Mixture of Weight-shared Heterogeneous Group Attention Experts for Dynamic Token-wise KV Optimization

Guanghui Song[1] **Dongping Liao**[2] **Yiren Zhao**[3]

Kejiang Ye[1,4] **Cheng-zhong Xu**[2] **Xitong Gao**[1,4*]

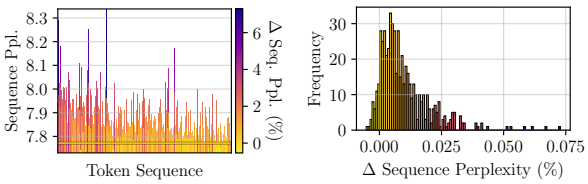
[1] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
[2] State Key Lab of IOTSC, University of Macau
[3] Imperial College London [4] Shenzhen University of Advanced Technology

Abstract

Transformer models face scalability challenges in causal language modeling (CLM) due to inefficient memory allocation for growing key-value (KV) caches, which strains compute and storage resources. Existing methods like Grouped Query Attention (GQA) and token-level KV optimization improve efficiency but rely on rigid resource allocation, often discarding "low-priority" tokens or statically grouping them, failing to address the dynamic spectrum of token importance. We propose *mixSGA*, a novel mixture-of-expert (MoE) approach that dynamically optimizes token-wise computation and memory allocation. Unlike prior approaches, *mixSGA* retains all tokens while adaptively routing them to specialized experts with varying KV group sizes, balancing granularity and efficiency. Our key novelties include: **(1)** a token-wise expert-choice routing mechanism guided by learned importance scores, enabling proportional resource allocation without token discard; **(2)** weight-sharing across grouped attention projections to minimize parameter overhead; and **(3)** an auxiliary loss to ensure one-hot routing decisions for training-inference consistency in CLMs. Extensive evaluations across Llama3, TinyLlama, OPT, and Gemma2 model families show *mixSGA*'s superiority over static baselines. On instruction-following and continued pretraining tasks, *mixSGA* achieves higher ROUGE-L and lower perplexity under the same KV budgets.

1 Introduction

Transformer architectures have emerged as the backbone of modern deep learning, powering state-of-the-art advancements across diverse fields such as natural language processing (Vaswani et al., 2017), computer vision (Alexey, 2020), reinforcement learning (Parisotto and Salakhutdinov, 2021) and beyond (Le et al., 2020; Chen et al., 2023).



(a) Change for each token. (b) Change distribution.

Figure 1: **Token importance is dynamic and has a wide spectrum.** We replace each token's forward pass with multi-query attention on Llama3.1-8b by averaging key and value states across heads, and see the sequence perplexity changes on a sample sequence of WikiText-2.

However, their self-attention mechanism, while effective, suffers from quadratic computational and memory costs with respect to the sequence length, posing scalability challenges (Tay et al., 2022).

Efforts towards addressing these challenges have largely focused on improving the efficiency of attention mechanisms. One line of methods seek to improve the design of the attention block. Grouped Query Attention (GQA) (Ainslie et al., 2023), for example, reduces computational overhead by clustering keys and values into coarse groups, which reduces the number of processed KV pairs. Nevertheless, GQA assumes static group sizes and allocates resources uniformly, disregarding variations in token importance. Some works have been devoted to optimize the memory footprint of the widely adopted KV cache (Waddington et al., 2013). Token-level approaches, such as DynamicKV (Zhou et al., 2024), introduce flexible KV cache allocation by prioritizing high-value tokens. However, these methods often involve rigid resource allocation strategies that neglect to fully exploit the significance of low-priority tokens.

Another promising line of work (Shazeer et al., 2017; Lepikhin et al., 2020) adopts MoEs to dynamically route tokens to a subset of experts, enabling efficient resource utilization. While these approaches achieve computational efficiency, they frequently suffer from imbalanced expert utilization.

*Corresponding author, xt.gao@siat.ac.cn.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 22902–22915
November 4-9, 2025

Moreover, their coarse-grained routing overlooks token-level variability, highlighting the need for finer-grained adaptivity in token-level resource allocation. In many cases, tokens deemed less important are outright discarded or receive minimal processing, which can lead to degraded performance for certain tasks.

Our work is motivated by the experimental findings presented in Figure 1, which reveal that token importance exhibits dynamic behavior and spans a wide spectrum. This observation naturally inspires the high-level idea of tailoring experts' token selection based on their importance. While this approach holds significant promise for efficiently leveraging the potential of token prioritization, it faces several critical challenges that must be addressed. First, current token-choice routing (TCR) approaches can result in unbalanced expert utilization, particularly challenging to tune for heterogeneous capacities of experts, as tokens may always prefer high capacity experts, posing the risk of collapsed routing mechanisms. Second, existing expert-choice routing (ECR) methods shifts imbalanced expert utilization to token utilization, where tokens may be assigned to multiple experts, while some tokens are ignored. Third, existing ECRs also introduce *training and inference disparities* in CLMs, where during the training or prefill phase, routings are made based on the complete sequence, whereas the decoding-phase routings are made based on past context.

To surmount these obstacles, we introduce *mixSGA*. Unlike prior work that discards less significant tokens, our method retains all tokens while dynamically allocating computation and memory resources proportionally to their importance. For the experts, we propose a weight-sharing mechanism across grouped attentions, allowing the model to remain lightweight while dynamically scaling based on token significance. To overcome the routing disparity between prefill and decode stages, we propose a layer-wise auxiliary loss that encourages routing consistency.

Our contributions are summarized as follows:

- **The *mixSGA* Framework**: *mixSGA* integrates dynamic token-wise routing with KV attention head grouping, enabling adaptive computational/memory allocation without discarding tokens. It also uses weight-sharing for parameter efficiency.
- **Autoregressive Expert-Choice Routing**: We propose a novel past-context routing mechanism with an auxiliary loss to ensure prefill-decode consistency in CLMs. It also enables flexible tuning of individual expert capacities.
- **Broad Empirical Validation**: We demonstrates superior efficiency and performance over static and dynamic baselines across OPT, Llama3 and Gemma2 models on diverse instruction-following and continued pretraining benchmarks.

2 Related Work

2.1 KV Cache Management

KV cache optimization enhances the memory efficiency of CLMs (Waddington et al., 2013). Recent methods such as PyramidKV (Cai et al., 2024), DynamicKV (Zhou et al., 2024), H2O (Zhang et al., 2024b) and NACL (Chen et al., 2024a) aim to reduce memory footprint, typically by prioritizing high-utility tokens based on heuristics, random, or learned importance, and evict those deemed less critical to stay within a constrained memory budget. Furthermore, methods like SnapKV (Li et al., 2024) and FastGen (Ge et al., 2024) focus on pattern- and importance-based token selection to optimize KV cache efficiency during inference.

Despite these improvements, such methods often rely on predefined grouping mechanisms, which may not fully capture token-level variability. As a result, they can overlook the nuanced importance of individual tokens, limiting their ability to optimize resource utilization effectively. In addition, inevitably remove tokens from the attention context, which may lead to degraded performance on fine-grained contextual understanding. By contrast, *mixSGA* adaptively allocates smaller KV cache sizes to less critical tokens without evicting them, striking a balance between efficiency and preserving full contextual integrity. It also preserves all tokens, and instead of hard eviction, adaptively allocates memory and compute resources in proportion to token importance. This design ensures that even less critical tokens retain their contextual influence, offering a more flexible and context-preserving alternative to hard cache eviction.

2.2 MoE Routing Strategies and Challenges

MoEs provide a scalable approach to increase model capacity without proportionally increasing computational costs (Shazeer et al., 2017; Lepikhin et al., 2020). In token-choice routing (TCR) MoE

models such as GShard (Lepikhin et al., 2020) and Switch Transformer (Fedus et al., 2022), each token independently select an expert or a subset of experts for resource-efficient computation. As TCR allows each token to independently select an expert, it may suffer from imbalanced expert utilization, inefficient resource allocation, and potentially unstable training dynamics. This is especially problematic when experts have heterogeneous capacities, as tokens may favor experts with higher capacities, making it challenging to balance expert loads. Expert-choice routing (ECR) (Zhou et al., 2022) enables experts to select tokens for processing while explicitly defining their capacities, improving load balancing and resource utilization. Despite its advantages over TCR, ECR presents two significant challenges in the context of CLMs: **(1)** it requires access to the entire input sequence to make routing decisions, which is incompatible with CLMs that rely solely on past tokens to predict the next token; and **(2)** it shifts the issue of *imbalanced* expert utilization to *imbalanced* token utilization, where some tokens may remain unprocessed by any expert, while others may be redundantly processed by multiple experts.

Our work differentiates itself from existing TCR and ECR methods by introducing a routing mechanism specifically designed for CLMs. This mechanism evaluates token significance based on partial sequence context, enables dynamic expert selection, and ensures prefill-decode routing consistency in decoder-only architectures. Additionally, our method accommodates experts with heterogeneous capacity, delivering fine-grained resource allocation and improved efficiency on computation and memory costs.

2.3 Grouped Attention Methods

Grouped Query Attention (GQA) (Ainslie et al., 2023) reduces the computational and memory costs by merging keys and values into larger groups, reducing the number of KV pairs processed during attention. This can lead to inefficiencies when token importance varies significantly, as structured merging fails to prioritize tokens critical to the task. Decoupled-Head Attention (DHA) (Chen et al., 2024b) adaptively merges attention heads across layers, while Align Attention (Jin et al., 2024b) uses ℓ_0 regularization to convert multi-head attention into group-based formulations. Mixture-of-Head Attention (MoH) (Jin et al., 2024a) reformulates attention heads as experts in a MoE, employing TCR for sparse head activation. Cross-layer Attention (CLA) (Brandon et al., 2024) merges key and value projectors across adjacent layers, representing one approach to cross-structure KV sharing that reduces parameter count and memory usage. (Wu et al., 2025) further generalizes CLA to provide a systematic study of static cross-layer KV sharing techniques, highlighting their efficiency benefit. ReAttention (Liu et al., 2025) offers a training-free approach to token selection based on attention scores as importance proxies, enabling infinite context with finite attention scope through dynamic token prioritization.

While these approaches learn to enhance structural efficiency, they rely on static group sizes for attention heads during inference, assuming uniform token importance and lacking fine-grained, token-level adaptability. Additionally, these methods do not support heterogeneous expert configurations with varying group sizes, limiting their adaptability.

In contrast, *mixSGA* integrates the strengths of grouped attention and token-level adaptivity by dynamically routing each token to weight-shared experts with heterogeneous KV configurations, based on learned token importance. Unlike prior methods, *mixSGA* retains all tokens, ensuring no loss of contextual information, while adaptively allocating computational and memory resources at both group and token levels.

3 The *mixSGA* Method

mixSGA, mixture of weight-shared grouped attention experts, combines dynamic token-wise expert assignment with token-level KV optimization to achieve efficient attention computation and minimize KV memory. This section elaborates on the key components, including the routing mechanism for expert selection, the mixture of weight-shared KV grouping experts, and the auxiliary loss designed to improve prefill/decode consistency.

3.1 Prefill and Training Phase Routing

Token-to-expert mapping score function Given an input sequence $X \in \mathbb{R}^{L \times D}$, where L is the sequence length and D is the embedding dimension, we define the following token-to-expert mapping scoring function for all tokens, a trainable linear layer $\mathsf{S}\colon \mathbb{R}^{L \times D} \to \mathbb{R}^{L \times E}$ with weight $\boldsymbol{\phi} \in \mathbb{R}^{D \times E}$ and bias $\boldsymbol{\beta} \in \mathbb{R}^{E}$, where E is the number of ex-

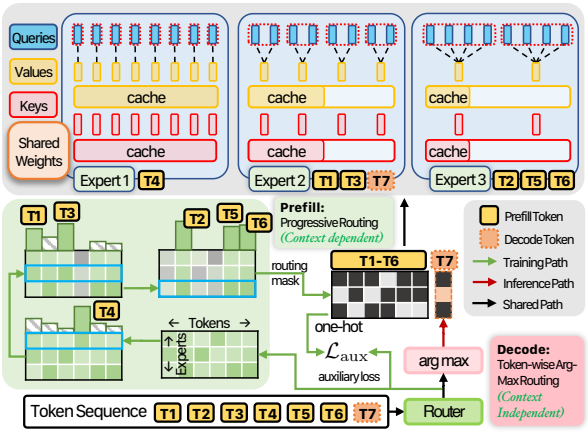


Figure 2: High-level overview of *mixSGA*. During training, the router learns to compute assignment scores for each token-expert pair. These scores are utilized to sequentially route ρ_e tokens to the e^{th} expert, ordered by their computational and memory costs, while leaving the remaining tokens to be routed to other experts. The experts consist of a set of key and value projections that generate state representations at varying levels of granularity. This process ensures a unique routing assignment for each token, which is subsequently used to encourage the router to produce one-hot decisions through an auxiliary loss. During decoding, each token independently selects its corresponding expert through $\arg\max$.

perts:

$$\mathsf{S}(\mathbf{x}) = \sigma(\mathbf{x}\boldsymbol{\phi} + \boldsymbol{\beta}), \tag{1}$$

and $\sigma(\cdot)$ is the sigmoid function. The sigmoid activation ensures bounded scores within $[0, 1]$, avoiding additional normalization during training.

MoEs with Heterogeneous Capacities To facilitate downstream KV cache optimization, our method employs a routing mechanism that dynamically assigns tokens to experts based on predefined capacity ratios. These ratios regulate token distribution among experts, aligning with memory and computational constraints. Assume that we have E experts, where with predefined capacity ratios for each expert $\boldsymbol{\rho} = \{\rho_1, \rho_2, \ldots, \rho_E\}$, representing the fraction of tokens it processes. The capacity ratios lie in the range $[0, 1]$, and are normalized such that the sum of all ratios is 1, *i.e.*, $\sum_{e=1}^{E} \rho_e = 1$. During training, our token-to-expert routing thus takes the scoring function output $\mathsf{S}(\mathbf{x})$ and greedily assigns tokens to experts progressively. For the e^{th} expert, we assign tokens based on the top-$\lceil \rho_e L \rceil$ scores, and route the remaining tokens to the next $(e+1^{\text{th}})$ expert. Formally, it employs the following sparse masking function $\mathbf{m}_e \colon \mathbb{R}^{L\times E} \to \{0,1\}^{L\times E}$, where:

$$\mathbf{m}_e(\mathbf{x}) = \mathbf{1}\Big[\mathsf{top}_{\lceil \rho_e L \rceil}\Big(\mathsf{S}(\mathbf{x}) \textstyle\prod_{i=1}^{e-1}(1 - \mathbf{m}_i(\mathbf{x}))\Big)\Big], \tag{2}$$

and $\mathbf{1}$ denotes the element-wise indicator function, producing 1 for the top-$\lceil \rho_e L \rceil$ scores, and 0 otherwise. Note $\mathbf{m}_e(\mathbf{x})$ depends on the masks of preceding experts, ensuring that tokens previously assigned to other experts are skipped, thereby guaranteeing an exclusive mapping of each token to a single expert.

3.2 Decode-Phase Routing

The preceding paragraphs outline the training/prefill phase of our token-wise ECR mechanism, which operates on a sequence of tokens as input. However, this routing approach cannot be directly applied to the decoding phase of CLMs, where tokens are generated iteratively, this means that we need a different routing strategy for the decode phase.

A key advantage of eq. (2) is that it ensures exclusive expert mapping for each token, resulting in $\sum_{e=1}^{E} \mathbf{m}_e(\mathbf{x})$ being a one-hot vector for each token. If we encourage both phases to have the same expert assignments, we can simply use $\arg\max \mathsf{S}(\mathbf{x})$ to determine the expert assignment during decoding. During the decoding phase, expert assignments for the next token are then determined by simply taking the $\arg\max$ of the scoring function, *i.e.*, This approach eliminates the need for a top-k operation over the entire input sequence, which is infeasible during decoding. To summarize, the prefill and decode phases use the following routing functions:

$$\begin{aligned} \mathbf{T}_{\text{prefill}}(\mathbf{x}) &= \textstyle\sum_{e=1}^{E} \mathbf{m}_e(\mathbf{x}), \\ \mathbf{T}_{\text{decode}}(\mathbf{x}) &= \mathbf{1}[\arg\max(\mathsf{S}(\mathbf{x})) = e]. \end{aligned} \tag{3}$$

3.3 Prefill-Decode Consistency Loss

To align $\arg\max \mathsf{S}(\mathbf{x})$ with the expert assignment $\arg\max \mathbf{T}_{\text{prefill}}(\mathbf{x})$, we introduce the following consistency loss where $\arg\max \mathbf{T}_{\text{prefill}}(\mathbf{x})$ extracts the expert index assigned to each token:

$$\mathcal{L}_{\text{aux}}(\mathbf{x}) = \mathcal{L}^{\text{sce}}\big(\mathsf{S}(\mathbf{x}), \arg\max \mathbf{T}(\mathbf{x})\big). \tag{4}$$

The total training loss for the model combines the primary language-modeling loss $\mathcal{L}_{\text{model}}$ with the auxiliary loss $\mathcal{L}_{\text{aux}}(\mathbf{x}^{(l)})$ applied across all layers $l \in \{1, \ldots, L\}$, weighted by α:

$$\mathcal{L} = \mathcal{L}_{\text{model}} + \tfrac{\alpha}{L} \textstyle\sum_{l=1}^{L} \mathcal{L}_{\text{aux}}(\mathbf{x}^{(l)}). \tag{5}$$

3.4 Mixture of Weight-Shared GQAs

KV projection Building on the token-wise expert assignment described earlier, we extend the attention mechanism by introducing a mixture of *weight-shared* GQAs. Each expert processes its assigned tokens independently and maintains KV caches tailored to its group configuration, achieving an efficient trade-off between computation and memory. Assuming a pretrained attention layer with $(\mathbf{w}^{\mathrm{k}}, \mathbf{w}^{\mathrm{v}}) \in \mathbb{R}^{D\times D}, (\mathbf{b}^{\mathrm{k}}, \mathbf{b}^{\mathrm{v}}) \in \mathbb{R}^{H\times D}$, key and value weights and biases, where D is the embedding dimension, we first define the following key and value projection $p_h^j \colon \mathbb{R}^{L\times D} \to \mathbb{R}^{H\times L\times (D/H)}$ for the h^{th} head, where $j \in \{\mathrm{k}, \mathrm{v}\}$, $h \in \{1, \ldots, H\}$, and:

$$P^j(\mathbf{x})_h = \left(\mathbf{w}^j \mathbf{x}^\top + \mathbf{b}^j\right)_{\left[\frac{D(h-1)}{H}+1 \,:\, \frac{Dh}{H}\right]}, \quad (6)$$

Here, the subscript $\mathbf{z}_{[a:b]}$ denotes the slice operation which selects elements from the first dimension of $\mathbf{z}$ ranging from a to b.

KV grouping Inspired by GQA (Ainslie et al., 2023), for each expert f_e^j, we design the following mechanism to reduce the number of projected KV heads from H to $H/2^e$ groups of size 2^e by taking the average of the corresponding grouped heads. Specifically, for each grouping $g \in G_e$ of expert e, we have $f_e^j \colon \mathbb{R}^{H\times L\times (D/H)} \to \mathbb{R}^{H/2^e\times L\times (D/H)}$:

$$f_{e,g}^j(\mathbf{x}) = {}^1\!/_{2^e} \textstyle\sum_{h\in g} p^j(\mathbf{x})_h, \quad (7)$$

where G_e groups a range of heads by size 2^e, For example, if $H = 4$ and $E = 3$, we have $G_1 = \{\{1\}, \{2\}, \{3\}, \{4\}\}$, $G_2 = \{\{1, 2\}, \{3, 4\}\}$, and $G_3 = \{\{1, 2, 3, 4\}\}$. Notably to ensure parameter efficiency, we share the same key and value weights across all experts. While for mathematical clarity we define the mean operation over the projected heads, one can easily instead aggregate the KV projection weights before applying the projection operation to achieve the same effect.

Due to this grouping, the total KV cache size is thus adjusted based on which expert processes the token, with the cache size of the e^{th} expert being $H/2^e$ of the original size.

Attention computation Before computing the attention, for expert e we match the KV head counts $H/2^e$ with the query head count H by repeating the KV heads 2^e times using $h_{e,g}^j \colon \mathbb{R}^{H/2^e\times L\times (D/H)} \to \mathbb{R}^{H\times L\times (D/H)}$:

$$h_{e,g}^j(\mathbf{x}) = f_{e,g}^j(\mathbf{x}) \otimes \mathbf{1}_{2^e}. \quad (8)$$

where $\otimes$ denotes the outer product, and $\mathbf{1}_{2^e}$ is a vector of ones of size 2^e. Finally, the overall result computed by the MoE is:

$$h^j(\mathbf{x}) = \textstyle\sum_{e=1}^{E} \mathbf{m}_e(\mathbf{x}) \odot h_{e,g}^j(\mathbf{x}). \quad (9)$$

It is noteworthy that since $\mathbf{m}_e(\mathbf{x})$ is sparse and has token-wise exclusive expert assignment, the most of the $h_{e,g}^j(\mathbf{x})$ are zeroed out and skipped. In practice, this is carried out efficiently with scatter and gather tensor operations.

The attention computation is then performed following the standard scaled dot-product attention mechanism, where $q(\mathbf{x})$ is the original query projection:

$$a(\mathbf{x}) = \mathsf{softmax}\Big(q(\mathbf{x}) h^{\mathrm{k}}(\mathbf{x})^\top / \sqrt{D}\Big) h^{\mathrm{v}}(\mathbf{x}). \quad (10)$$

Expert Allocation for Memory Efficiency *mixSGA* computes varying KV sizes per token thanks to its dynamic routing mechanism assigning tokens to experts of different group sizes. For $E = 3$ experts, the group sizes are $1, 2, 4$ respectively, and the head counts are thus $H, H/2, H/4$. This means that on average given a ratio of $a : b : c$, all tokens require $(a + b/2 + c/4)/(a + b + c)$ of the original KV size. Along with the KV cache, we also store a single index value for each token to track expert assignment.

Integration with KV eviction Although *mixSGA* dynamically allocates per-token KV sizes, it remains fully compatible with KV eviction such as H2O (Zhang et al., 2024b) and NACL (Chen et al., 2024a) to further reduce memory usage.

4 Experiments

4.1 Supervised Fine-tuning

Models and methods We evaluate *mixSGA* on the following CLMs: OPT-{125m,355m} (Zhang et al., 2022), Llama3.1-8b, Llama3.2-{1b,3b} (Touvron et al., 2023), and Gemma2-2b (Gemma Team et al., 2024), covering various model sizes and architectures. As a default baseline, we implement a GQA-variant of the original models which forms KV head groups of size 2 by initializing the KV projection matrices with the mean of the group. For fair comparisons, *mixSGA* is configured with expert density ratios which maintain the same active KV head counts, and thus the same KV size, as GQA. It keeps the pretrained weights from the original models, and randomly initializes the newly added

routing weights with He initialization (He et al., 2015) and biases with zeros.

Training and evaluation setup We fine-tune the modified models on the Dolly-15k instruction-following dataset (Conover et al., 2023) with 14,000 training samples, and evaluate their performance on 5 conversational datasets: Dolly (DL, 500 testing samples from Dolly-15k), Self-Instruct (SI) (Wang et al., 2023), Vicuna (VC) (Chiang et al., 2023), Super-Natural Instructions (SN) (Wang et al., 2022), and Unnatural Instruction (UI) (Honovich et al., 2023). In addition to the ROUGE-L (R-L) scores, which measure the longest common sub-sequence between generated and reference answers, we also evaluate all answers to the queries using DeepSeek-V3 (DeepSeek-AI et al., 2024) to provide feedback scores ranging from 0 to 10. The template to generate feedback is provided in Section A. All hyperparameter configurations are provided in Section A for reproducibility.

Main Results For supervised fine-tuning tasks, we initiate our approach by conducting a grid search on a smaller model (OPT-355M) to determine the optimal expert density ratios, incrementing by 0.1 while maintaining the total KV size constant at 50% of the original model. Our results show that allocating tokens as 30% to experts with a group size of 1, 10% to size 2, and 60% to size 4 optimizes performance across most metrics. This 3:1:6 ratio consistently outperforms other configurations. As shown in Table 1, *mixSGA* consistently outperforms GQA across various benchmarks and model sizes. These results demonstrate *mixSGA*'s ability to dynamically allocate resources and improve performance over static GQA baselines.

4.2 Continued Pretraining

Models and methods We investigate *mixSGA*'s ability in continued pretraining on additional corpus. We used a TinyLlama-1.1B model (Zhang et al., 2024a), which was pretrained on SlimPajama (Soboleva et al., 2023) and StarCoder (Li et al., 2023) and adapted its weights to GQA with group size set to 2, CLA (Brandon et al., 2024), and *mixSGA*. Both CLA and *mixSGA* aligns the same KV cache size as the GQA baseline.

Training and evaluation setup We train the models with each method applied for one epoch of MiniPile (Kaddour, 2023), which amounts to 1.6 billion tokens. We use a diverse set of benchmarks to evaluate the resulting models: HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), Winogrande (Sakaguchi et al., 2019), ARC-Easy (ARC-E), ARC-Challenge (ARC-C) (Clark et al., 2018), and the perplexity on Wikitext-2 (Merity et al., 2016). For the first six tasks, higher accuracy (%) indicates better performance, while lower perplexity on Wikitext-2 reflects stronger language modeling ability. The training and evaluation details are provided in Section A.

Main Results In our continued pretraining setting, the key challenge is to recover previously learned capabilities of the model with a fraction of data drawn from a distribution domain similar to the original pretraining data. As shown in Table 2, *mixSGA* consistently demonstrates competitive or superior accuracy on most benchmarks. It attains 37.00% on HellaSwag and 56.30% on Winogrande, both surpassing GQA (group size = 2) and CLA. Performance on ARC-C (25.17%) also exceeds that of the baselines, highlighting *mixSGA*'s strength in handling more challenging tasks. *mixSGA* also shows a clear advantage in Wikitext-2 PPL, delivering the lowest value (20.46) among all models. To summarize, these results indicate that *mixSGA* can enable the model to preserve previously acquired knowledge, as applying it to existing models does not impact their pretrained weights.

***mixSGA* compliments cache eviction better** To investigate the compatibility of *mixSGA* with dynamic KV cache eviction strategies, we conduct a set of controlled experiments by integrating H2O (Zhang et al., 2024b) with both GQA and *mixSGA* on Gemma2-2b. These experiments are designed to evaluate whether the orthogonal benefits of token-level eviction and token-wise KV allocation can be combined effectively. Both GQA and *mixSGA* are configured to operate under a shared KV budget of 50% of the original size, with H2O applied as a post-processing eviction method to further compress memory. We vary the H2O keep ratio from 80% down to 20% to simulate increasing memory pressure. The results, shown in Table 3, demonstrate that *mixSGA* consistently outperforms GQA across all compression levels. This validates that *mixSGA* not only preserves the contextual coherence lost in aggressive token eviction, but also enhances the effectiveness of cache compression when used in conjunction with existing methods like H2O. The results demonstrate that integrating *mixSGA* with cache eviction policies further

Architecture	Method	Expert Ratios	Dolly R-L	Dolly DSv3	Self-Instruct R-L	Self-Instruct DSv3	Vicuna R-L	Vicuna DSv3	SN R-L	SN DSv3	UN R-L	Avg. R-L
OPT-125m	GQA	0:1:0	17.70	2.19	6.93	2.35	12.57	1.81	8.33	2.28	10.56	11.22
	mixSGA	1:1:2	19.80	3.22	6.89	3.22	12.33	2.16	11.21	2.65	13.77	**12.80**
	mixSGA	3:1:6	17.65	3.25	7.75	2.96	10.79	2.48	8.77	2.99	12.83	11.56
OPT-355m	GQA	0:1:0	21.11	3.19	7.88	2.09	10.86	1.75	10.51	2.27	12.77	12.63
	mixSGA	1:1:2	17.21	3.36	9.19	3.06	10.55	2.10	11.03	2.72	14.67	12.53
	mixSGA	3:1:6	21.43	3.48	8.68	3.57	12.19	2.64	11.34	2.85	14.90	**13.71**
Llama3.2-1B	GQA	0:1:0	20.09	3.45	7.90	3.17	13.21	2.41	12.43	2.74	14.50	13.63
	mixSGA	1:1:2	18.87	4.02	9.01	3.68	10.93	2.97	14.09	3.33	17.70	14.12
	mixSGA	3:1:6	20.11	4.05	10.03	3.65	14.41	2.99	15.52	3.24	20.42	**16.10**
Llama3.2-3B	GQA	0:1:0	23.26	4.19	9.95	3.45	14.93	3.54	15.73	3.68	18.23	16.42
	mixSGA	1:1:2	25.49	5.08	11.20	4.66	15.34	4.29	19.46	4.12	24.19	19.14
	mixSGA	3:1:6	25.57	5.23	13.13	4.43	14.61	3.86	18.32	4.18	24.24	**19.17**
Llama3.1-8B	GQA	0:1:0	27.40	4.85	11.60	4.60	15.36	3.43	21.72	4.22	23.75	19.97
	mixSGA	1:1:2	26.50	6.40	17.22	6.01	15.06	4.90	32.52	6.43	33.91	25.04
	mixSGA	3:1:6	28.47	6.97	17.30	5.93	19.19	4.88	35.81	6.68	34.62	**27.08**
Gemma2-2B	GQA	0:1:0	25.68	5.64	10.43	3.73	16.53	4.25	20.00	4.27	23.68	19.26
	mixSGA	1:1:2	24.79	6.18	16.08	5.37	12.70	5.26	26.01	5.55	27.39	21.39
	mixSGA	3:1:6	26.15	6.25	17.36	5.62	14.47	5.40	26.82	5.98	28.71	**22.70**

Table 1: Supervised fine-tuning of a range of models on the Dolly-15k instruction-following dataset (Conover et al., 2023). Evaluation includes ROUGE-L (R-L) and DeepSeek-V3 feedback scores (DSv3) on 5 conversational datasets. *mixSGA* demonstrates consistent improvements over GQA baselines with the same KV budgets. The "Avg. R-L" column shows the average ROUGE-L scores across all datasets.

Method	↑HS	↑PI	↑WG	↑AE	↑AC	↑Avg.	↓WT
GQA	36.70	**70.62**	55.90	54.92	23.89	48.41	22.66
CLA	35.90	68.82	55.40	**55.47**	23.81	47.88	24.62
mixSGA	**37.00**	69.53	**56.30**	54.84	**25.17**	**48.57**	**20.46**

Table 2: Continued pretraining on TinyLlama-1.1B with MiniPile. (↑: higher is better, ↓: lower is better, HS: HellaSwag, PI: PIQA, WG: Winogrande, AE: ARC-E, AC: ARC-C, WT: Wikitext-2.)

KR	Method	↑HS	↑PI	↑WG	↑AE	↑AC	↑Avg.	↓WT
80%	GQA	**36.8**	69.58	55.04	53.41	23.81	47.73	22.70
	mixSGA	36.5	**70.02**	**55.33**	**53.91**	**25.77**	**48.31**	**20.53**
60%	GQA	36.3	68.62	**54.06**	53.17	23.63	47.16	22.72
	mixSGA	**36.5**	**70.18**	53.51	**53.66**	**25.34**	**47.84**	**20.63**
40%	GQA	**36.1**	67.94	53.27	52.19	24.40	46.78	22.80
	mixSGA	35.8	**69.10**	**54.14**	**52.27**	**24.92**	**47.25**	**20.98**
20%	GQA	35.2	63.18	49.96	44.36	21.33	42.81	23.56
	mixSGA	**35.5**	**64.80**	**50.75**	**44.51**	**22.53**	**43.62**	**22.19**

Table 3: Integrating H2O with various KV keep ratios on Gemma2-2b. *mixSGA* consistently outperforms GQA across most tasks and H2O KV keep ratios (KR).

enhances its applicability in inference tasks while reducing KV memory footprint.

4.3 Ablation Studies

To comprehensively attribute the impact of each component in *mixSGA*, we perform ablation studies under three key aspects by varying the following: expert density ratios and expert counts, and the auxiliary loss with learned routing mechanism. Experiments in Tables 4 and 5 and Table 6 are conducted

Ratios	DL	SI	VC	SN	UN	Avg.
1:9:2	18.41	7.80	11.49	9.78	13.53	12.20
1:6:2	**19.60**	8.47	**12.14**	11.53	14.79	13.31
1:1:2	18.87	**9.01**	10.93	**14.09**	**17.70**	**14.12**

Table 4: Effect of different expert group ratios under the same KV size budget (50%) for Llama3.2-1B. Results are reported for ROUGE-L across multiple benchmarks. (DL: Dolly Evaluation, SI: Self-Instruct, VC: Vicuna, SN: Super-Natural Instructions, UN: Unnatural Instructions, Avg.: Average ROUGE-L across benchmarks)

on Llama3.2-1b and Gemma2-2B respectively, following the same setup in Section 4.1. We provide detailed analyses of the results below.

Varying the expert ratios Table 4 investigates the effect of varying density ratios among experts while keeping a fixed KV size budget of 50%. We systematically increase the ratio assigned to the 2nd expert in a group of size 2, testing configurations from 1:1:2 to 1:9:2, Our results reveals that evaluation metrics improve as the 2nd expert's ratio decreases, indicating a preference for allocating more tokens to the 1st and 3rd experts. This suggests the model prioritizes assigning important tokens to the 1st expert, which retains the original model's KV projection weights, while routing less significant tokens to the smallest (3rd) expert.

Varying the expert counts In Table 5, we investigate the influence of employing 2-3 experts

Ratios	DL	SI	VC	SN	UN	Avg.
3:1:6	**20.11**	**10.03**	**14.41**	**15.52**	**20.42**	**16.10**
3:4:0	18.11	9.12	15.47	12.78	16.25	14.35
1:1:2	**18.87**	**9.01**	**10.93**	**14.09**	**17.70**	**14.12**
1:2:0	13.94	6.83	14.75	8.80	11.06	11.08

Table 5: Effect of redistributing KV cache across tokens under fixed KV size (50% of the original model) for Llama3.2-1B. Results are reported for ROUGE-L following the style in Table 4.

Ratios	DL	SI	VC	SN	Avg.
mixSGA	**26.15**	**17.36**	**14.47**	**26.82**	**21.20**
Random router	24.65	12.56	12.24	20.98	17.68
No auxiliary loss	10.07	6.22	4.56	8.54	7.35

Table 6: Ablation study on the effect of auxiliary loss and learned routing for Gemma2-2B with 3:1:6 expert ratios under a 50% KV budget. Results report ROUGE-L scores across benchmarks.

while maintaining a fixed total KV budget of 50%. Specifically, we compare configurations with 3:1:6, 3:4:0, 1:1:2, 1:2:0 ratios. Here, a value of 0 for the 3rd expert indicates its exclusion from the model. Remarkably, we observe that introducing a 3rd expert significantly enhances performance, achieving an average ROUGE-L score improvement of up to 3.12 across all benchmarks. Given the variable information content of individual tokens, this finding highlights the critical role of the 3rd expert in capturing less crucial tokens within the input sequence, allowing the other two experts to focus on processing more significant ones.

Learned Routing To assess the auxiliary loss and learned routing mechanism, we conduct experiments on Gemma2-2B with a 3:1:6 expert ratio, following Section 4.1. As shown in Table 6, we found that removing the auxiliary loss leads to inconsistent routing between prefill and decoding, resulting in near-random expert assignments (0.3458:0.3306:0.3236 for the 3 experts on Dolly), as the model never learns to route according to expert density ratios. This causes a severe average ROUGE-L drop (21.20 to 7.35). We also found that replacing the learned router with a router that randomly assigns experts per the 3:1:6 ratio degrades performance.

Varying KV Budgets To evaluate the influence of varying KV budgets on language modeling ability, we conducted comparative experiments involving *mixSGA*, GQA, and CLA across different KV budgets using the TinyLlama continued pretraining

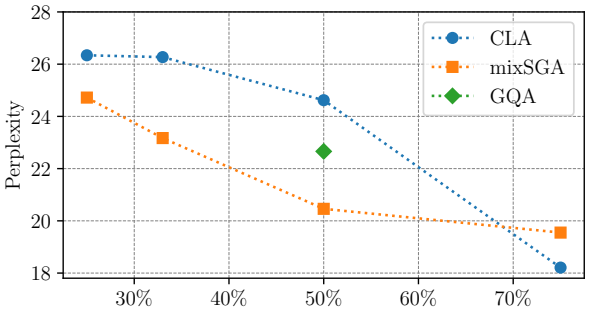


Figure 3: Comparing TinyLlama-1.1B continued pretraining with *mixSGA* and baselines (GQA and CLA) across varying KV size ratios. Lower perplexity indicates better language modeling performance.

task as outlined in Section 4.2. For *mixSGA*, the configurations were set as follows: 0:0:1 for a 25% KV budget, 1:1:8 targeting 35%, 3:1:6 for 50%, and 1:1:0 for 75%. CLA was configured to align with these KV sizes. Given that the TinyLlama-1.1B attention module comprises only 4 heads, GQA could thus only employ a group size of 2 to achieve a 50% KV budget.

As illustrated in Figure 3, *mixSGA* consistently achieves superior performance, manifesting in lower perplexity across most KV budgets compared to the baselines. Notably, CLA experiences a pronounced increase in perplexity as the KV budget decreases, particularly below 50%, where its performance deteriorates significantly. This highlights the challenges faced by static approaches in maintaining accuracy under constrained KV budgets. Conversely, *mixSGA* exhibits enhanced robustness, with lower perplexity levels across various budgets, suggesting that its dynamic token routing mechanism enables more effective resource allocation. This adaptability underscores its capability to deliver improved language modeling performance, even under limited KV budgets.

5 Conclusion

This paper introduced *mixSGA*, a framework that combines dynamic token-wise expert-choice routing with attention grouping to optimize KV representations. By using weight-shared heterogeneous attention experts, *mixSGA* adaptively allocates resources based on token importance. Our experiments with Llama3, OPT, and Gemma2 model families show that *mixSGA* outperforms baseline approaches in computational efficiency and task performance, with improved scalability in resource-constrained scenarios. The routing mechanism of *mixSGA* ensures consistency between prefill and de-

code phases. Overall, *mixSGA* offers a scalable and efficient solution for dynamic KV optimization.

6 Limitations and Risks

Our method assigns tokens to diverse experts within each layer to improve efficiency. However, we only configure the same capacity ratios across different layers, which may ignore diverse token importance across lower and deeper layers. Therefore, our method can be further improved with a global importance metric, automatically configuring capacity ratios tailored for each layer. This potentially offers versatility and flexibility for better resource utilization. Moreover, mixSGA's efficiency gains in computational and memory costs enable resource-constrained researchers to leverage advanced LLMs, fostering innovation in fields like NLP and healthcare while supporting sustainable AI through reduced energy consumption. This democratization of access can accelerate scientific progress and broaden AI's societal benefits. However, these efficiencies could lower barriers for harmful uses, such as generating misinformation or amplifying biases in routing decisions, and may introduce vulnerabilities like slowdowns from adversarial inputs. Future work may incorporate fairness-aware routing and adversarial robustness to ensure ethical deployment, aligning technological advances with responsible AI practices.

Acknowledgments

This work is supported in part by the National Key R&D Program of China (2023YFC3321600), National Natural Science Foundation of China (62376263 and 62372443), Guangdong Basic and Applied Basic Research Foundation (2023B1515130002), Natural Science Foundation of Guangdong (2024A1515030209 and 2024A1515011970), Shenzhen Science and Technology Innovation Commission (JCYJ20230807140507015 and JCYJ20220531100804009). This work is also supported in part by Technology Development Fund of Macao S.A.R (FDCT) under 0123/2022/AFJ and FDCT 0081/2022/A2, and was carried out in part at SICC, which is supported by SKL-IOTSC, University of Macau.

References

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 *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Dosovitskiy Alexey. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv: 2010.11929*.

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

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. 2024. Reducing transformer key-value cache size with cross-layer attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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

Hailin Chen, Amrita Saha, Steven Hoi, and Shafiq Joty. 2023. Personalized distillation: Empowering open-sourced llms with adaptive learning for code generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6737–6749.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. 2024a. Nacl: A general and effective kv cache eviction framework for llms at inference time. *Preprint*, arXiv:2408.03675.

Yilong Chen, Linhao Zhang, Junyuan Shang, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, and Yu Sun. 2024b. DHA: Learning decoupled-head attention from transformer checkpoints via adaptive heads fusion. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. Free dolly: Introducing the world's first truly open instruction-tuned llm.

DeepSeek-AI et al. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Preprint*, arXiv:2101.03961.

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. 2021. A framework for few-shot language model evaluation.

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. *Preprint*, arXiv:2310.01801.

Gemma Team et al. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Preprint*, arXiv:1502.01852.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2023. Unnatural instructions: Tuning language models with (almost) no human labor. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14409–14428, Toronto, Canada. Association for Computational Linguistics.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. 2024a. Moh: Multi-head attention as mixture-of-head attention. *Preprint*, arXiv:2410.11842.

Qingyun Jin, Xiaohui Song, Feng Zhou, and Zengchang Qin. 2024b. Align attention heads before merging them: An effective way for converting mha to gqa. *Preprint*, arXiv:2412.20677.

Jean Kaddour. 2023. The minipile challenge for data-efficient language models. *Preprint*, arXiv:2304.08442.

Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. Distillm: Towards streamlined distillation for large language models. In *Forty-first International Conference on Machine Learning*.

Hang Le, Juan Pino, Changhan Wang, Jiatao Gu, Didier Schwab, and Laurent Besacier. 2020. Dual-decoder transformer for joint automatic speech recognition and multilingual speech translation. In *COLING 2020 (long paper)*.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *Preprint*, arXiv:2006.16668.

R Li, LB Allal, Y Zi, N Muennighoff, D Kocetkov, C Mou, M Marone, C Akiki, J Li, J Chim, et al. 2023. Starcoder: May the source be with you! *Transactions on machine learning research*.

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

Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Qipeng Guo, Yuerong Song, Kai Lv, Hang Yan, Linlin Li, Qun Liu, and Xipeng Qiu. 2025. ReAttention: Training-free infinite context with finite attention scope. In *The Thirteenth International Conference on Learning Representations*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Emilio Parisotto and Russ Salakhutdinov. 2021. Efficient transformers in reinforcement learning using actor-learner distillation. In *International Conference on Learning Representations*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

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. https://cerebras.ai/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2022. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6).

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Preprint*, arXiv:1706.03762.

Daniel Waddington, Juan Colmenares, Jilong Kuang, and Fengguang Song. 2013. Kv-cache: A scalable high-performance web-object cache for manycore. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 123–130. IEEE.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. *Preprint*, arXiv:2212.10560.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, and Xudong Shen. 2022. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

You Wu, Haoyi Wu, and Kewei Tu. 2025. A systematic study of cross-layer KV sharing for efficient LLM inference. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 396–403. Association for Computational Linguistics.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024a. TinyLlama: An open-source small language model. *Preprint*, arXiv:2401.02385.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2024b. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36.

Xiabin Zhou, Wenbin Wang, Minyan Zeng, Jiaxian Guo, Xuebo Liu, Li Shen, Min Zhang, and Liang Ding. 2024. DynamicKV: Task-aware adaptive kv cache compression for long context llms. *Preprint*, arXiv:2412.14838.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Y Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V Le, and James Laudon. 2022. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems*.

A Experimental Setup

Our experiments are conducted on open-sourced datasets. These datasets serve as artifacts for research purposes, which is aligned with the goal of our experimental evaluation. Specifically, Databricks-Dolly-15k dataset uses CC BY-SA 3.0 license. Wikitext-2 is available under the Creative Commons Attribution-ShareAlike License. The remaining datasets in lm-eval-harness are available under the MIT License.

A.1 Supervised Fine-Tuning Tasks

For the supervised fine-tuning tasks, we apply templates to both the training and test datasets, following the standard procedure described in (Gu et al., 2024; Ko et al.). All input text was standardized to ensure consistency and fairness across different models. For the DeepSeek-V3 feedback evaluation (DeepSeek-AI et al., 2024), we use the template shown in Figure 4, with a temperature coefficient set to 0.7 to balance the randomness and diversity of the generated outputs. We first construct the training data from the Databricks-Dolly-15k dataset (Conover et al., 2023), wherein we randomly select 14,000 samples for training and equally leave 500 samples for validation and testing, respectively.

As part of our baselines, we modify the original pretrained models by integrating them into a more advanced GQA setup. For models not originally including GQA results, we apply the GQA mechanism. In cases where the models already have GQA results, we replace them with a more compressed version of GQA, which offers stronger compression levels. This ensures that our baseline is consistently adapted for a fair comparison with the new methods.

We performed full parameter fine-tuning for the OPT model series (OPT-{125m, 355m}). The batch size was set to 32, and we used a cosine learning rate schedule. The learning rate was initially set to $5e^{-5}$ and decayed according to the cosine decay scheduler. The models were trained for 40 epochs to ensure sufficient fine-tuning.

The routing weights were initialized using He initialization (He et al., 2015). For the learning rate setup, we initialized the learning rate at $5e^{-5}$, and the learning rate decay followed the same cosine schedule. We used a batch size of 32 for both training and evaluation. We employed gradient accumulation to simulate a larger batch size without exceeding memory constraints.

A.2 Continued Pretraining Tasks

In the continued pretraining tasks, we fine-tune the TinyLlama-1.1b model using the MiniPile dataset (Kaddour, 2023), which contains 1.6 billion tokens. The pretraining weights for the TinyLlama model were originally trained on a much larger dataset containing 3 trillion tokens (Zhang et al., 2024a). To adapt it as our baseline, we reduce the number of KV heads by half and implement a deeper GQA configuration. This modification of pretrained weights degrades the original model's performance, as halving the KV heads impacts the model's integrity. Therefore, we perform continued pretraining on the 1.6 billion tokens of MiniPile to recover the model's performance and address this degradation. Note that for all methods, we use the same hyperparameter settings in continued pretraining experiments, as illustrated in Table 7 and Table 8, for fair comparison.

We train the TinyLlama-1.1B model for one epoch on the MiniPile dataset and then evaluate its performance using lm-eval-harness(Gao et al., 2021) framework in a zero-shot setting across several benchmarks, including HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), Winogrande (Sakaguchi et al., 2019), ARC-Easy (ARC-E), ARC-Challenge (ARC-C) (Clark et al., 2018), and perplexity on Wikitext-2 (Merity et al., 2016). These benchmarks present comprehensive assessment of the model's language modeling abilities and task-specific performance.

```
Please act as an impartial judge and evaluate the quality of the response provided by an AI assistant to the user question displayed below. Consider factors such as helpfulness, relevance, accuracy, depth, and creativity. While evaluating, focus on clarity, usefulness, and effort. Please rate the response on a scale of 1 to 10 by following this format: 'Rating: [[x.xx]]', for example: 'Rating: [[5.00]]'.
```

Figure 4: System prompt template for DeepSeek-V3 feedback evaluation.

B Computational Resources

The experiments were conducted on two types of server equipped with NVIDIA A100 and V100 GPUs, configured by different model sizes and precision types.

Model Size	1.1B
Max LR	2e-4
LR Scheduler	cosine
Optimizer	AdamW
β_1	0.9
β_2	0.95
Warmup Ratio	0.015
Weight Decay	0.1
Gradient Clipping	1.0
Precision	Bfloat16
Batch Size (tokens)	256K
Epochs	1
DataSet	MiniPile
GPU	A100

Table 7: Training Hyperparameters for Continued Pretraining (TinyLlama-1.1B)

Model Size	1.1B
Hidden Size	2048
Intermediate Size	5632
Max Trained Length	2048
# Layers	22
# Attention Heads	32
# KV Heads	4
RMS Norm eps	1e-5
Vocab Size	32000

Table 8: Model Hyperparameters for TinyLlama 1.1B

For the Llama 8B model, we used servers with 4 NVIDIA A100 GPUs (80GB) and Intel Xeon Gold 6230R processors with 104 CPU cores. We use bfloat16 (bf16) precision to align with the precision applied for pretraining and reduce memory burden.

For other Llama and Gemma model series, our experiments were performed on servers with 4 NVIDIA A100 GPUs (40GB) and the same CPU and precision configurations.

The OPT models (OPT-{125m, 355m}) were trained on 4 NVIDIA V100 GPUs (32GB), with Intel Xeon Gold 5118 processors with 48 CPU cores, using float32 (fp32) precision due to the V100's lack of hardware support for bfloat16 format.

C Additional Experiments

C.1 Compute and Memory Overheads

Table 9 presents the real compute and memory overheads of *mixSGA* compared to GQA. It shows that *mixSGA* incurs a marginal increase in FLOPs and

Metrics	Method	O-125m	L3.2-1b	G2-2b
#Params	GQA	118m	1.22b	2.55b
	mixSGA	125m	1.24b	2.61b
#FLOPs	GQA	94.8k	113k	237k
	mixSGA	94.9k	113k	237k
KV size per token (bytes)	GQA	36,864	32,768	106,496
	mixSGA	36,867	32,772	106,502

Table 9: Comparison of parameter counts, FLOPs, memory usage, and time for GQA and *mixSGA* under the same KV size budget (50%). (O-125m: OPT-125m, L3.1-8b: Llama3.1-8b, L3.2-1b: Llama3.2-1b, G2-2b: Gemma2-2b)

Method	L3.2-1b	L3.2-3b	L3.1-8b
GQA	59.75	40.16	26.77
mixSGA	57.28	38.65	25.92
Δ (%)	4.13	3.75	3.17

Table 10: Decoding throughput (tokens/s) of GQA and *mixSGA* under a 50% KV budget, with batch size 1, 10,000 tokens generated, and averaged over five trials. Higher is better. (L3.2-1b: Llama3.2-1b, L3.2-3b: Llama3.2-3b, L3.1-8b: Llama3.1-8b)

KV sizes, with slightly higher parameter overheads due to routing weights. To compare the real inference time of *mixSGA* and GQA, we measure decoding throughput (tokens per second) under a 50% KV budget, using a batch size of 1 and generating 10,000 tokens over five trials. As shown in Table 10, *mixSGA* achieves throughput performance of 57.28, 38.65, and 25.92 tokens/s on Llama3.2-1b, Llama3.2-3b, and Llama3.1-8b, respectively, compared to GQA's 59.75, 40.16, and 26.77 tokens/s. This results in a modest 3-4% overhead for *mixSGA*, reflecting its dynamic routing complexity. These results, based on a naïve implementation, suggest significant potential for optimization to further reduce this gap.

C.2 Optimal Expert Ratio Analysis

To understand why the 3:1:6 expert ratio consistently yields superior performance across models, we analyze the allocation of expert ratios under a fixed 50% KV budget. The ratios (x, y, z) for three experts are constrained as follows:

$$\begin{cases} x + 0.5y + 0.25z = 0.5, & \text{Budget,} \\ x + y + z = 1, & \text{Allocation,} \\ 0 \le x, y, z \le 1, & \text{Bound.} \end{cases} \tag{11}$$

We perform a grid search on Gemma2-2B, varying x in 0.05 increments, resulting in six feasible configurations as shown in Table 11. Notably, $x = 0.3, y = 0.1, z = 0.6$, (*i.e.*, x:y:z = 3:1:6) achieves the highest average ROUGE-L score (21.20) across Dolly (DL), Self-Instruct (SI), Vicuna (VC), and Super-Natural Instructions (SN), outperforming other configurations. This indicates that allocating 30% of tokens to the first expert (group size 1), 10% to the second (group size 2), and 60% to the third (group size 3) optimizes performance by prioritizing significant tokens for the first expert while efficiently handling less critical tokens with the third. These results justify the adoption of the default ratio for our experiments in this study.

Ratios	**DL**	**SI**	**VC**	**SN**	**Avg.**
1:17:2	20.61	10.18	6.42	14.54	12.94
1:7:2	23.49	10.70	14.10	17.72	16.50
3:11:6	25.36	12.60	**16.90**	18.52	18.35
1:2:2	25.80	13.45	15.54	19.04	18.46
1:1:2	24.79	16.08	12.70	26.01	19.90
3:1:6	**26.15**	**17.36**	14.47	**26.82**	**21.20**

Table 11: ROUGE-L scores (↑) for Gemma2-2B with varying expert ratios under a 50% KV budget across Dolly (DL), Self-Instruct (SI), Vicuna (VC), and Super-Natural Instructions (SN).