_q d_h}$, and learnable scale $\gamma^{\mathrm{DQ}} \in \mathbb{R}^{r_q}$ applied to the latent head. The reshaped query heads $\boldsymbol{Q}_i \in \mathbb{R}^{n \times d_h}$ are then rescaled by a parameter-free row-wise RMS normalization, and $\boldsymbol{C}$ is normalized with a learnable scale $\gamma^{\mathrm{KV}} \in \mathbb{R}^{d_h}$,

$$\hat{\boldsymbol{Q}}_i \;=\; \boldsymbol{Q}_i \big/ \sqrt{\tfrac{1}{d_h}\|\boldsymbol{Q}_i\|^2 + \epsilon}, \qquad \bar{\boldsymbol{C}} \;=\; \gamma^{\mathrm{KV}} \odot \boldsymbol{C} \big/ \sqrt{\tfrac{1}{d_h}\|\boldsymbol{C}\|^2 + \epsilon}. \tag{20}$$

As in SA, partial RoPE on the last d_r dimensions is applied to $\hat{\boldsymbol{Q}}_i$ and $\bar{\boldsymbol{C}}$, and the attention output is multiplied by $\boldsymbol{R}_t^{-1}$ on the right,

$$\bar{\boldsymbol{O}}_{i,t} = \sum_{s \le t} a_{i,t,s}\, \bar{\boldsymbol{C}}_s\, \boldsymbol{R}_{s-t}, \qquad \ell_{i,t,s} = \tfrac{1}{\sqrt{d_h}}\, \hat{\boldsymbol{Q}}_{i,t} \boldsymbol{R}_t \big(\bar{\boldsymbol{C}}_s \boldsymbol{R}_s\big)^{\top}, \tag{21}$$

where $a_{i,t,s}$ is the causal softmax of $\ell_{i,t,s}$ over $s \le t$.

Third, the value up-projection is grouped rather than per head. The h_q heads are partitioned into G groups of h_q/G heads, with group $g \in \{1, \dots, G\}$ containing heads $\mathcal{G}_g = \{(g-1)h_q/G +$

$1, \ldots, gh_q/G\}$. Each group has its own projection $\boldsymbol{W}_g^{\mathrm{UV}} \in \mathbb{R}^{(h_q d_h/G)\times(d/G)}$, and the outputs of group g are concatenated and projected by it,

$$\tilde{\boldsymbol{O}}_{g,t} \;=\; \mathrm{Concat}\big(\bar{\boldsymbol{O}}_{i,t}\big)_{i\in\mathcal{G}_g}\, \boldsymbol{W}_g^{\mathrm{UV}} \;\in\; \mathbb{R}^{d/G}. \tag{22}$$

The G group outputs are then concatenated and projected by $\boldsymbol{W}^{\mathrm{O}} \in \mathbb{R}^{d\times d}$,

$$\boldsymbol{O}_t \;=\; \mathrm{Concat}\big(\tilde{\boldsymbol{O}}_{1,t}, \ldots, \tilde{\boldsymbol{O}}_{G,t}\big)\, \boldsymbol{W}^{\mathrm{O}} \;\in\; \mathbb{R}^{d}. \tag{23}$$

B.2 GQA

For a sequence of n tokens with hidden states $\boldsymbol{H} \in \mathbb{R}^{n\times d}$, GQA (Ainslie et al., 2023) derives query heads $\boldsymbol{Q}_i$ and separate key and value heads $\boldsymbol{K}_j, \boldsymbol{V}_j \in \mathbb{R}^{n\times d_h}$ by per-head projection,

$$\boldsymbol{Q}_i = \boldsymbol{H}\boldsymbol{W}_i^{\mathrm{Q}}, \qquad \boldsymbol{K}_j = \boldsymbol{H}\boldsymbol{W}_j^{\mathrm{K}}, \qquad \boldsymbol{V}_j = \boldsymbol{H}\boldsymbol{W}_j^{\mathrm{V}}, \tag{24}$$

where $i \in \{1, \ldots, h_q\}$, $j \in \{1, \ldots, h_{kv}\}$, and $\boldsymbol{W}_i^{\mathrm{Q}}, \boldsymbol{W}_j^{\mathrm{K}}, \boldsymbol{W}_j^{\mathrm{V}} \in \mathbb{R}^{d\times d_h}$. Each pair $(\boldsymbol{K}_j, \boldsymbol{V}_j)$ is shared by h_q/h_{kv} query heads under the same grouping $g(i)$ as in SA. Following Henry et al. (2020), we apply QKNorm by row-wise RMSNorm with learnable scales $\gamma^{\mathrm{Q}}, \gamma^{\mathrm{K}} \in \mathbb{R}^{d_h}$,

$$\bar{\boldsymbol{Q}}_i = \gamma^{\mathrm{Q}} \odot \boldsymbol{Q}_i \big/ \sqrt{\tfrac{1}{d_h}\|\boldsymbol{Q}_i\|^2 + \epsilon}, \qquad \bar{\boldsymbol{K}}_j = \gamma^{\mathrm{K}} \odot \boldsymbol{K}_j \big/ \sqrt{\tfrac{1}{d_h}\|\boldsymbol{K}_j\|^2 + \epsilon}, \tag{25}$$

while values are left unnormalized. Partial RoPE on the last d_r dimensions is applied to $\bar{\boldsymbol{Q}}_i$ and $\bar{\boldsymbol{K}}_j$ but not to $\boldsymbol{V}_j$. The attention weight from query position t to source position s for head i is the causal softmax

$$a_{i,t,s} \;=\; \frac{\exp(\ell_{i,t,s})}{\sum_{s'\le t}\exp(\ell_{i,t,s'})}, \qquad \ell_{i,t,s} \;=\; \tfrac{1}{\sqrt{d_h}}\, \bar{\boldsymbol{Q}}_{i,t}\boldsymbol{R}_t\big(\bar{\boldsymbol{K}}_{g(i),s}\boldsymbol{R}_s\big)^{\top}. \tag{26}$$

Because the value carries no rotation, the output requires no inverse rotation,

$$\boldsymbol{O}_{i,t} \;=\; \sum_{s\le t} a_{i,t,s}\, \boldsymbol{V}_{g(i),s}. \tag{27}$$

The per-head outputs are concatenated and projected by $\boldsymbol{W}^{\mathrm{O}} \in \mathbb{R}^{(h_q d_h)\times d}$,

$$\boldsymbol{O}_t \;=\; \mathrm{Concat}\big(\boldsymbol{O}_{1,t}, \ldots, \boldsymbol{O}_{h_q,t}\big)\, \boldsymbol{W}^{\mathrm{O}}. \tag{28}$$

C Architectural Hyperparameters

Table 1: Model configuration of GQA.

Model Size	# Parameters	# Layers	h_q	h_{kv}	d	d_h	d_r	d_f
3.3B	3280.88M	28	24	8	3072	128	64	9384

Table 2: Model configuration of SA.

Model Size	# Parameters	# Layers	h_q	h_{kv}	d	d_h	d_r	d_f	s_a
3.3B	3282.26M	28	24	4	3072	128	64	9816	1
3.3B	3282.26M	28	24	2	3072	128	64	9816	2
3.3B	3280.88M	28	24	8	3072	128	64	9640	1
3.3B	3281.57M	28	24	4	3072	128	64	9728	2
3.3B	3280.88M	28	24	2	3072	128	64	9640	4
3.3B	3282.26M	28	24	8	3072	128	64	9560	2
3.3B	3282.26M	28	24	4	3072	128	64	9560	4
3.3B	3282.26M	28	24	2	3072	128	64	9304	8
3.3B	3280.88M	28	24	8	3072	128	64	9384	4
3.3B	3281.57M	28	24	4	3072	128	64	9216	8
3.3B	3282.26M	28	24	8	3072	128	64	9048	8

Table 3: Model configuration of DeepSeek-V4's MQA.

Model Size	# Parameters	# Layers	h_q	h_{kv}	d	d_h	d_r	r_q	d_f	G
3.3B	3281.61M	28	24	1	3072	512	64	1024	8192	4

Table 4: Validation perplexity (lower is better) and common-sense reasoning accuracy (%, higher is better) of SA against GQA. Rows are grouped by the product of attention compute and KV cache. Top, perplexity on seven pretraining corpora. Bottom, accuracy on seven common-sense reasoning benchmarks (normalized accuracy where applicable, standard accuracy otherwise).

Method	KV Cache	Wikipedia	C4	Pile	RefinedWeb	Cosmopedia	FineWeb	FineWeb-Edu	Avg
SA-1	$4d_h$	13.951	15.970	11.917	18.092	8.330	15.012	8.938	13.173
SA-2	$2d_h$	13.966	15.824	12.005	17.932	8.418	14.901	8.894	13.134
SA-1	$8d_h$	13.840	15.860	11.846	17.872	8.359	14.915	8.897	13.084
SA-2	$4d_h$	13.691	15.722	11.670	17.671	8.354	14.783	8.833	12.960
SA-4	$2d_h$	13.618	15.765	11.796	17.735	8.494	14.818	8.850	13.011
GQA	$16d_h$	13.515	15.769	11.746	17.810	8.360	14.825	8.854	12.983
SA-2	$8d_h$	13.783	15.711	11.676	17.689	8.375	14.740	8.792	12.967
SA-4	$4d_h$	13.644	15.618	11.725	17.592	8.329	14.690	8.790	12.912
SA-8	$2d_h$	14.081	15.728	12.012	17.741	8.359	14.787	8.843	13.079
SA-4	$8d_h$	13.316	15.542	11.446	17.448	8.150	14.597	8.738	12.748
SA-8	$4d_h$	13.351	15.521	11.414	17.454	8.173	14.583	8.752	12.750
SA-8	$8d_h$	13.598	15.469	11.710	17.412	8.269	14.536	8.725	12.817
Method	**KV Cache**	**ARC-E**	**ARC-C**	**OpenBookQA**	**BoolQ**	**HellaSwag**	**Winogrande**	**PIQA**	**Avg**
SA-1	$4d_h$	69.57	41.64	41.40	63.98	62.24	60.22	74.97	59.15
SA-2	$2d_h$	70.37	41.72	41.80	61.80	62.22	58.56	76.12	58.94
SA-1	$8d_h$	69.19	41.55	42.40	65.14	62.51	56.91	75.14	58.98
SA-2	$4d_h$	70.88	42.32	42.00	64.25	62.52	60.38	74.59	59.56
SA-4	$2d_h$	70.12	41.64	42.00	62.39	62.30	58.17	75.79	58.91
GQA	$16d_h$	69.28	41.47	42.40	63.36	62.67	59.43	75.14	59.11
SA-2	$8d_h$	69.44	42.06	44.20	64.25	62.99	59.91	76.22	59.87
SA-4	$4d_h$	67.63	41.04	42.20	63.39	62.66	59.04	74.92	58.70
SA-8	$2d_h$	69.28	39.93	43.20	62.78	62.29	59.27	75.19	58.85
SA-4	$8d_h$	70.71	40.53	42.00	61.80	63.37	59.43	75.41	59.04
SA-8	$4d_h$	69.23	41.81	43.00	63.52	63.08	59.04	76.01	59.38
SA-8	$8d_h$	69.11	40.53	43.80	63.64	63.56	61.48	75.84	59.71

D Experimental Results

Table 5: Fit of the compute-cache scaling law to the eleven SA configurations, with $C = s_a$ and $M = h_{kv}$. Rows are grouped by the product CM. The error is the predicted minus the measured validation loss.

Setting	C	M	Loss	Eq. (16)	Error
SA-1 (h_{kv}=4)	1	4	2.19028	2.19026	−0.00002
SA-2 (h_{kv}=2)	2	2	2.18542	2.18610	+0.00068
SA-1 (h_{kv}=8)	1	8	2.18567	2.18570	+0.00003
SA-2 (h_{kv}=4)	2	4	2.17846	2.17805	−0.00041
SA-4 (h_{kv}=2)	4	2	2.18039	2.18083	+0.00043
SA-2 (h_{kv}=8)	2	8	2.17385	2.17350	−0.00035
SA-4 (h_{kv}=4)	4	4	2.17360	2.17278	−0.00082
SA-8 (h_{kv}=2)	8	2	2.17966	2.17855	−0.00111
SA-4 (h_{kv}=8)	4	8	2.16769	2.16823	+0.00053
SA-8 (h_{kv}=4)	8	4	2.16925	2.17050	+0.00125
SA-8 (h_{kv}=8)	8	8	2.16616	2.16594	−0.00021
RMSE					0.00066