Cost-Optimal Grouped-Query Attention for Long-Context Modeling

Yingfa Chen[1*], **Yutong Wu**[2*], **Chenyang Song**[1], **Zhenleng Thai**[1],
Xingyu Shen[1], **Xu Han**[1†], **Zhiyuan Liu**[1†], **and Maosong Sun**[1]
[1]NLP Group, DCST, IAI, BNRIST, Tsinghua University, Beijing, China
[2]SIST, University of Science and Technology Beijing, Beijing, China
yingfa-c24@mails.tsinghua.edu.cn, wuyutong_yuna@163.com
{han-xu,liuzy}@tsinghua.edu.cn

Abstract

Grouped-Query Attention (GQA) is a widely adopted strategy for reducing the computational cost of attention layers in large language models (LLMs). However, current GQA configurations are often suboptimal because they overlook how context length influences inference cost. Since inference cost grows with context length, the most cost-efficient GQA configuration should vary accordingly. In this work, we analyze the relationship among context length, model size, GQA configuration, and model loss, and introduce two innovations: (1) we decouple the total head size from the hidden size, enabling more flexible control over attention FLOPs; and (2) we jointly optimize the model size and the GQA configuration to arrive at a better allocation of inference resources between attention layers and other components. Our analysis reveals that commonly used GQA configurations are highly suboptimal for long-context scenarios. Moreover, we propose a recipe for deriving cost-optimal GQA configurations. Our results show that for long-context scenarios, one should use fewer attention heads while scaling up the model size. Configurations selected by our recipe can reduce both memory usage and FLOPs by more than 50% compared to Llama-3's GQA, with *no degradation in model capabilities*. Our findings offer valuable insights for designing efficient long-context LLMs.[1]

1 Introduction

It is well established that increasing the size of large language models (LLMs) can improve their language modeling qualities (Hestness et al., 2017; Kaplan et al., 2020). Thus, many prior studies have focused on minimizing model size while maintaining quality to ensure cost-effectiveness (Hoffmann et al., 2022; Hu et al., 2024; Abdin et al., 2024). However, the vast majority of LLMs are Transformer-based (Vaswani et al., 2017; Grattafiori et al., 2024), and the cost of running such architectures does not solely depend on the model size. Specifically, during inference, a cache of keys/values (i.e., KV cache) is maintained to avoid recomputation in attention layers, resulting in **memory costs** that scale linearly with the context length. Also, attention layers include the computation of pair-wise attention scores and the weighted summation of value vectors, incurring per-token **computational costs** that scale linearly with the context length. Many studies have aimed to reduce these costs, including KV cache compression (Li et al., 2024a), prompt compression (Pan et al., 2024; Li et al., 2024b), sparse attention (Lou et al., 2024; Ge et al., 2024; Jiang et al., 2024), etc.

One of the most widely used techniques for reducing memory costs is Grouped-Query Attention (GQA) (Ainslie et al., 2023), in which attention heads are split into groups and the heads in each group share the same KV vectors. Current implementations of GQA have two critical limitations: (1) Most existing models unnecessarily restrict the total number of head dimensions to be equal to the hidden size, resulting in redundant FLOPs (floating-point operations). (2) When deciding on the number of attention heads and groups, current models do not take into account the influence of context length on the computational and memory costs, resulting in suboptimal long-context configurations.

In this paper, we aim to optimize the cost-effectiveness of GQA Transformers from the perspective of resource allocation. Concretely, we categorize inference costs into *time-invariant costs*, which are constant with respect to context length (e.g., fixed model parameters), and *time-variant costs*, which grow with context length (e.g., attention computation and KV cache). To freely

[1] The code and models are available at `https://www.github.com/THUNLP/cost-optimal-gqa`.

[*] indicates equal contribution.

[†] indicates corresponding authors.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 5360–5376
November 4-9, 2025 ©2025 Association for Computational Linguistics

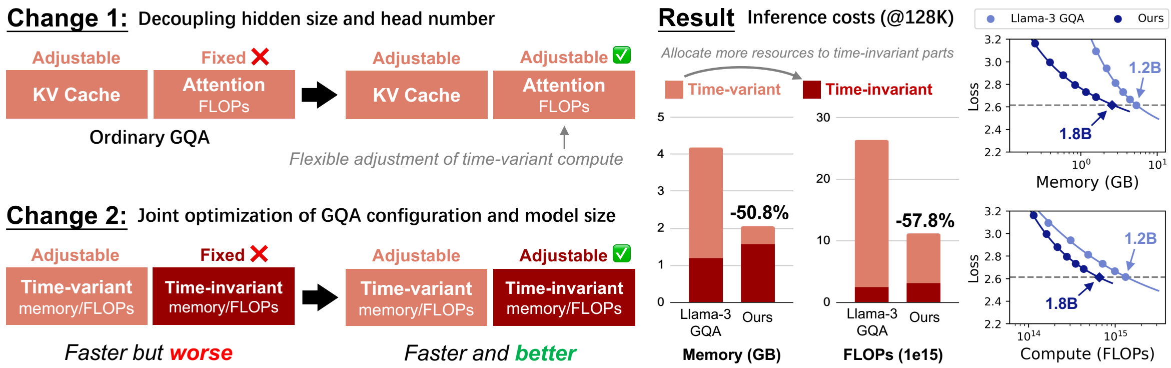

Figure 1: Our approach makes two changes to unlock the flexible adjustment of memory and compute allocation between *time-invariant* components (model weights) and *time-variant* components (KV cache/attention computation). Optimizing resource allocation results in **cost-optimal GQA configuration** ("Ours"), which has markedly lower memory and FLOPs usage compared to Llama-3, without compromising model capabilities.

control the resource allocated to time-variant and time-invariant parts, we make two changes to the existing GQA design procedures: (1) By decoupling the total number of head dimensions and the model hidden size, we unlock a free hyperparameter to control the compute allocated to attention operations. (2) We jointly optimize GQA configurations and model size to modulate the resource allocation between time-variant and time-invariant components. After these changes, we can answer our main research question:

> *Given an expected inference context length and target loss, how can GQA be configured to minimize inference costs while achieving that loss?*

To avoid sweeping all combinations of model sizes and GQA configurations, we present a three-step search procedure (detailed in Section 4). Our approach is empirically validated on models up to 1.2B parameters. Empirical results show that the widely used Llama-3 GQA configuration (Grattafiori et al., 2024) is highly suboptimal at 128K (which is the context length supported by Llama-3). Instead, our approach gives a configuration that achieves the same loss while reducing inference FLOPs and memory usage by more than 50% (Figure 1 (right)).

The contributions of this paper can be summarized by the following points:

- By decoupling the model hidden size from the attention head number and jointly optimizing the model size and GQA configuration, we can flexibly allocate memory and compute resources among time-variant and time-invariant components.
- We present the first rigorous study to search for the optimal GQA configuration in terms of inference costs for reaching a target loss. Our three-step approach can precisely identify cost-optimal GQA configurations without exhaustively sweeping many configurations.
- Our framework reveals valuable insights for designing more cost-effective Transformer LLMs, especially in long-context scenarios.

2 Related Work

This paper explores how to build efficient long-context LLMs based on GQA Transformer. Please refer to the LLM-related surveys (Zhao et al., 2023; Lu et al., 2024) for more details on LLMs.

Grouped-Query Attention The original Transformer model employs multi-head attention (MHA) (Vaswani et al., 2017), in which each layer consists of multiple heads that are computed in parallel, and the layer's output is the sum of the heads' outputs. To improve decoding efficiency, especially improving memory efficiency, multi-query attention (MQA) (Shazeer, 2019) shares the weights of all key and value projections among all heads, significantly reducing KV cache size and memory bandwidth requirements during autoregressive decoding. Grouped-query attention (GQA) (Ainslie et al., 2023) extends this by partitioning heads into groups where each group shares a common KV projection. Formally, MHA is a variant of GQA with independent KV projections per query head,

Notation	Meaning	Adjustable? Vanilla GQA	 This paper	Constrained by
T	Context length	✗	✗	*None*
N	Model size	✗	✓	*None*
n_h	Attention head number	✗	✓	*None*
n_{kv}	KV head number	✓	✓	*None*
L	Number of layers	✗	✗	N and pre-defined aspect ratio (d/L)
d	Model hidden size	✗	✗	N and pre-defined aspect ratio (d/L)
d_{ff}	FFN intermediate size	✗	✗	$d_{\text{ff}} \approx 8d/3$
d_h	Head size	✗	✗	$d_h = 64$
V	Vocabulary size	✗	✗	Pre-defined vocabulary

Table 1: Notations in the paper. We optimize more free hyperparameters, resulting in better cost-efficiency.

while MQA corresponds to the extreme where all queries share one common KV projection. Recent attention methods based on low-rank factorization, such as MLA (DeepSeek-AI et al., 2024), can also be viewed as variants of GQA. Hence, it can be said that most of the current popular LLMs (Groeneveld et al., 2024; Biderman et al., 2023; Hu et al., 2024; Grattafiori et al., 2024; Yang et al., 2025b) are built based on GQA.

Efficient Long-Context Attention Attention mechanisms pose a major bottleneck in long-context settings due to high computational and memory costs, especially from the KV cache. To mitigate this, techniques like sparse attention (Lou et al., 2024; Ge et al., 2024; Jiang et al., 2024), prompt compression (Pan et al., 2024; Xiao et al., 2024), and KV cache compression (Liu et al., 2024; Hooper et al., 2024; Zhang et al., 2024; Yao et al., 2024; Cai et al., 2024) have been proposed. While these methods build on and optimize GQA, they often compromise performance relative to vanilla GQA. Our work focuses on identifying cost-optimal GQA configurations for long-context scenarios through precise characterization of model size, context length, and attention head configurations in terms of their impacts on model performance, computational cost, and memory cost. The efficient long-context attention methods described above remain orthogonal to our GQA architecture search and can be subsequently applied as complementary optimizations to the cost-optimal GQA structures. For more details on efficient long-context attention methods, please refer to the surveys (Yuan et al., 2024; Shi et al., 2024).

Scaling Laws for LLMs Recent studies on scaling laws for LLMs (Hestness et al., 2017; Kaplan et al., 2020; Hoffmann et al., 2022) have established that model loss follows a log-linear relationship concerning model size and training data size. They utilize this relationship to minimize the model loss given a fixed training FLOPs budget. However, there are two critical limitations: (1) These works do not consider the influence of context length on the computational and memory costs. (2) These laws prioritize the optimal allocation of compute during training, ignoring inference costs. Although Sardana et al. (2023) supplement scaling laws by accounting for total inference FLOPs, their inference cost estimation ignores the influence of context length and memory usage during inference. Our work extends these studies by accounting for both the computational and memory costs during inference and addressing the impact of context lengths.

3 Preliminaries: Computational and Memory Costs of GQA Transformers

In this section, we first briefly introduce GQA Transformers (Ainslie et al., 2023) and describe key model configurations and their impact on computational and memory costs. Then, we provide a more accurate formula for the computational and memory costs of Transformer-based LLMs that explicitly considers context length and can guide the design of cost-optimal long-context LLMs. Table 1 lists the main notations in this paper, and Appendix A provides a more complete list.

3.1 GQA Transformers

A Transformer model consists of L layers, each of which consists of an attention block and a feed-forward network (FFN) block. For each layer, let $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^d$ denote the i-th input and output embedding, where d is the model hidden dimension.

Attention Blocks For each head in an attention block, $\mathbf{x}_i$ is first projected into query $\mathbf{q}_i = \mathbf{x}_i\mathbf{W}_q \in$

Component	Parameters	Per-token FLOPs
Input emb.	dV	0
ATT proj.	$2Ldd_h(n_h + n_{kv})$	$4Ldd_h(n_h + n_{kv})$
ATT comp.	0	$4LTn_hd_h$
FFN	$2Ldd_{\text{ff}}$	$4Ldd_{\text{ff}}$
Output emb.	0	$2dV$

Table 2: Parameters and per-token FLOPs (**forward pass**) of the main components in Transformers. "Input emb." and "Output emb." represent the input and output embedding layers, respectively, sharing the same embedding weights. "ATT proj." and "ATT comp." represent the projection and computation processes of all attention blocks, respectively.

$\mathbb{R}^{d_h}$, key $\mathbf{k}_i = \mathbf{x}_i\mathbf{W}_k \in \mathbb{R}^{d_h}$, value $\mathbf{v}_i = \mathbf{x}_i\mathbf{W}_v \in \mathbb{R}^{d_h}$, where d_h is the head dimension, then the attention head output is computed as

$$\tilde{\mathbf{h}}_i = \text{softmax}\left(\frac{\mathbf{q}_i\mathbf{K}_i^\top}{\sqrt{d_h}}\right)\mathbf{V}_i\mathbf{W}_o^\top \in \mathbb{R}^d, \quad (1)$$

where $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o \in \mathbb{R}^{d\times d_h}$ are learnable projection matrices. $\mathbf{K}_i^\top = \left[\mathbf{k}_1^\top \oplus \cdots \oplus \mathbf{k}_i^\top\right]$ and $\mathbf{V}_i^\top = \left[\mathbf{v}_1^\top \oplus \cdots \oplus \mathbf{v}_i^\top\right]$ are the KV cache for the current attention head, where $\oplus$ denotes the concatenation along the sequence dimension. In MHA Transformers, each attention block consists of n_h heads computed in parallel, and the final attention output $\mathbf{h}_i \in \mathbb{R}^d$ is the sum of all head outputs. In GQA, every n_h/n_{kv} query heads share the same KV projection matrices, where n_{kv} is the number of KV heads.

FFN Blocks An FFN block is defined as

$$\mathbf{y}_i = \sigma\left(\mathbf{h}_i\mathbf{W}_{\text{up}}^\top\right)\mathbf{W}_{\text{down}} \in \mathbb{R}^d, \quad (2)$$

where $\mathbf{W}_{\text{up}} \in \mathbb{R}^{d\times d_{\text{ff}}}$, $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d_{\text{ff}}\times d}$ are learnable projection matrices and $\sigma(\cdot)$ is an elementwise activation function.

Hyperparameter Constraints Let V denote the vocabulary size and N denote the model size. We assume that d_h and V are fixed[2], and $d_{\text{ff}} \approx 8d/3$, following common LLM design choices (Grattafiori et al., 2024; Groeneveld et al., 2024; Biderman et al., 2023). For each model size N, we assume that the optimal aspect ratio d/L is determined in advance (taken from Biderman et al. (2023)), so each N corresponds to a unique pair (d, L). Table 1 (right) lists these constraints.

Type	Time-invariant	Time-variant
FLOPs (C_{infer})	$2N$	$4TLd_hn_h$
Mem. (M_{infer})	N	$2TLd_hn_{kv}$

Table 3: The time-invariant and time-variant costs of GQA Transformers during inference.

3.2 Inference Costs of GQA Transformers

Table 2 summarizes the number of parameters for each component in the Transformer model and the FLOPs associated with it. Table 3 summarizes the memory and computational costs during inference.

Inference Computational Costs $C_{\text{infer}}(T)$ is the number of FLOPs used to process one token within the context with T tokens. This is roughly given as

$$\begin{aligned} C_{\text{infer}}(T) &= C_{\text{const}} + C_{\text{att}}(T) \\ &= \underbrace{2N}_{\text{Time-invariant}} + \underbrace{4TLd_hn_h}_{\text{Time-variant}}, \end{aligned} \quad (3)$$

where C_{const} denotes the "time-invariant FLOPs", the number of FLOPs invariant to the current time step. $C_{\text{att}}(T)$ denotes the "time-variant FLOPs", which is the number of FLOPs used to compute the attention softmax process.

Inference Memory Costs $M(T)$ is defined as the memory required to process one token within the context with T tokens. Ignoring the necessary system overhead, we need to store the model parameters and the KV cache, which is roughly:

$$\begin{aligned} M_{\text{infer}}(T) &= N + N_{kv}(T) \\ &= \underbrace{N}_{\text{Time-invariant}} + \underbrace{2TLd_hn_{kv}}_{\text{Time-variant}}, \end{aligned} \quad (4)$$

where N denotes the number of model parameters and $N_{kv}(T)$ denotes the number of values in the KV cache for the context with T tokens.

Takeaways As listed in Table 3, inference costs can be split into four types: time-invariant FLOPs and memory, and time-variant FLOPs and memory. The time-invariant costs are directly proportional to the model size (N), while time-variant FLOPs can be controlled by n_h, and time-variant memory can be controlled by n_{kv}. Thus, adjusting N, n_h, and n_{kv} permits fine-grained control over these four kinds of costs. This analysis also implies that a large model may have lower inference costs if its time-variant costs are low enough.

[2] Keeping d_h and V constant for varying model sizes is a common practice. Examples include Llama-3 (Grattafiori et al., 2024) and Qwen3 (Yang et al., 2025a).

Training Costs Since this work mainly focuses on minimizing inference costs, the calculation for the training costs is left to Appendix C.

4 Method

Our objective is to *find the GQA configuration that minimizes inference costs while attaining a given loss*. We approach this by framing the problem as balancing the time-variant and time-invariant costs. In order to unlock the ability to flexibly allocate different amounts of compute and memory to the time-variant and time-invariant components, we make two changes to existing GQA design procedures: (1) We decouple the number of attention heads from the model hidden dimension, and (2) we jointly optimize the model size and the GQA configuration. Figure 1 (left) shows the effect of these two changes, and Table 1 shows the adjustability of different hyperparameters in this work compared to vanilla GQA.

Change 1: Decoupling the Head Number from the Hidden Dimension Most existing GQA Transformers adopt $n_h \times d_h = d$, which is arbitrarily chosen in the original Transformer paper (Vaswani et al., 2017). This is an unnecessary restriction, rendering GQA unable to adjust the time-variant FLOPs. We decouple n_h from d, unlocking a free hyperparameter n_h that controls the number of FLOPs of attention blocks.

Change 2: Joint Optimization of Model Size and GQA Configuration In addition to the time-variant costs, we also want to control the time-invariant costs (FFNs, attention QKV/output projections, etc.). Specifically, by reducing N, but increasing n_h, we can allocate more compute to time-variant components. Similarly, we can allocate more compute to time-invariant components by increasing N and decreasing n_h. This paper aims to identify the optimal allocation of memory and compute between the time-variant and time-invariant components, by jointly tweaking the GQA configuration (n_h, n_{kv}) and the model size N.

4.1 Cost-Optimal GQA Search

Objective Formulation With the ability to freely adjust the time-variant and time-invariant costs, we formulate the optimization objective as follows,

$$\begin{aligned} &\arg\min_{n_h, n_{kv}, N} Z(T, N, n_h, n_{kv}) \\ &\quad \text{s.t. } \mathcal{L}(T, N, n_h, n_{kv}) \leq \mathcal{L}^* \\ &\quad \text{where } Z = \lambda M_{\text{infer}}^{\alpha} + (1-\lambda) C_{\text{infer}}^{\beta}, \end{aligned} \tag{5}$$

where $\mathcal{L}^*$ is the target LM loss, $\mathcal{L}$ is the model loss, $\lambda \in [0, 1], \alpha, \beta \in \mathbb{R}$ control the trade-off between compute and memory based on deployment constraints[3]. Setting $\lambda = 1$ minimizes only M_{infer}, while $\lambda = 0$ minimizes only C_{infer}. We refer to Z as the *hardware-aware cost*. By default, we set $\lambda = 0.9, \alpha = 1/2, \beta = 1/3$ based on hardware utilization tests in our environment. In other words, the inputs to the optimization objective are $(\mathcal{L}^*, T)$ and the outputs are (N, n_h, n_{kv}).

Influence of Context Length We empirically observe that the effect of context length T on loss $\mathcal{L}$ is largely invariant to N, n_h, and n_{kv} (verified in Section 5.7). This means we can train with moderate context lengths (e.g., $T = 8$K) and extrapolate the loss to longer contexts, saving precious computation resources. However, the influence of model size N and GQA head configuration $H = (n_h, n_{kv})$ on loss is coupled and must be jointly modeled. To this end, we adopt a three-step procedure:

Step 1: Candidate Selection Define a candidate set of attention configurations:

$$\begin{aligned} H_{\text{cand}} = \{&n_h = 1, 2, 4, \ldots, \max(d)/d_h\} \\ &\times \{n_{kv} = 1, 2, 4, \ldots, \max(d)/d_h\} \\ \text{s.t. } &n_{kv} \leq n_h, \end{aligned} \tag{6}$$

where $\max(d)$ is the hidden size of the largest model used to fit scaling curves in step 2. We round $\max(d)/d_h$ to the nearest power of 2 if necessary.

Step 2: Scaling Curves Fitting For each $H \in H_{\text{cand}}$, we train a series of small-scale models with varying N using a sufficiently long context length (we use $T = 8$K), and fit the model loss using a power-law scaling function[4] as

$$\mathcal{L}(N; H) = \left(\frac{a}{N}\right)^b + E, \tag{7}$$

where a, b are configuration-dependent coefficients and E is the "natural entropy of language".

[3] Although M_{infer} and C_{infer} have different measurement units, (λ, α, β) allow us to control the importance of compute and memory resources under a unified metric.

[4] We use the number of non-embedding parameters because it produces more predictable scaling laws in our experiments.

Step 3: Cost Minimization For each GQA configuration H, we solve for the smallest model size $N^*(H)$ that satisfies the loss constraint as

$$N^*(H) = \frac{a}{(\mathcal{L}^* - E)^{1/b}}. \quad (8)$$

Then, we calculate the inference cost for each configuration and select the one with the lowest cost

$$(N^*(H), H^*) = \arg\min_{H} Z(T, N, n_{kv}, n_h). \quad (9)$$

5 Experiments

We first explain the experimental settings (Section 5.1). Then, we present the main results and takeaways (Section 5.2), followed by the actual cost-optimal GQA configurations derived using our approach (Section 5.3) and an analysis of the influence of n_h and n_{kv} on LM loss (Section 5.4). After that, we present the results for the setting where total training FLOPs is aligned (Section 5.6). Finally, we verify that the effect of T on $\mathcal{L}$ is largely independent of N and H (Section 5.7).

5.1 Experimental Settings

More details of the experimental settings are in Appendix D, Appendix E, and Appendix F.

Model Configurations We adopt the popular Llama-3 (Grattafiori et al., 2024) architecture. For each GQA configuration, we train models from 3M to 1.2B in size. We keep the model configurations as close as possible to Biderman et al. (2023). We have $\max(d)/d_h = 32$, this results in 21 candidate configurations (i.e., $|H_{\text{cand}}| = 21$).

Data Configurations We use SlimPajama (Soboleva et al., 2023) in our experiments. It is a deduplicated version of the RedPajama corpus (Weber et al., 2024) with 627B tokens. In most of our experiments, we use a 20:1 ratio between training data and model parameters, as suggested by Hoffmann et al. (2022). Additionally, we always ensure that each batch has 512K tokens.

Training Configurations We follow common practices in most of our experiments. We use the AdamW optimizer with the WSD learning rate scheduler (Hu et al., 2024). We choose the maximum learning rate by sweeping different values with the MHA model for each model size.

	Expected inference context length (T)				
$\mathcal{L}^*$	8K	16K	32K	64K	128K
3.0	$32, 1$	$16, 1$	$8, 1$	$4, 1$	$4, 1$
2.9	$32, 1$	$16, 1$	$16, 1$	$8, 1$	$4, 1$
2.8	$32, 2$	$16, 1$	$16, 1$	$8, 1$	$8, 1$
2.7	$32, 4$	$16, 2$	$16, 1$	$16, 1$	$8, 1$
2.6	$32, 8$	$16, 4$	$16, 2$	$16, 2$	$8, 1$
2.5	$32, 16$	$16, 8$	$16, 4$	$16, 2$	$16, 2$
2.4	$32, 32$	$32, 32$	$32, 8$	$32, 8$	$32, 4$
2.35	$32, 32$	$32, 32$	$32, 32$	$32, 16$	$32, 8$

Table 4: The cost-optimal GQA configuration (n_h, n_{kv}) for different target loss $\mathcal{L}^*$ and context lengths (T), while minimizing the *hardware-aware cost* (Z, see Section 4.1). For reference, the loss of 1B, 3B, and 8B of Llama-3 GQA is 2.615, 2.448, and 2.362, respectively.

5.2 Loss vs. Inference Costs

Here, we compare the loss-cost tradeoffs of different GQA configurations. Figure 2 reports the results for a subset of H_{cand}, showing LM loss as functions of various inference costs (M_{infer}, C_{infer}, and Z), with a context length of 128K tokens. To save space, we report the result of other context lengths in Appendix H.1.

Takeaway 1 We find that loss does not have a simple relationship (e.g., power-plus-constant function) with either memory or computational costs. However, it is possible to predict the loss by fitting the loss as a function of N, then transforming the fitted curves along the x-axis to account for the time-variant costs. Fitting loss as a power-plus-constant function of N is highly accurate, with R^2 values over 0.999.

Takeaway 2 The commonly used Llama-3 GQA configuration (i.e., $H = d/d_h, 8$)[5] is highly suboptimal at 128K context length. For instance, Llama-3.2-1B uses this head configuration and supports 128K context length. At that length, using $H = (8, 1)$ and increasing the model size to 1.8B would **achieve the same loss (2.615) while reducing 50.8% and 57.8% inference memory and FLOPs usage, respectively** (shown in Figure 1 (right)). Alternatively, using $H = 8, 1$ can achieve a loss that is 0.117 lower than Llama-3.2-1B with the same per-token inference budget in terms of Z.

5.3 Cost-Optimal GQA Configuration

Table 4 reports the cost-optimal GQA for different expected inference context lengths T and target

[5] We use "Llama-3 GQA" to refer to the GQA configuration on Llama-3 and not the actual publicly released checkpoint, which is trained on huge amounts of proprietary data.

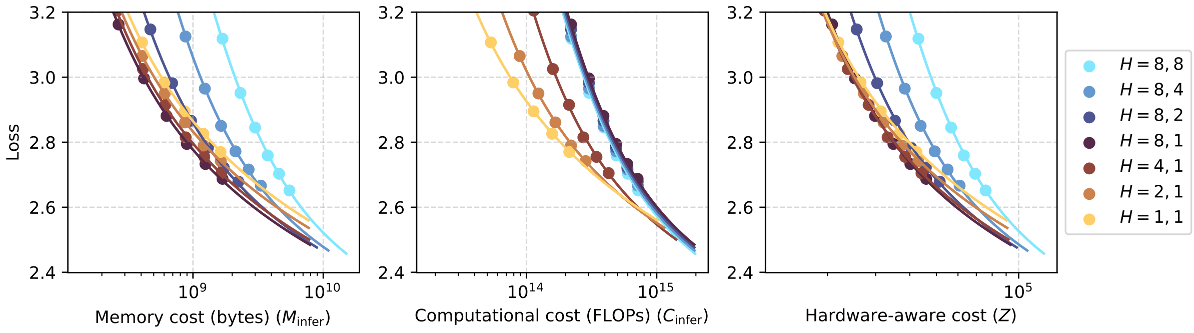

Figure 2: Loss as a function of inference costs with a context length of 128K, assuming we use BF16 for both parameters and the KV cache. $H = (n_h, n_{kv})$ denotes the attention head configuration. n_h and n_{kv} have different effects on the memory cost, computational cost, and loss. x-axis is in log scale.

losses $\mathcal{L}^*$. When the target loss is high, the model is small, making the time-invariant costs low. Thus, the optimal configuration allocates more resources to the time-invariant part by increasing N and reducing n_h and n_{kv}. Similarly, when T is great, the time-variant costs are high, making it more attractive to reduce n_h and n_{kv} more aggressively. The results also indicate that there is nothing especially attractive about the commonly used Llama-3 GQA configuration $(d/d_h, 8)$. For certain combinations of $\mathcal{L}^*$ and T, the GQA configuration is cost-optimal. However, for a greater number of combinations, it is sub-optimal. The result implies that the configuration of GQA Transformers should consider the expected inference context length. Directly applying the popular GQA configuration results in wasting hardware resources.

5.4 Influence of Query and KV Heads

Figure 3 shows the relationship between loss and the number of query heads and KV heads (i.e., different GQA configurations), with a model size of 1.2B. Similar results are observed with other model sizes as well. We emphasize two main takeaways.

Takeaway 1 The loss reduction by increasing either n_h or n_{kv} exhibits diminishing returns. This means that when n_h or n_{kv} is great, increasing these hyperparameters to reduce loss may not be worth the cost increase. We also found that they exhibit a power-plus-constant relationship (details in Appendix I).

Takeaway 2 Increasing n_h reduces the loss more than increasing n_{kv} by the same amount, although both of them cause the same parameter increase. This means the n_h is more important for model expressivity. Having more query heads allows the model to capture a greater number of dependency patterns. Meanwhile, having more KV heads provides more capacity to store information for each token. The empirical results may indicate that the former is more important for performance.

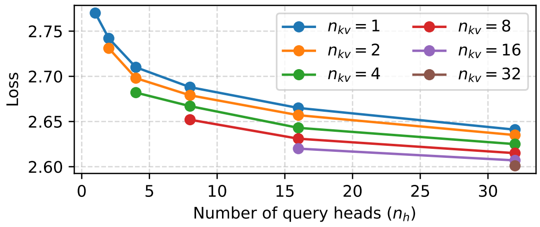

Figure 3: The loss for different number of query heads (n_h) and KV heads (n_{kv}), with 1.2B model parameters.

Evaluation Metric	$H = 32, 8$ (Llama-3 GQA)	$H = 8, 1$ (Ours)
Train. throughput (tok/s)	18,655	**31,260**
Infer. throughput (tok/s)	12,921	**20,643**
Common-sense	45.7%	45.5%
NIAH (1-8K)	90.9%	96.9%
NIAH (16K)	30.4%	46.0%
NIAH (32K)	15.1%	18.7%
NIAH (64K)	6.1%	7.9%
NIAH (128K)	5.2%	6.7%

Table 5: The throughput of two GQA configurations at 128K context length, and their accuracy on common-sense reasoning (average of 8 tasks) and retrieval tasks (NIAH, varying context length). Although $H = 8, 1$ has more parameters (1.8B vs. 1.2B), it is much faster for both training and inference.

5.5 Downstream Performance

Now, we compare the cost-optimal configuration against Llama-3 GQA in terms of training/inference throughput and downstream performance. At $T = 128\text{K}$ and $\mathcal{L}^* = 2.615$ (the loss

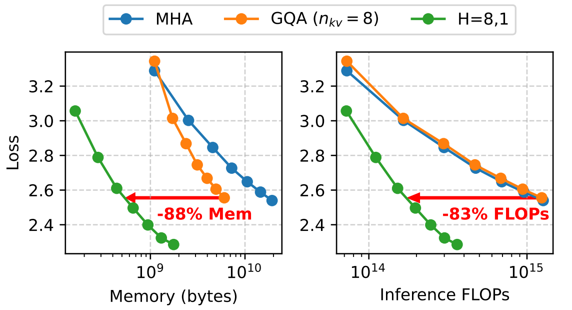

Figure 4: Loss as a function of memory and computational costs, aligned by total training FLOPs at 128K tokens. Each curve is trained with the same amount of training compute.

of Llama-3 GQA at 1.2B model size), the cost-optimal GQA configuration is $H = 8, 1$. Specifically, we train two models, one with $H = 32, 8$ (Llama-3 GQA) and one with $H = 8, 1$. Training starts with a 4K context length on 20B tokens. It is then trained with 128K context length for 1B tokens. More training details is given in Appendix J.1.

Training throughput is computed based on the training time while inference throughput is measured with a batch size of 1 on one NVIDIA A800 GPU (with $T = 128\text{K}$). For downstream performance, we evaluate the models on zero-shot common-sense reasoning (Gao et al., 2024) and needle-in-a-haystack (NIAH) (Hsieh et al., 2024), which are two widely used LLM benchmarks (more details in Appendix J.2). The result is shown in Table 5. One can see that the differences in common-sense reasoning and long-context retrieval are rather small. Meanwhile, the cost-optimal model ($H = 8, 1$) is much more efficient.

5.6 Aligning Training Costs

In the previous sections, the training data is always 20 tokens per parameter (i.e., the Chinchilla law). This favors configurations that spend more FLOPs per token. Instead, we can allow more compute-efficient configurations to use more training data to align the training costs of different configurations.

Figure 4 reports the result when we always train with $T = 128\text{K}$[6]. We find that using fewer heads is even more advantageous due to the additional training data, resulting in a model with the same loss but with 88% and 83% lower memory and FLOPs usage.

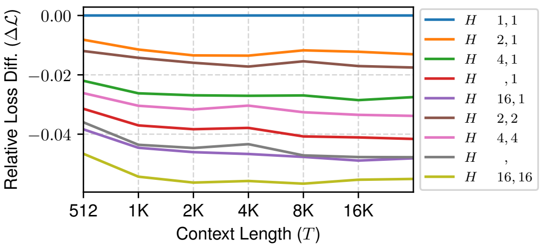

Figure 5: Relative loss difference between various GQA configurations and the $H = 1, 1$ model, as a function of context length T. Model size is 470M.

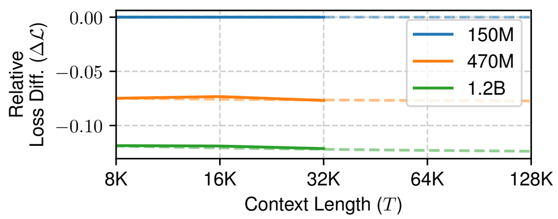

Figure 6: Relative loss difference between varying model size and the 150M model, as a function of context length T. These are MHA models.

5.7 Influence of Context Length

In this section, we empirically show that the relationship between context length T and loss $\mathcal{L}$ is largely invariant to N and n_h when T is sufficiently large. To this end, we measure the relative loss difference between various models and a "baseline":

$$\Delta\mathcal{L}(T) = \frac{\mathcal{L}(T) - \mathcal{L}_{\text{baseline}}(T)}{\mathcal{L}_{\text{baseline}}(T)}$$

Figure 5 shows the relative loss difference between various GQA configurations with $H = 1, 1$ as the baseline. Figure 6 shows this relationship when varying N, with N=150M as the baseline. The results show that the relative loss difference is relatively flat when $T >$ 8K (all fluctuations are less than 1%). The main takeaway is that when applying our cost optimization procedure to longer contexts, we do not have to repeat step 2 (an expensive process) with longer contexts since the loss change of each model will remain roughly the same.

6 Conclusion

To optimize the allocation of FLOPs and memory between time-invariant and time-variant components of GQA Transformers, we first decouple the number of attention heads from the model hidden dimensions, enabling a more flexible distribution of FLOPs and memory. Next, we refine the estimation of computational and memory costs in existing

[6] LMs are usually trained with short contexts most of the time, so this result may not apply.

approaches by incorporating context length. Our findings reveal that typical configurations of GQA are significantly suboptimal for specific context lengths. Through detailed analysis, we provide valuable insights for improving resource allocation by jointly adjusting the model size and the number of attention heads. As the demand for greater inference context lengths continues to grow, our work marks a critical advancement toward efficient long-context LLMs.

Limitations

Like most phenomena in neural language models, we cannot be certain that the conclusions will hold when the models are further scaled up. The power-plus-constant scaling law is also not guaranteed, although it has been empirically validated up to hundreds of billions of parameters. Similarly, there is no guarantee that these laws and our conclusions will hold for an arbitrarily large amount of training data. In general, we have kept our experiments close to research conventions, and the scale of the largest models in our experiments (i.e., 1.2B for Llama-3 GQA and 1.8B for our cost-optimal GQA) is comparable to some real-world LLMs.

We have not thoroughly ablated the influence of all possible hyperparameters due to limited resources. Some important hyperparameters that may affect our conclusions include the head dimension, vocabulary size, and model aspect ratio. More extensive ablation studies are a promising future research direction.

More recent LLMs have adopted even more advanced techniques that complicate the calculations of inference costs. Some notable techniques include speculative decoding, sparse mixture-of-experts, and hybrid recurrence-attention models. Applying our approach to such models remains a promising area for future research.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (2024YFB4505603) and a grant from the Guoqiang Institute, Tsinghua University.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, and 1 others. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of EMNLP*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv preprint arXiv:1607.06450*.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. In *Proceedings of ICML*.

Ruisi Cai, Yuandong Tian, Zhangyang Wang, and Beidi Chen. 2024. LoCoCo: Dropping In Convolutions for Long Context Compression. In *Proceedings of ICML*.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, and 81 others. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. *arXiv preprint arXiv:2405.04434*.

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, and 5 others. 2024. The Language Model Evaluation Harness.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive KV cache compression for llms. In *Proceedings of ICLR*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 17 others. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*.

Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, and 24 others. 2024. OLMo: Accelerating the Science of Language Models. In *Proceedings of ACL*.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory F. Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. 2017. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, and 3 others. 2022. An empirical analysis of compute-optimal large language model training. In *Proceedings of NeurIPS*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. *arXiv preprint arXiv:2401.18079*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? *arXiv preprint arXiv:2404.06654*.

Shengding Hu, Yuge Tu, Xu Han, Ganqu Cui, Chaoqun He, Weilin Zhao, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Xinrong Zhang, Zhen Leng Thai, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, and 5 others. 2024. MiniCPM: Unveiling the potential of small language models with scalable training strategies. In *COLM*.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, and 1 others. 2024. MInference 1.0: Accelerating Prefilling for Long-Context LLMs via Dynamic Sparse Attention. In *Proceedings of ICML*.

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

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of ICLR*.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. 2024a. A survey on large language model acceleration based on KV cache management. *arXiv preprint arXiv:2412.19442*.

Zongqian Li, Yinhong Liu, Yixuan Su, and Nigel Collier. 2024b. Prompt compression for large language models: A survey. *arXiv preprint arXiv:2410.12388*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. In *Proceedings of ICML*.

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

Zhenyan Lu, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Xiwen Zhang, Nicholas D Lane, and Mengwei Xu. 2024. Small language models: Survey, measurements, and insights. *arXiv preprint arXiv:2409.15790*.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of ACL*.

Alec Radford and Karthik Narasimhan. 2018. Improving language understanding by generative pre-training.

Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. 2023. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws. *arXiv preprint arXiv:2401.00448*.

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

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the Cost Down: A Review on Methods to Optimize LLM's KV-Cache Consumption. In *Proceedings of COLM*.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of NeurIPS*.

Maurice Weber, Daniel Y Fu, Quentin Gregory Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Re, Irina Rish, and Ce Zhang. 2024. RedPajama: an Open Dataset for Training Large Language Models. In *Proceedings of NeurIPS Datasets and Benchmarks Track*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *Proceedings of ICLR*.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. 2020. On Layer Normalization in the Transformer Architecture. In *Proceedings of ICML*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2025b. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Yao Yao, Zuchao Li, and Hai Zhao. 2024. SirLLM: Streaming Infinite Retentive LLM. In *Proceedings of ACL*.

Jiayi Yuan, Hongyi Liu, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhuo Xu, Zirui Liu, and 1 others. 2024. KV Cache Compression, But What Must We Give in Return? a Comprehensive Benchmark of Long Context Capable Approaches. In *Proceedings of EMNLP*.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. In *Proceedings of NeurIPS*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2024. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of NeurIPS*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Notation	Meaning
	Model hyperparameters
V	Vocabulary size, always set to 50,304.
L	Number of layers
d	Model hidden dimension
d_h	Head size, always set to 64.
d_{ff}	FFN intermediate size, we always set $d_{\text{ff}} = 8d/3$.
σ	The activation function in FFN
n_h	Number of attention heads
n_{kv}	Number of KV heads (or groups in GQA)
	Inference and Training Costs
C_{infer}	The computational cost (in FLOPs) per forward pass with a context length of T tokens.
M_{infer}	The memory usage (in floating-point values) of serving the model with a context length of T tokens.
C_{train}	The computational cost (in FLOPs) used to train the model with a context length of T tokens.
M_{train}	The memory usage (in floating-point values) of training the model with a context length of T tokens.
Z	Hardware-aware costs combining both M_{infer} and C_{infer}. Defined in Section 4.1.
	Other parameters
T	Context length
N	Number of model parameters.
D_{train}	Number of training tokens.
λ, α, β	Hyperparameters controlling the importance of memory and compute resources.

Table 6: List of notations used in the paper.

A Notations

For completeness, we provide a list of notations we used in the paper, reported in Table 6.

B Discussions

What About Other Efficient Attention? This paper primarily adjusts the allocation of compute and memory usage by tweaking the model size (controlled with L and d) and head configuration (n_h, n_{kv}) in GQA, which is a rather simple method. As mentioned, there are many techniques for improving the efficiency of the attention layer, although those have enjoyed less adoption. When using these techniques, the computational and memory costs may be considerably different, and some of our conclusions may not apply. Despite so, our work is still a valuable improvement over existing implementations of GQA.

Recently, Multi-head Latent Attention (MLA) (DeepSeek-AI et al., 2024) was proposed as a strong alternative to GQA for reducing the KV cache size. During inference, MLA reformulates the attention computation such that all heads share a unified representation for keys and values. In this case, our analysis still applies, since MLA can be seen as a kind of GQA with a different head dimension (d_h) and number of attention heads (n_h, n_{kv}), and it uses a more complex function to generate the QKV vectors.

What If Context Length Varies? The formulas for computational costs (see Table 8) are affine functions of T, so the *expected costs* are:

$$\begin{aligned}\mathbb{E}(C_{\text{infer}}(T)) &= C_{\text{infer}}(\mathbb{E}(T))\\ \mathbb{E}(M_{\text{infer}}(T)) &= M_{\text{infer}}(\mathbb{E}(T))\\ \mathbb{E}(C_{\text{train}}(T_{\text{train}})) &= C_{\text{train}}(\mathbb{E}(T_{\text{train}}))\\ \mathbb{E}(M_{\text{train}}(T_{\text{train}})) &= M_{\text{train}}(\mathbb{E}(T_{\text{train}}))\end{aligned}$$

where T_{train} is the context length during training. Hence, it suffices to compare the costs with the expected context length.

Will the Findings Break Down When Scaling Up the Model/Data Size? This is a never-ending argument against most neural architectural changes, because no matter the scale of our experiments, we can never be sure that the behavior holds for larger scales. However, our experiments have already covered model sizes up to 1.2B, which is already the size of some widely-used models at the moment (Grattafiori et al., 2024; Yang et al., 2025b). Empirically, it has been widely validated that the scaling law is highly predictable to a good extent beyond the largest model (e.g., Llama-3 accurately predicted the loss of a 405B model with experiments on model sizes up to 16B). Thus, we are confident that our conclusions hold at least for models up to 10B parameters.

B.1 How to Calculate the Costs of Models of Arbitrary Sizes?

In step 3 of our procedure (proposed in Section 4.1), we arrive at a critical model size $N^*(H)$. It is a real value, so it does not correspond to an actual model configuration. To calculate the inference costs ($M_{\text{infer}}, C_{\text{infer}}, Z$) of a model of this size, we need H and the aspect ratio of the model $a = d/L$. H is already given, which may be a function of d. For the aspect ratio, we perform linear interpolation between the nearest two pre-defined model configurations. The pre-defined model aspect ratios are given in Table 7. Then, we use binary search to find the L that corresponds to $N^*(H)$. We can calculate d from L and a. Then, we calculate n_h and n_{kv} from d and the specified configuration. With all these values (non-integers) known, we can calculate the model size as well as the inference costs.

N	L	d
1.2B	36	1536
1.8B	36	2048
4B	48	2560
6B	54	3072
13B	64	4096
33B	72	6144
64B	80	8192

Table 7: The pre-defined configurations used to calculate the aspect ratio of arbitrarily sized models. For models smaller than 1.2B, we use the configurations in Table 9.

Cost Type	**Time-invar.**	**Time-var.**
Infer. FLOPs (C_{infer})	$2N$	$4TLd_hn_h$
Infer. Mem. (M_{infer})	N	$2TLd_hn_{kv}$
Train. FLOPs (C_{train})	$6D_{\text{train}}N$	$12D_{\text{train}}TLd_hn_h$
Train. Mem. (M_{train})	$4N$	TdL

Table 8: The time-invariant and time-variant costs of GQA Transformers during inference and training.

To produce an actual model in practice, we suggest simply choosing the configuration (N, n_h, n_{kv}) closest to the derived answer in step 3. The slight variations in the performance of the resulting configuration are negligible compared to the huge cost savings gained by selecting the cost-optimal configuration using our approach.

C Training Costs of GQA Transformers

Training Computational Costs In addition to inference costs, different head configurations also result in different training costs, because the number of training FLOPs, C_{train}, is a function of C_{infer}. Following Kaplan et al. (2020), we estimate the FLOPs of the backward pass as double the FLOPs of the forward pass. Let D_{train} denote the number of training tokens, T_i denotes the number of tokens preceding the i-th training token in the training corpora, then the training FLOPs are:

$$C_{\text{train}} \approx 3D_{\text{train}} C_{\text{infer}} \left(\overline{T}\right) \tag{10}$$

$$= 6D_{\text{train}}(N + \underbrace{2L\overline{T}d_h n_h}_{\text{Attention}}), \tag{11}$$

where $\overline{T}$ is the average value of $\{T_i | i = 1, \cdots, D_{\text{train}}\}$. When all examples in the training corpora are set to the constant length T_{train}, during training, we have $\overline{T} = T_{\text{train}}/2$. However, in practice, when training long-context LLMs, it is more common to use short contexts for most of the time, and only use long contexts consisting of a small number of tokens to adapt the model to the target context length. Hence, the time-variant FLOPs may only make up a small portion of the training FLOPs, making the cost largely independent of the GQA configuration. Consequently, our paper considers training costs, but focuses more on optimizing inference costs.

Training Memory Costs We only need to store model parameters, activations, gradients, and optimizer states during training. Assuming the widely-used Adam (Kingma and Ba, 2015) optimizer without offloading any storage to the CPU, the memory cost is roughly:

$$M_{\text{train}}(T) \approx 4N + \underbrace{TdL}_{\text{Activations}}. \tag{12}$$

While it is important to lower the cost of caching activations when T is large, we do not have a free hyperparameter to adjust this cost (like n_h for computational costs and n_{kv} for memory costs). To reduce the size of activations, we have to modify d and/or L, which either drastically changes the model size or its aspect ratio. Either of such changes leads to major consequences that are beyond the scope of this paper. Regarding the $4N$ part of training memory cost, it is only dependent on the total model size, so it suffices to minimize the model size, which is already addressed in many existing works (Kaplan et al., 2020; Grattafiori et al., 2024; Sardana et al., 2023).

D Model Configurations

Table 9 shows the configurations of the models in our experiments for fitting the scaling law. In general, we ensure that $d_h = 64$, $d_{\text{ff}} \approx 8d/3$ (rounded to the closest multiple of 32) when scaling the model size, which is adopted from common hyperparameters found in existing LLMs such as GPT (Radford and Narasimhan, 2018) and Llama (Grattafiori et al., 2024). We also ensure that the aspect ratio d/L is similar to those used by existing modeling scaling works (Biderman et al., 2023; Hoffmann et al., 2022; Yang et al., 2025b). We use the GPT-2 tokenizer, which has a vocabulary size of 50,304, and we tie the input and output embeddings.

Learning Rate The maximum learning rate (LR) is chosen by a grid search on $\{1 \times 10^i, 2 \times 10^i, 5 \times 10^i \mid i = -3, -4, -5\}$ with the vanilla MHA, and choosing the one with best LM loss. Then, we just keep the LR the same across different GQA configurations. While different configurations may have different optimal LR, exhaustively sweeping all LR for each configuration is prohibitively expensive.

Differences From Vanilla GPT Compared to the vanilla GPT model (Radford and Narasimhan, 2018), we make the following changes to better align with more recent LLMs:

- We use RoPE (Su et al., 2024) with a θ value of 500,000, which is widely used in current LMs (Grattafiori et al., 2024).
- We use SwiGLU FFN instead of the ReLU FFN in GPT.
- We use pre-norm (Xiong et al., 2020) and use RMSNorm (Zhang and Sennrich, 2019) instead of LayerNorm (Ba et al., 2016), which is more common in current LLMs. The epsilon in RMSNorm is 10^{-6}.
- Our model has no bias terms or dropout, which is also common practice and can slightly increase the training efficiency.

E Data Processing

In most of our experiments, we used SlimPajama (Soboleva et al., 2023). We append an EOS token to each document in the corpus before chunking the documents into the specified training length. If the last chunk is shorter than the specified training length, it will be discarded.

F Training Configurations

Here, we provide the default training configurations we used during the experiments.

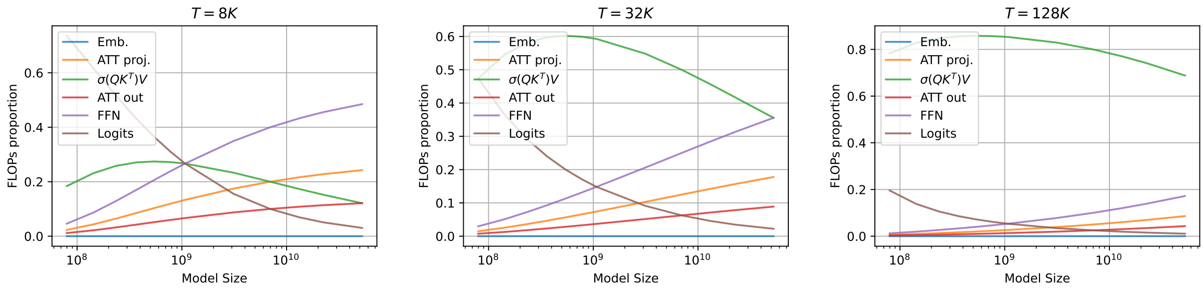

Figure 7: The proportion of FLOPs allocated to different components in a Transformer LM, with multi-head attention and RoPE. As the context length increases, most FLOPs are spent on the time-variant computation of the attention operator $\sigma(\mathbf{QK}^\top)\mathbf{V}$, where σ is the row-wise softmax function.

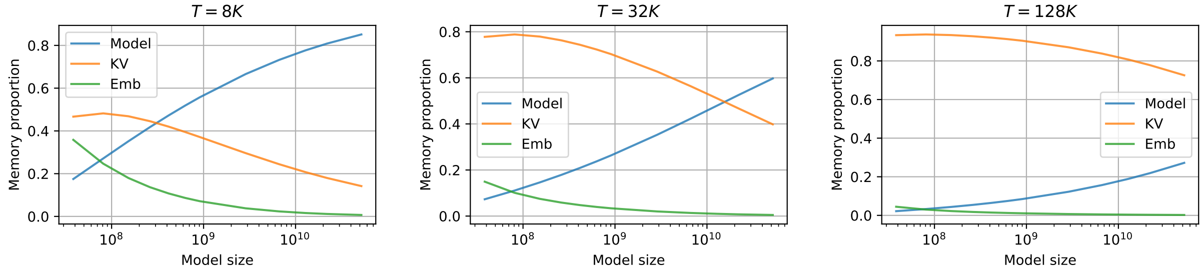

Figure 8: The proportion of memory allocated to different components in a Transformer LM, with multi-head attention and RoPE. As the context lengths increase, most of the memory usage is spent on storing the KV cache.

Model size	L	d	d_h	LR
3M	4	256	64	1e-3
19M	6	512	64	1e-3
85M	12	768	64	1e-3
150M	12	1024	64	1e-3
200M	16	1024	64	5e-4
470M	24	1280	64	5e-4
680M	24	1536	64	2e-4
1.2B	36	1536	64	2e-4

Table 9: The configurations of the vanilla models with MHA in our experiments, we try to keep it as close to the configurations from Biderman et al. (2023) as possible.

- **Optimizer:** We use the widely-used AdamW optimizer (Kingma and Ba, 2015), with $\beta_1 = 0.9, \beta_2 = 0.95$, and a weight decay of 0.1. We only apply weight decay to linear layers, which excludes the re-scaling factor in RMSNorm. We also use a gradient clipping value of 1.0.

- **Learning rate scheduler:** We use the warmup-stable-decay (WSD) LR scheduler (Hu et al., 2024), with a maximum LR of $5 \cdot 10^{-4}$, 10% warmup steps steps and 20% decay steps. Warmup starts from 0 and increases linearly to the maximum LR. The decay stage uses a cosine annealing scheme, where the minimum LR is 10% of the maximum LR.

- **Batch size:** 512K tokens.

- **Floating-point precision:** We use BF16 during training and FP16 during evaluation.

Hardware All training experiments were run on A800 GPUs, mostly with 8 GPUs.

G Memory and Compute Allocations by Model Size

Figure 7 and 8 show the FLOPs and memory breakdown of different components as a function of model size. One can see that changes in the model size and/or context length can influence the allocation of FLOPs and memory between different components in the model. For instance, when the context has 128K tokens, the vast majority of FLOPs is spent computing the attention scores and value summation (i.e., softmax $\left(\mathbf{q}_i\mathbf{K}^\top/\sqrt{d_h}\right)\mathbf{V}$), and the vast majority of memory is spent caching KVs. With 1B model parameters, roughly 90% of memory will be spent storing the KV cache, and only 10% will be used to store the model parameters (assuming the KVs and model parameters use the same precision). In other words, the time-variant costs dominate the overall inference costs. Thus, at this context length, we can minimize the overall costs by allocating more resources to the time-invariant components by increasing N and decreasing n_h and n_{kv}.

H More Results: Loss vs. Inference Costs

Here, we provide the results for the relationship between loss and inference costs for other context lengths. The results are shown in Figure 9, 10, and 11. We can see that for shorter context lengths, the gain of reducing n_h or n_{kv} is relatively small, but the commonly used GQA ($n_{kv} = 8$) configuration is still suboptimal at 32K context length. At 1.2B parameters, GQA uses more FLOPs and memory than $H = 8, 1$. For longer context lengths such as 512K, we can achieve the same loss with less than 10% of the original memory usage by using fewer KV heads, but a larger model (increasing N).

H.1 Influence of Query and KV Heads for Different Context Lengths

Here, we provide the supplementary results for Section 5.4 for other context lengths (8K, 32K, and 512K). Similar to the previous section, a greater context length means that the advantage of using fewer heads is greater. In the following section, we explicitly fit the relationship between loss and n_h and n_{kv} with power-plus-constant functions.

I The Scaling Laws of Attention Heads

In this section, we show that one can predict the loss for a certain head configuration using experiments with a smaller number of heads. Specifically, we find that—for the first time—the relationship between loss and the number of attention heads closely resembles a power-plus-constant function:

$$\mathcal{L}(n_h) = a n_h^b + c$$

where $\mathcal{L}$ is the LM loss, and $a, b, c \in \mathbb{R}$ are coefficients. Figure 12 shows that this relationship is observed with different model sizes. The concrete functions for the curves are:

$$\mathcal{L} = 0.579 n_h^{-0.124} + 2.473 \qquad \text{(470M)}$$
$$\mathcal{L} = 0.398 n_h^{-0.177} + 2.583 \qquad \text{(680M)}$$
$$\mathcal{L} = 0.301 n_h^{-0.227} + 2.622 \qquad \text{(1.2B)}$$

Since the larger model has a greater constant term, this means that these curves will intersect at a certain point (at around $n_h = 8\text{K}$). This is likely incorrect, since the 1.2B model has strictly more parameters than the other models (although at such large values of n_h, the relative difference in model size is very small). This means that the fitted curves will break down before $n_h = 8\text{K}$. Fortunately, virtually all LLMs with open weights have fewer than 128 heads, and the fitted curves are very accurate up to 128 heads with R^2 values over 0.999. Thus, we conclude that the law is empirically accurate for the vast majority of openly available LLMs.

Similarly, Figure 13 shows that this trend is consistent across different context lengths. The fitted curves are

$$\mathcal{L} = 1.513 n_h^{-0.039} + 1.53 \qquad (T = 1K)$$
$$\mathcal{L} = 1.436 n_h^{-0.041} + 1.53 \qquad (T = 2K)$$
$$\mathcal{L} = 1.356 n_h^{-0.044} + 1.53 \qquad (T = 8K)$$

When n_h approaches infinity, the model parameters will be dominated by the attention projection matrices (i.e., QKVO projections). Hence, they converge to the same constant term, which is known as the "natural entropy of language". During curve fitting, this constant term is chosen to minimize to fitting error, and we arrive at 1.53. The R^2 values of these fits are over 0.999.

From these results, we conclude that this power-plus-constant scaling law between loss and the number of heads is exhibited independently of model size and context length. One important implication of this result is that increasing the number of heads to improve model quality gives diminishing returns. This means that beyond a certain point, the loss reduction brought by further increasing the number of heads is not worth the cost increase.

I.1 Constant Number of KV Heads

Some LMs (e.g., Llama-3 (Grattafiori et al., 2024)) keep the number of KV heads constant when scaling up the model. Therefore, we also investigate the relationship between LM loss and n_h when n_{kv} is constant. Figure 14 shows this relationship with different values of n_{kv} and two model sizes. We

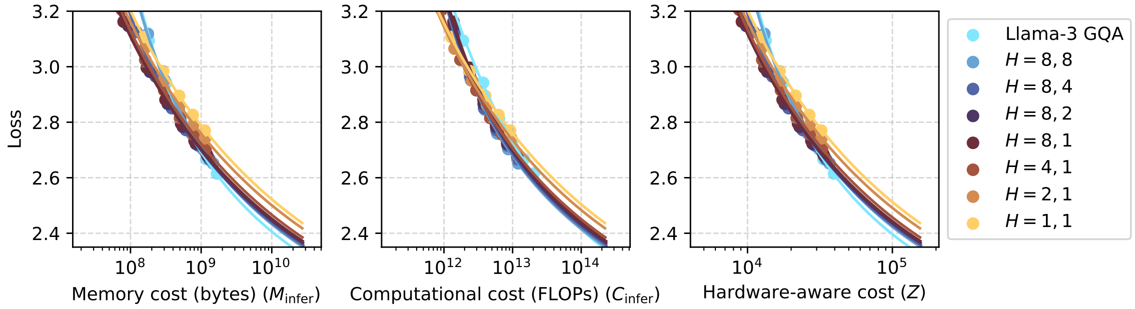

Figure 9: Loss as a function of memory, computational, and hardware-aware (Z in Section 4.1) costs during inference with a **context length of 8K tokens**.

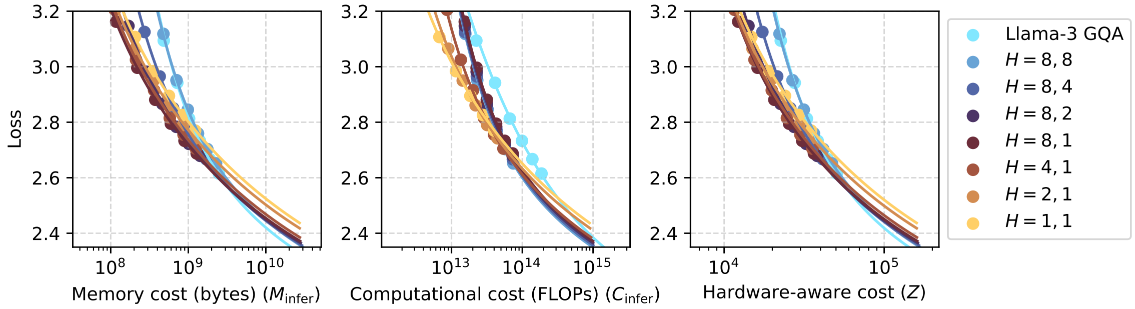

Figure 10: Loss as a function of memory, computational, and hardware-aware (Z in Section 4.1) costs during inference with a **context length of 32K tokens**.

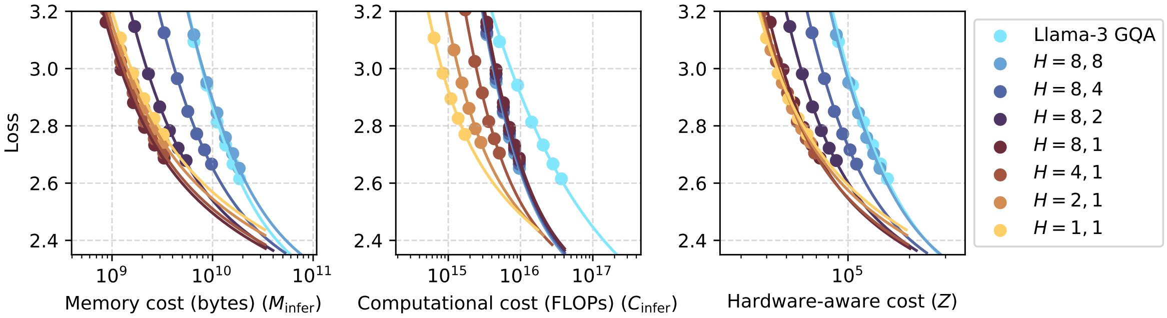

Figure 11: Loss as a function of memory, computational, and hardware-aware (Z in Section 4.1) costs during inference with a **context length of 512K tokens**.

discover that the relationship is still a power-plus-constant law, but the fitted curves are notably less accurate, with R^2 values over 0.97. It is worth noting that the increase in fitting error compared to Section I) may be attributed to the use of a smaller model (150M vs. 470M).

J Experimental Details: Downstream Performance

This section provides details for Section 5.5.

J.1 Training

The training run for both the Llama-3 GQA and $H = 8, 1$ (cost-optimal GQA) models are exactly the same. It consists of two phases. The first phase uses the same settings as the scaling experiments in Section 5.1. After 20B tokens, we continue training with 128K context length for 1B tokens, using new optimer states. This phase uses a lower maximum LR of 1e-5 for stability and to avoid catastrophic forgetting.

J.2 Evaluation

Here, we provide more details regarding the downstream task performance evaluation in Section 5.5. We use LM-Evaluation-harness (Gao et al., 2024) for common-sense reasoning, and the needle-in-a-haystack tasks from RULER (Hsieh et al., 2024).

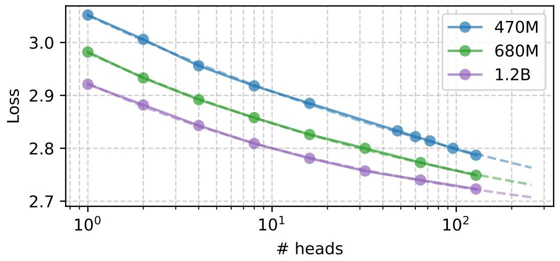

Figure 12: The relationship between LM loss and the number of attention heads, fitted with a power-plus-constant function. The training context length is 1K.

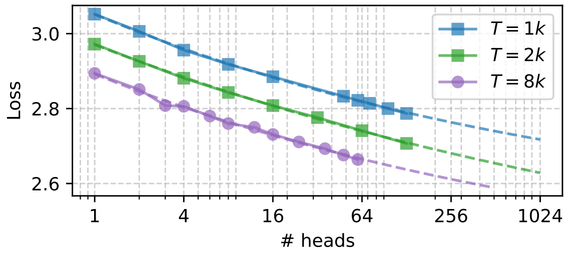

Figure 13: The relationship between LM loss and the number of attention heads, fitted with a power-plus-constant function. The model size is 470M.

For both of these tasks, we evaluate the last four checkpoints of the model, and report the average score of it. This is for reducing the randomness in the results.

Common-Sense Reasoning Tasks We use the popular LM-Evaluation-Harness (Gao et al., 2024) for evaluating common-sense reasoning capabilities. We evaluate on the common-sense reasoning tasks specified by the official implementation, which includes 9 tasks/datasets: ARC-Challenge, ARC-Easy, BoolQ, HellaSwag, Lambada, PIQA, SocialIQA, Wikitext, and Winograd. The scores we report in Table 5 are the average accuracy score (excluding Wikitext, which is evaluated with perplexity). When available, we use the normalized accuracy scores instead of raw accuracy scores.

Retrieval Task We report the average accuracy of the synthetic S-NIAH tasks from RULER (Hsieh et al., 2024), which tests the model's ability to retrieve a certain "needle" (i.e., some special information) from a large body of irrelevant text.

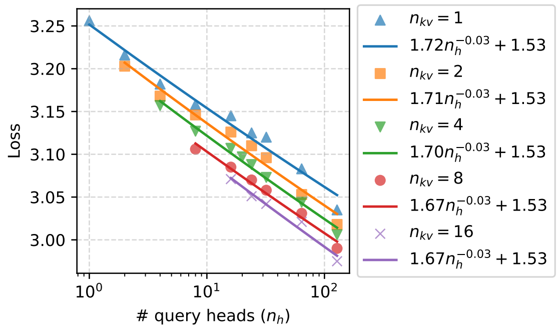

Figure 14: The relationship between loss and n_h when n_{kv} is constant. Model size is 150M.

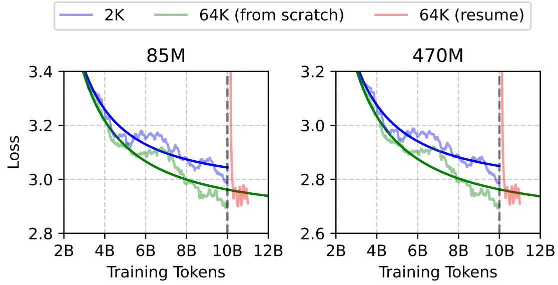

Figure 15: The loss curves of a model with 2K context length adapted to 64K through post-training compared to a model trained with 64K from scratch.

J.3 Context Length Extension by Post-Training

LLMs are typically trained on shorter sequences in practice, followed by adaptation to longer contexts using a smaller amount of data tailored to the target context length. To ensure the validity of our conclusions in such training scenarios, we adapted a checkpoint initially trained with a 2K context length to a 64K context length through continual pretraining. This adapted model was then compared to a model trained from scratch with a 64K context length. As illustrated in Figure 15, the adapted model rapidly converges toward the performance of the model trained from scratch with a 64K context length. This indicates that, with sufficient post-training, the loss of the adapted model approaches that of a model trained entirely from scratch. Consequently, our findings regarding inference costs and the relationship between loss, context length, and head configuration remain applicable to post-training scenarios.

K AI Assistance in Research and Writing

We have used AI for code completion during implementation and grammar-check during paper-writing. We do not explicitly instruct AI to write any part of this paper.