DHA: Learning Decoupled-Head Attention from Transformer Checkpoints via Adaptive Heads Fusion

Yilong Chen[1,2]*, **Linhao Zhang**[3]*, **Junyuan Shang**[3]‡, **Zhenyu Zhang**[3], **Tingwen Liu**[1,2]†, **Shuohuan Wang**[3], **Yu Sun**[3]
[1] Institute of Information Engineering, Chinese Academy of Sciences
[2] School of Cyber Security, University of Chinese Academy of Sciences
[3] Baidu Inc.
{chenyilong, liutingwen}@iie.ac.cn
{zhanglinhao, shangjunyuan, zhangzhenyu07, wangshuohuan, sunyu02}@baidu.com

Abstract

Large language models (LLMs) with billions of parameters demonstrate impressive performance. However, the widely used Multi-Head Attention (MHA) in LLMs incurs substantial computational and memory costs during inference. While some efforts have optimized attention mechanisms by pruning heads or sharing parameters among heads, these methods often lead to performance degradation or necessitate substantial continued pre-training costs to restore performance. Based on the analysis of attention redundancy, we design a Decoupled-Head Attention (DHA) mechanism. DHA adaptively configures group sharing for key heads and value heads across various layers, achieving a better balance between performance and efficiency. Inspired by the observation of clustering similar heads, we propose to progressively transform the MHA checkpoint into the DHA model through linear fusion of similar head parameters step by step, retaining the parametric knowledge of the MHA checkpoint. We construct DHA models by transforming various scales of MHA checkpoints given target head budgets. Our experiments show that DHA remarkably requires a mere 0.25% of the original model's pre-training budgets to achieve 97.6% of performance while saving 75% of KV cache. Compared to Group-Query Attention (GQA), DHA achieves a 5× training acceleration, a maximum of 13.93% performance improvement under 0.01% pre-training budget, and 4% relative improvement under 0.05% pre-training budget.

1 Introduction

Transformer-based large language models (LLMs) shine in various natural language tasks due to their powerful understanding and generation capabilities [1, 2, 3]. Multi-Head Attention (MHA) is widely used in LLMs, with the number of heads increasing as the model size grows. However, MHA inference overhead increases linearly with the expansion of the context and model sizes, due to the surprisingly large memory consumption of the *KV Cache* mechanism. For instance, a 7 billion-parameter model with 32 heads and 32 layers, an input batch size of 4, and a sequence length of 32k results in 64GB of KV cache, which is $4.7\times$ larger than the model weights.

To reduce computational and memory overhead during inference, a widely used approach involves adapting the MHA model to a more efficient structure through the reuse of parameters across multiple heads [4, 5, 6] , such as Multi-Query Attention (MQA) [4] and Grouped-Query Attention (GQA) [5]. These methods utilize a portion of the original training computation which avoid information loss

* denotes equal contribution. † denotes corresponding author. ‡ denotes project lead.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

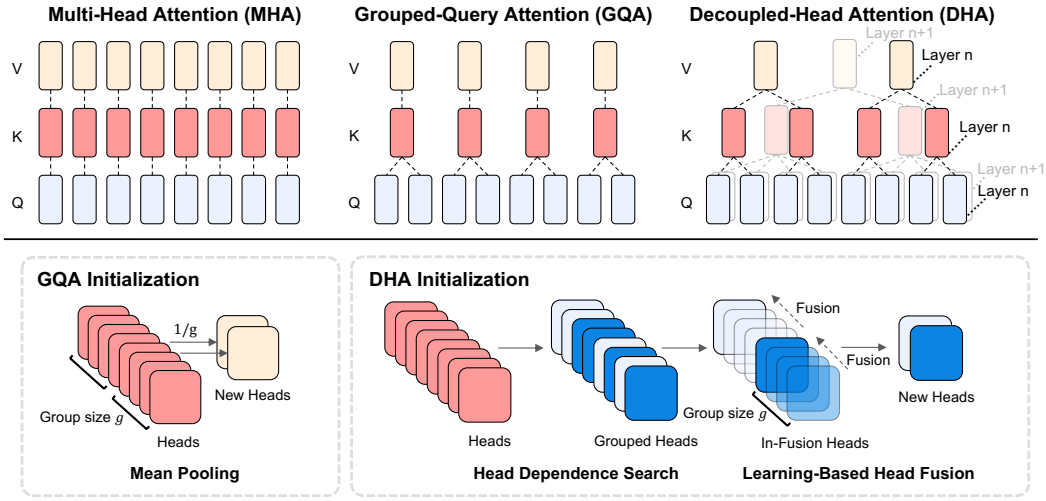


Figure 1: **Upper:** Overview of Decoupled-head method. Multi-Head attention (MHA) has equal query, key and value heads. Grouped-Query attention (GQA) instead shares single key and value heads for each group of query heads. Decoupled-Head attention (DHA) shares key heads and value heads for different groups of query heads in different layers. **Lower:** GQA Initialization: Heads are mean pooled into a single head; DHA Initialization: DHA search head grouping and progressively fuse heads to maintain parameter functions.

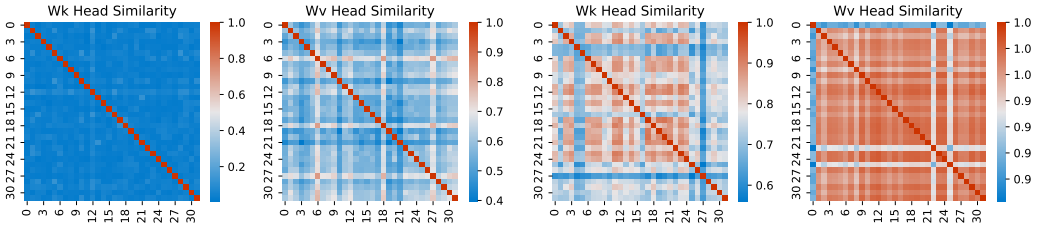
due to training-inference inconsistencies, a common issue in pruning-based [7, 8, 9, 10, 11] works. However, the training computation is prohibitively expensive for recovering the model's performance, due to the information loss in the parameters when creating the initial point.

Thus, in this work, we seek to address the following question:

*How can we construct a **more efficient** model while keeping **costs as low as possible**?*

With the limited understanding of parameter characteristics in modern LLMs, we first perform an empirical analysis from the perspectives of heads' parameter similarity. We observe that there are some head-clusters with high internal similarity in MHA checkpoints. Similar head clusters imply a enormous redundancy in MHA, which coincides with the sparsity found in previous studies [12, 13]. In particular, the clusters of key heads and value heads across different layers show a **decoupled** distribution, meaning that there is a significant variation in the distribution of head-cluster similarities across layers, key heads and value heads, as illustrated in Fig. 2a,2b. Intuitively, we can prune redundant heads based on the above characteristics. Nonetheless, each head has its unique role, and thus no heads should be arbitrarily discarded. Furthermore, we find that linear fusion based on multiple similar heads can reconstruct the original head functionality without causing a significant performance drop (see Sec. 3.1). Based on this observation, we believe that selectively fusing corresponding heads in clusters can construct a more efficient architecture with minimum loss.

In this paper, we propose **Decoupled-Head Attention (DHA)**, an efficient attention architecture developed through the **Adaptive Head Fusion** of checkpoints' parameters. Recalling the decoupled heads parameter characteristics, DHA allocates different numbers of key heads and value heads at different layers to balance model efficiency and performance. The MHA checkpoint can be rapidly transformed into DHA with three stages: **Search**, **Fusion**, and **Continued Pre-training (CT)**. During the Search stage, we group similar functional heads together and determine reasonable allocations of key heads and value heads for each layer. Specifically, we reconfigure the original key and value head into multiple linear combinations of heads within the same layer. Thus, we can allocate the heads based on the loss after replacement. In the Fusion stage, we perform linear fusion on similar heads, ensuring the preservation of original functionality. Leveraging the Augmented Lagrangian approach [14, 15], the Fusion operator initializes from MHA and explores possible head combinations in the early training, followed by refined intra-group head fusion in the later. Based on well-trained operators on unlabeled data, we can rapidly obtain high-performing initial points for DHA from MHA checkpoints, requiring only a minimal amount of Continued Pre-training to restore performance.



(a) Head Parameter Similarity in 0th Layer (b) Head Parameter Similarity in 21st Layer

Figure 2: Visualization of the similarity between heads within the MHA of LLaMA2-7B model at the 0th layer (a) and the 21st layer (b). Details in Appendix E.1. Key heads and value heads exhibit decoupled distributions.

To verify the effectiveness, we construct DHA on models of different sizes, such as LLaMA2-7B [3], Sheared-LLaMA-2.7B & -1.3B [16] with the heads budget ratio set at 50% and 25%. With a modest fusion training of just 0.2 billion tokens, DHA learns sufficiently competent initial points. As the continued pretraining progresses, DHA continuously outperforms GQA narrowing the gap with MHA on 9 representative downstream tasks. DHA only requires 0.25% of MHA pre-training budget. Meanwhile, DHA is capable of reducing *KV Cache* by up to 75% compared to MHA with minimal accuracy trade-off (maximum of 5.6%). Compared to GQA, DHA achieves a 5× training acceleration, a maximum 13.93% performance improvement under 0.01% pre-training budget, and 4% relative improvement under 0.05% pre-training budget. Overall, DHA exhibits great performance and efficiency, which can be quickly adapted to various existing MHA Transformer models.

2 Background

Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_p) \in \mathbb{R}^{p \times d_{\text{model}}}$ denote the input prompts of hidden states of a Transformer layer, where p stands for the number of tokens and d_{model} for the hidden state dimension.

Multi-Head Attention (MHA) MHA [17] performs the attention with H different heads. For h-th head, different weight matrices $\mathrm{W}_q^h, \mathrm{W}_k^h, \mathrm{W}_v^h \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are used to project the input sequence into query, key, value vector, where d_k represents head dim. Denote softmax funcion as σ, we have:

$$\text{MHA} = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_{\text{H}}\right) \mathrm{W}_O, \text{where head}_h = \sigma\left(\mathbf{X}\mathrm{W}_q^h (\mathbf{X}\mathrm{W}_k^h)^T \cdot \frac{1}{\sqrt{d_k}}\right)\mathbf{X}\mathrm{W}_v^h \tag{1}$$

Ultimately MHA combines heads' outputs through the output projection $\mathrm{W}_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$[2].

Grouped-Query Attention (GQA) & Multi-Query Attention (MQA) To accelerate inference, MQA [4] and GQA [5] have been proposed based on the idea of reusing head parameter weights. In these variants, H different query heads are divided into G groups, where the heads within the same group share the same key heads and value heads parameter matrices. Given the mapping relationship from the h-th query head to a GQA key and value heads using the many-to-one function $g(h)$, we define the h-th head forward pass as:

$$\text{head}_h = \sigma\left(\mathbf{X}\mathrm{W}_q^h (\mathbf{X}\mathrm{W}_k^{g(h)})^T \frac{1}{\sqrt{d_k}}\right)\mathbf{X}\mathrm{W}_v^{g(h)}, \text{where } \mathrm{W}_{k/v}^{g(h)_{\text{init}}} = \frac{\sum_{\mathrm{W}_{k/v} \in \mathbb{K}/\mathbb{V}_{g(h)_{\text{init}}}} \mathrm{W}_{k/v}}{|\mathbb{K}/\mathbb{V}_{g(h)_{\text{init}}}|} \tag{2}$$

Here, $\mathbb{K}/\mathbb{V}_{g(h)_{\text{init}}}$ refers to MHA key/value heads parameters within the $g(h)_{\text{init}}$-th group during GQA initialization. When transitioning from an MHA checkpoint, GQA uses the mean pooling method for heads within the group. MQA is a special case of GQA where $G = 1$[3].

Due to mean pooling for initialization, GQA results in loss of capability when converting from MHA, necessitating expensive pre-training to recover. We aim to identify better initialization and more refined head mapping relationships to achieve superior performance with reduced training costs.

[2]MHA consists of H heads and stores a $2 \times H \times p \times d_{\text{model}}$ dimension KV cache for accelerating inference.
[3]GQA and MQA consist of $H + 2 \times G$ heads in total and store a $2 \times G \times p \times d_{\text{model}}$ dimension KV cache.

3 Observation

To study the inherent characteristics of head parameters in MHA, we use Centered Kernel Alignment [18] to calculate the heads' similarity within each layer's W_k, W_v. Based on the average heads' similarity, we define the redundancy of each MHA layer. For details, please refer to Appendix B.1.

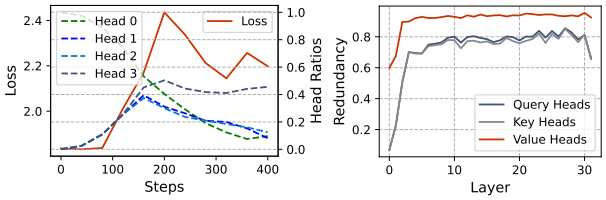
3.1 Head clusters in MHA

Observation From Fig. 2a and Fig. 2b, we observe that clusters form spontaneously among heads, with high similarity within clusters and low similarity between clusters. It indicates that heads among different clusters may have distinct functionalities, processing linguistic features in various aspects.

Analysis Given the numerous similar head clusters in W_k and W_v, we identified the opportunity to linearly fuse functionally similar heads within clusters while retaining each head's parameterized knowledge. We conducted an empirical study, transforming the parameters of Head 0 in MHA into a linear fusion of the parameters from Heads 0, 1, 2, and 3. We share the fusion head across four query heads and progressively optimize the fusion ratio under the LmLoss. For details, please refer to Sec. 4.2. As shown in Fig. 3a, the loss remains unchanged as the proportion of Head 0 decreases and only increases when four heads parameters' ratios approach an even distribution. It suggests that fusing similar parameters can reduce the number of heads without significant information loss.

3.2 Variability across Layers and KV pairs

Observation The distribution of similar head clusters varies between different layers. As illustrated in Fig.2a, 2b, the 0th layer of MHA shows few similar head clusters, while the 21st layer exhibits many. Within the same layer, value heads exhibit more clusters and higher similarity compared to key heads, indicating a divergence between the two. Fig. 3b shows that the redundancy is lower in the initial and final layers, and higher in the middle layers. Moreover, W_v redundancy significantly exceeding that of W_k.



(a) Loss in 4-Head Fusion (b) 32-Layer Redundancy

Figure 3: (a) Model loss with heads proportions in linear fusion. (b) Layer Redundancy of the query, key, value head parameter matrices in the LLaMA2-7B model MHA.

Analysis Inspired by layer and key-value head variability, we propose allocating more heads to layers with lower redundancy to enhance learning and expression. Since W_v shows higher redundancy than W_k, we can decouple and allocate more heads budget to critical key components, while compressing redundant value heads at a higher compression rate. Finer grouping and sharing based on the parameters function may contribute to compression rates and performance improvements.

4 Method

In this section, we propose a more efficient Decoupled Head Attention (DHA) architecture and its construction process. We define DHA in Sec. 4.1 and Adaptive Head Fusion algorithm in Sec. 4.2. Then we demonstrate the adaptive construction based on the MHA checkpoint, which can be divided into: **Search**, **Fusion**, and **Continued Pre-training** (Discussed in in Sec. 4.3). Finally, we introduce practical application of our DHA architecture on the LLaMA2 model in Sec. 4.3.

4.1 Decoupled-Head Attention (DHA)

We present a more efficient attention architecture called Decoupled-Head Attention (DHA). Based on observed significant functional differences among different layers' key value heads, DHA adaptively allocates more heads to critical components, thus enhancing overall model efficiency and performance.

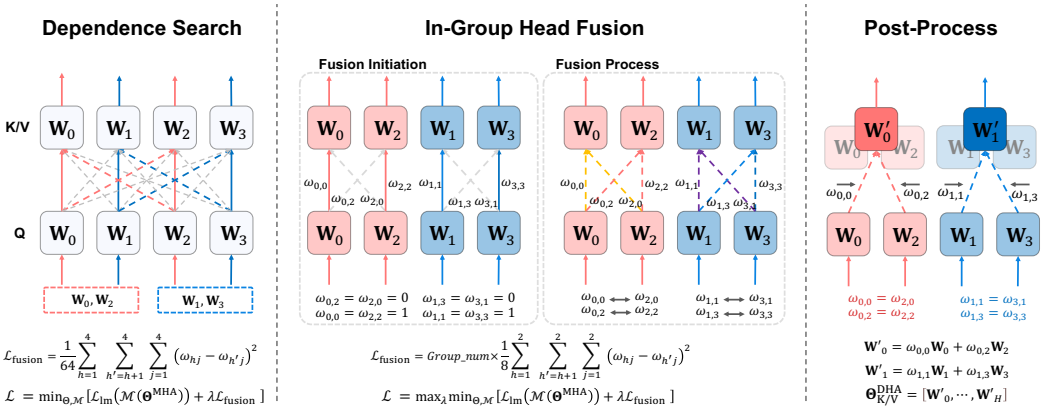


Figure 4: An illustration of DHA. First, we reconstruct the a single head forward as a linear combination of multiple heads' forward with proportions ω, grouping heads with similar functions based on multi-step optimization. Next, we initialize and optimize the fusion operators. $\Leftrightarrow$ indicates the optimization narrows the distance between proportions ω. Finally, we fuse heads within groups and continued pre-training DHA model.

Definition Defined model with L layers and H^{Q} heads in a layer, the numbers of Key heads and Value heads in the l-th layer are denoted as $H_l^{\mathrm{K}}, H_l^{\mathrm{V}}$. We define the **many-to-one** mapping functions $d^{\mathrm{K}}(h,l)$ and $d^{\mathrm{V}}(h,l)$ representing key and value head corresponding to the h-th query head in l-th DHA layer. The computation be formalized as follows:

$$\mathrm{head}_{h,l} = \sigma\left(\mathbf{X}\mathrm{W}_q^h(\mathbf{X}\mathrm{W}_k^{d^{\mathrm{K}}(h,l)})^T \cdot \frac{1}{\sqrt{d_k}}\right)\mathbf{X}\mathrm{W}_v^{d^{\mathrm{V}}(h,l)} \tag{3}$$

DHA shares a key and value head in multi query heads' computation based on independent mapping functions at different layers[4]. GQA can be considered a special case of DHA, where not only all layers share the same mapping functions, but the mapping functions for keys and values are identical.

4.2 Learning Efficient MHA Transformation via Linear Heads Fusion

Due to the high cost of building an efficient Attention mechanism in LLM from scratch, we construct DHA based on the existing MHA checkpoint using minimal computational budgets. Based on the head clustering phenomenon in MHA, we propose a linear fusion method for similar heads within clusters. By incrementally fusing head parameters, we compress the number of heads while retaining the original model's knowledge, significantly reducing training budgets and improving performance.

Goal Formally, we define a model with Layer number L and Head number H as $\mathbf{\Theta}_{L,H} = [\mathrm{W}_1, \cdots, \mathrm{W}_L]$, where $\mathrm{W}_l \in \mathbb{R}^{D\times D}$ denotes the weight of layer l with input and output dimension D. In the initialization, our goal is to transfer knowledge from a MHA model $\mathbf{\Theta}^{\mathrm{MHA}}_{L,H_1}$ to a DHA model $\mathbf{\Theta}^{\mathrm{DHA}}_{L,H_2}$, where $H_1 > H_2$. By learning a fusion operation that minimizes the functional difference between MHA and DHA model, the goal can be formalized as

$$\arg\min_{\mathbf{\Theta},\mathcal{M}} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\mathcal{L}_{\mathrm{lm}}\left(\boldsymbol{x};\mathcal{M}(\mathbf{\Theta}^{\mathrm{MHA}})\right) + \lambda\mathcal{L}_{\mathrm{fusion}}\left(\boldsymbol{x};\mathcal{M}(\mathbf{\Theta}^{\mathrm{MHA}}),\mathbf{\Theta}^{\mathrm{DHA}}\right)\right] \tag{4}$$

Where $\mathcal{M}$ is the fusion operator, $\mathcal{D}$ is the training dataset, $\mathcal{L}_{\mathrm{lm}}$ is the training loss function, $\mathcal{L}_{\mathrm{fusion}}$ measures the transformation from MHA to DHA, and λ is the learnable scale factor.

Fusion Operator During DHA initialization, the fusion operator $\mathcal{M}$ constructs new heads based on the linear combinations of the original key and value heads within the group, and shares the new heads among the query heads' forward. Define each group $\mathbb{K}_{d^{\mathrm{K}}(h,l)}, \mathbb{V}_{d^{\mathrm{V}}(h,l)}$ represents key, value heads group corresponding to the h-th query head in l-th layer, $g = \left\{g^{\mathrm{K}}, g^{\mathrm{V}}\right\}$ as the group size. By introducing variables $\omega_h = \{\omega_{hj}\}_{j=1}^{g}, \omega \in \mathcal{M}$ represents the proportion of j-th key, value head

[4]DHA consists of $H = H^{\mathrm{Q}} + \sum_{l=1}^{L} H_l^{\mathrm{K}} + \sum_{l=1}^{L} H_l^{\mathrm{V}}$ heads in total.

involved in the h-th query head forward within group. For each group, a head have forward pass as:

$$\text{head}_{h,l} = \sigma\left(\mathbf{X}\mathrm{W}_q^h(\mathbf{X}\mathrm{W}_k^{d^{\mathrm{K}}(h,l)})^T \cdot \frac{1}{\sqrt{d_k}}\right)\mathbf{X}\mathrm{W}_v^{d^{\mathrm{V}}(h,l)}, \text{where } \mathrm{W}_{k/v}^{d^{\mathrm{K/V}}(h,l)} = \sum_{j=1}^{g^{\mathrm{K/V}}} \omega_{hj}\mathrm{W}_{k/v}^j \tag{5}$$

where ω_{hj} will be initialized to Kronecker delta function, which equals 1 if and only if $h = j$, and equals 0 otherwise. Under this initialization setting, the forward computation of DHA is completely equivalent to that of MHA, see Fig. 4.

Optimization During the optimization phase, we design a fusion loss to optimize the initialized model towards DHA target architecture. Note that after initialization, the mapping of heads within the group $\mathrm{W}_q^{h,l} \to \mathrm{W}_{k/v}^j$ is a **many-to-many** mapping, denoted by the function $d_{\text{init}}^{\mathrm{K/V}}(h,l)$. This indicates that in the forward process of each query, the key head or value head can be expressed as different linear combinations of g MHA heads. According to Eq. 3, we aim to achieve a **many-to-one** mapping that a single fused key head or value head are shared across multiple query heads in DHA, denoted by the function $d^{\mathrm{K/V}}(h,l)$. Thus, we design a fusion loss $\mathcal{L}_{\text{fusion}}$ to optimize the initial mapping functions to converge to a single mapping function, i.e., $d_{\text{init}}^{\mathrm{K/V}}(h,l) \to d^{\mathrm{K/V}}(h,l), \forall h \in \mathbb{K}_n/\mathbb{V}_n$. Specifically, we define the optimization objective as minimizing the difference between the mapping functions of different query heads h and h' within the l-th layer and n-th group:

$$\mathcal{L}_{\text{head}_l^n}(h,h') = \frac{1}{g}\left\|\sum_{j=1}^{g}\omega_{hj}\mathrm{W}_{k/v}^j - \sum_{j=1}^{g}\omega_{h'j}\mathrm{W}_{k/v}^j\right\|^2 = \frac{1}{g}\left(\sum_{j=1}^{g}(\omega_{hj}-\omega_{h'j})\mathrm{W}_{k/v,ij}^j\right)^2 \tag{6}$$

where $g = g^{\mathrm{K/V}}$ represents the number of heads within a group. Since $W_{k/v,ij}^{j,\mathrm{MHA}}$ can be regarded as an orthogonal scalar, and thus we only need to optimize fusion variables ω , so we have:

$$\mathcal{L}_{\text{fusion}} = \sum_{l=1}^{L}\sum_{n=1}^{N}\sum_{h=1}^{g}\sum_{h'=h+1}^{g}\mathcal{L}_{\text{head}_l^n}(h,h'), \text{subject to } \mathcal{L}_{\text{head}_l^n}(h,h') = \frac{1}{g}\sum_{h=1}^{g}\sum_{j=1}^{g}(\omega_{hj}-\omega_{h'j})^2 \tag{7}$$

Where N represents the number of groups, $N = \frac{H_1}{g}$. The fusion loss can be measured as the mean squared error loss of the head and head fusion variables within each group at each layer.

Augmented Lagrangian approach When the fusion loss is zero, the key and value heads corresponding to query heads within the group are optimized to share the same fusion variables. This allows the new DHA key-value head parameters to be effectively shared among the queries in the group. Given that it is challenging to optimize the loss to a very small value, we use an augmented Lagrangian approach [14, 15] for incremental architectural transformations. Define t as the target loss, b as the base decay factor, s as the current global step, k as the total number of steps in the warm-up phase, the overall training optimization is an adversarial game:

$$\max_{\lambda}\min_{\boldsymbol{\Theta},\mathcal{M}}\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\mathcal{L}_{\text{lm}}\left(\boldsymbol{x};\mathcal{M}(\boldsymbol{\Theta}^{\text{MHA}})\right) + \lambda\max\left(\mathcal{L}_{\text{fusion}} - t, 0\right)\right], \text{where } t = \max\left(0, b^s\left(1-\frac{s}{k}\right)\right) \tag{8}$$

Our Augmented Lagrangian approach enforces the constraint $\mathcal{L}_{\text{fusion}} \leq t$, where the Lagrange multiplier λ is updated during training. The update increases the loss unless the constraint is satisfied. Early in training, the model tolerates more significant discrepancies between head weights, promoting exploration. As training progresses, the margin shrinks, enforcing stricter adherence to minimizing discrepancies and refining head alignment within the group.

4.3 Adaptive DHA Transformation on LLaMA Model

Based on the observation of similar head clusters and key-value head parameter variability across layers, DHA employs the adaptive transformation. It allows DHA to search for and fuse similar heads while allocating different group sizes across layers. As shown in Fig. 4, the transformation can be divided into three stages: **Search**, **Fusion** and **Continued Pre-training**.

In the beginning, we initialize the DHA operators to the MHA model. Next, we perform 240 **Search** steps, calculating $\mathcal{L}_{\text{fusion}}$ for each layer and $\mathcal{L}_{\text{head}}$ for all heads. Based on the $\mathcal{L}_{\text{head}}$, we perform head grouping intending to minimize the average loss of heads within each group and maximize the average loss of heads between groups and groups. Based on $\mathcal{L}_{\text{fusion}}$, we use a dynamic programming

algorithm to allocate more head budget to layers with higher loss within a total budget. It allows us to fuse the most similar heads to minimize loss during the fusion process and selectively compress the model's most redundant components. For more details, see Apendix B.3, B.4.

During **Fusion** phase, we modified the forward propagation path of MHA in the form of DHA based on the layer head budget and head grouping obtained during the **Search** phase. Then we antagonistically optimize the fusion operator and update Lagrangian multipliers λ, the $\mathcal{L}_{\text{fusion}}$ that marks this DHA fusion process decreases. When $\mathcal{L}_{\text{fusion}}$ is less than 1e-3, we terminate the fusion algorithm and enter the **Continued Pre-training** phase.

During the **Continued Pre-training** phase, we fuse MHA head parameters based on averaged fusion weights to construct DHA initialization. DHA initialization can recover the performance with a small amount of restorative pre-training. For more information, please refer to Appendix B.2.

Our method can theoretically transform MHA architecture in any transformer model to efficient DHA architecture. Using LLaMA models as case studies, we implemented DHA transformation with various compression rates on all MHA layers. Notably, we expanded the dimension of each head's fusion coefficient ω from 1 to the head's dimension d_k, allowing for finer-grained parameter fusion and better knowledge retention. Intuitively, we learn different fusion ratios for each dimension of the head. Only a very small number of additional parameters need to be introduced, DHA significantly accelerates training and improves performance.

5 Empirical Evaluation

5.1 Experimental Setup

Data. To train DHA operators and extend pre-training, we employ the RedPajama [19], which parallels the LLaMA training data across seven domains: CommonCrawl, C4, GitHub, Wikipedia, Books, ArXiv, and Stack-Exchange. This dataset comprises a validation set with 2 million tokens, a training set containing 4 billion tokens and an additional pre-training set totaling 50 billion tokens.

Training. Our experimental framework utilizes the Sheared-LLaMA codebase [16] implemented on the Composer package [20], and is executed on 8 NVIDIA A100 GPUs (80GB). The models are trained with a sequence length of 4096, employing a global batch size of 64 during the fusion phase and 256 during the continued pre-training phases. The learning rates were set at 1e-4 for language modeling loss, and 1e-2 for Lagrangian multipliers and fusion operators respectively.

Budget. DHA models were trained for 1000 steps (0.2B token budget) during the fusion phases. For the continued pre-training, we trained both baseline models and DHA for up to 50000 steps (50B token budget). To evaluate the training acceleration capability of DHA, we evaluate its performance under two budget scenarios. First, we set a budget of **1B tokens** to compare the early-stage rapid convergence capabilities of DHA and GQA. Then, we set a budget of **50B tokens** to further assess the performance of DHA over a more extended training period.

Evaluation. We employed the lm-evaluation-harness [21] to evaluate our models. For common sense and reading comprehension tasks, we report 0-shot accuracy results for SciQ [22], PIQA [23], WinoGrande (Wino.) [24], ARC Easy(ARC-E.) [25], and HellaSwag (HellaS.) [26], alongside 25-shot accuracy for ARC Challenge (ARC-C.) [27]. In the assessments of continued QA and text understanding, we report 0-shot accuracy for LogiQA [28], 32-shot BoolQ [29], and 0-shot LAMBADA [30]. All reported results were calculated with the mean and stderr of multiple experiments.

Instruction tuning evaluation. To assess our models' capabilities after instruct tuning [31, 32], we fine-tune both DHA and baseline models on 10,000 instruction-response pairs from the ShareGPT dataset [5] and evaluate on another 1,000 instructions, using GPT-4 for response evaluator [33]. The win rate of our model relative to the baseline is reported. For detailed information, refer to Appendix C.1.

[5] `https://sharegpt.com`

Table 1: Comprehensive assessment of model's fundamental capabilities, in which DHA models demonstrate competitive performance while requiring significantly fewer training resources. Models with † use MHA.

		Commonsense & Comprehension						Continued		LM	
Model	**Budget**	**SciQ**	**PIQA**	**Wino.**	**ARC-E**	**ARC-C**	**HellaS.**	**LogiQA**	**BoolQ**	**LAMB.**	**Average**
LLaMA2-7B†	2T	94.1	78.1	69.1	76.3	49.7	58.9	25.7	80.8	74.1	67.4
DHA-7B-50%	50B	93.4	78.5	69.1	73.8	45.9	58.6	22.5	79.1	71.1	65.8
DHA-7B-25%	50B	92.4	78.5	68.6	72.9	43.9	57.6	22.4	76.7	70.2	64.8
GQA-7B-50%	1B	90.7	**76.8**	66.5	**71.3**	41.9	53.6	**22.4**	70.5	67.0	62.3
DHA-7B-50%	1B	**90.8**	76.5	**66.7**	**71.3**	**44.6**	**55.1**	**22.4**	**74.8**	**67.2**	**63.3**
GQA-7B-25%	1B	86.5	74.3	59.1	67.6	37.5	49.2	**24.1**	65.8	58.3	58.0
DHA-7B-25%	1B	**90.0**	**75.2**	**63.8**	**70.4**	**39.3**	**52.2**	21.1	**72.3**	**62.9**	**60.7**
S.-LLaMA-2.7B†	2T	91.2	76.1	64.9	67.3	38.8	52.2	22.1	74.4	68.3	61.7
GQA-2.7B-50%	1B	86.7	74.8	59.0	64.0	34.2	48.2	**23.8**	64.9	60.3	57.3
DHA-2.7B-50%	1B	**86.8**	**75.1**	**59.5**	**64.6**	**35.1**	**48.7**	22.4	**66.4**	**61.7**	**57.8**
GQA-2.7B-25%	1B	82.0	72.8	54.9	58.4	31.0	42.9	**21.7**	58.5	49.6	52.4
DHA-2.7B-25%	1B	**85.6**	**74.1**	**57.6**	**61.5**	**32.4**	**45.9**	**21.7**	**63.1**	**56.9**	**55.4**
S.-LLaMA-1.3B†	2T	87.0	73.6	58.2	60.9	29.5	45.4	21.8	65.5	61.3	55.9
GQA-1.3B-50%	1B	84.3	**72.3**	**55.8**	57.5	28.2	41.8	20.7	62.9	52.9	52.9
DHA-1.3B-50%	1B	**84.5**	72.0	55.2	**58.1**	**28.7**	**42.6**	**21.5**	**63.7**	**55.4**	**53.6**
GQA-1.3B-25%	1B	76.6	70.0	52.9	51.9	23.5	37.6	21.0	**59.9**	41.0	48.3
DHA-1.3B-25%	1B	**82.8**	**71.1**	**54.0**	**55.4**	**25.8**	**40.5**	**21.5**	57.6	**48.6**	**50.8**

Table 2: Ablation Results of DHA *w.o.* Linear Heads Fusion and Adaptvie Transformation. Experiments are conducted with LLaMA2-7B with 25% heads budget and 0.5B & 1B training budget on 0-shot Evaluation.

Models	**SciQ**	**PiQA**	**Wino.**	**ARC-E.**	**ARC-C.**	**LogiQA**	**LAMB.**	**Average**	**Diff**
DHA-7B-25% (0.5B)	88.6	75.9	61.3	68.2	36.1	23.8	63.2	59.6	–
w.o. Linear Heads Fusion	83.4	73.7	57.3	63.6	29.4	22.0	51.9	54.5	−5.1
w.o. Adaptvie Transformation	87.9	74.1	60.1	69.4	34.7	19.5	62.1	58.3	−1.3
DHA-7B-25% (1B)	90.0	75.2	63.8	70.4	37.5	21.1	62.9	60.1	–
w.o. Linear Heads Fusion	87.5	74.5	60.7	67.3	32.8	21.7	58.3	57.5	−2.6
w.o. Adaptvie Transformation	89.5	74.6	62.8	69.1	36.3	21.6	62.4	59.5	−0.6
DHA-7B-25% (5B)	91.7	76.8	64.4	70.9	42.8	21.8	68.4	62.4	–
GQA-7B-25% (5B)	91.5	76.6	63.9	70.5	42.3	22.1	67.8	62.1	−0.3

Baselines. We selected the LLaMA2-7B model and Sheared-LLaMA-2.7B&1.3B (S.-LLaMA-2.7B&1.3B) as the MHA baselines. For each scaled model's checkpoint, we constructed 25% and 50% compressed GQA and DHA models in 0.5B & 1B tokens (0.01% & 0.05% of pretrain budget [6]).

5.2 Experimental Results

Foundational Capabilities. Tab. 1 shows the foundational capabilities of DHA and GQA models at 50% and 25% compression rates (e.g., 64 key value heads compress to 16) across different scales. DHA was obtained by transforming LLaMA using adaptive head fusion and then further pre-trained with 1B tokens. For comparison, we constructed GQA with the same compression rates and training budget. Experiments show that DHA can achieve efficient architecture with only 0.05% of the original model's pre-training cost without significant performance loss. Compared to GQA, DHA consistently achieved better performance across all model scales and pre-training cost settings. Under the same checkpoint and training budget settings, DHA demonstrates significant improvements at higher compression rates. For example, with LLaMA7B at a 25% compression rate, DHA achieved a 4% relative performance improvement over GQA. This showcases DHA's fusion algorithm's ability to efficiently retain knowledge at high compression rates and the advantage of DHA's decoupled architecture in adaptively compressing redundant components. Possibly due to the lack of relevant data, DHA performed on par with LogiQA. As shown in Fig. 6, DHA's performance advantage

[6]LLaMA2 was pre-trained on 2T data; Sheared-LLaMA pruned LLaMA on 50B RedPajama data.

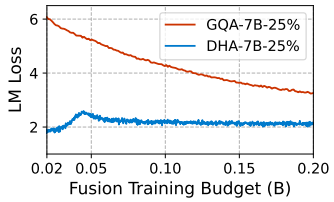


Figure 5: LM Loss with Fusion Training (B) between GQA-7B-25% and DHA-7B-25%.

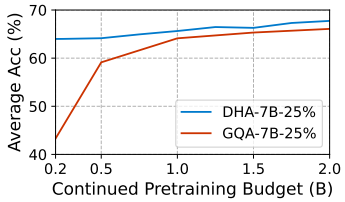


Figure 6: Task Average Accuracy (%) with CT (B) of DHA-7B-25% and GQA-7B-25%.

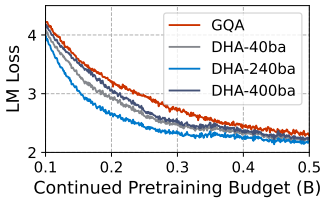


Figure 7: LM Loss with CT (B) between GQA-7B-25% and Cold-Start DHA after X Step Search.

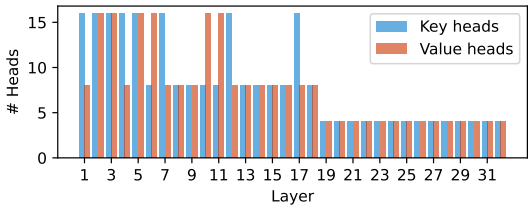


Figure 8: Allocation of key-value head budgets with 32 layers in DHA-7B-25% after 240 step Search.

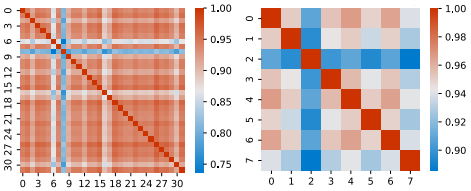


Figure 9: Similarity of value heads in the 7th layer of LLaMA2-7B MHA **(Left)** or the transformed DHA-7B-25% DHA **(Right)**.

becomes more remarkable with reduced training budgets. It indicates that DHA effectively retains knowledge of larger models, significantly reducing pre-training costs.

Better Initialization. We examined whether DHA offers a better initialization point than GQA by pre-training both DHA and GQA models on the original RedPajama dataset. Fig. 5 shows that the initial loss of the GQA model is high and decreases slowly. In contrast, the DHA model starting from MHA exhibits a minor increase in LM loss as fusion progresses, maintaining a consistently lower loss. DHA converges with just 0.1B data, demonstrating a 5× training speedup compared to GQA. Fig. 6 reports the average downstream task accuracy of DHA and GQA during continued pre-training. DHA achieves comparable performance to GQA's at 1B tokens with only 0.2B tokens, outperforming GQA's 0.2B token performance by 13.93%. This demonstrates DHA's effectiveness in retaining parameter information. Ultimately, DHA achieves a higher performance ceiling than GQA due to retaining information from the original model and its more efficient architecture, whereas GQA loses information during initialization.

5.3 Analysis

Ablation Study. We report the effects of ablating Linear Heads Fusion and Adaptive Transformation in Tab. 2. When training with less data (0.5B), ablating Linear Heads Fusion leads to significant performance degradation, indicating that this method preserves crucial knowledge in LLMs, greatly accelerates DHA model training, and enhances performance. Adaptive Transformation allocates parameters more efficiently during construction, thereby strengthening the model's capability and reducing training difficulty. When we allocate more training budget to 1B, DHA's efficient architecture after Adaptive Transformation plays a more significant role, enhancing the model's performance ceiling. When continuing to pre-train the DHA model, it demonstrated strong learning capabilities and sustained performance improvements, ultimately achieving 97.6% of the performance with just 0.25% of the original model's pre-training budget, while saving 75% of the KV cache.

Training Budget Allocation. Allocating more computation to the fusion phase aids in better retention of information within the checkpoint. Our experiments assessed the effects of budget allocations between the fusion and CT phases within **a fixed budget of 2 billion tokens**. Tab. 3 shows that increasing the fusion budget consistently from 0.05B to 0.2B improves model performance at the initialization point. Training with just 0.1B data is sufficient to achieve a good starting point, and increasing fusion budget will not affect the final performance. This experiment also

Table 3: Data budget allocation to fusion and continued pre-training(CT) and 0-shot Task Average Accuracy (%) in DHA-1.3B.

Fusion		**CT**	
Tokens	Avg.Acc	Tokens	Avg.Acc
0.05B	33.74	4.95B	59.08
0.10B	**38.32**	**4.90B**	**59.53**
0.15B	48.26	4.85B	59.46
0.20B	52.54	4.80B	59.16

demonstrates the necessity and effectiveness of the fusion stage under low-resource conditions. When we have a larger training budget, we can allocate more resources to the fusion stage to achieve a better initialization point for DHA.

Heads Budget Allocation. We investigated how the model adaptively allocates decoupled head group sizes across different layers under global head budgets. As illustrated in Fig. 8, the head numbers of DHA layers decrease from higher to lower across layers. Deeper layers exhibit higher compression rates due to greater redundancy. However, the initial and crucial layers need more heads, suggesting they may have specialized functions. As shown in Fig. 7, we presented the LM loss for the cold-start training of DHA models initialized with parameter averaging under different DHA configurations obtained at various search steps. Despite using the same initialization method as GQA, DHA exhibits a faster loss decline and a lower final loss. This indicates that DHA's architecture can accelerate training and achieve better performance, even without Linear Heads Fusion method.

Parameter Characteristics in DHA. For interpretability analysis, we visualized the parameter characteristics of the post-fusion DHA model in Fig. 9 (detials in Appendix E.2), and compared them with those prior to fusion. The DHA parameter distribution shows consistency with MHA's. This indicates that DHA effectively aggregates multiple similar functional heads within clusters and new fused heads successfully reconstruct the functionalities of multiple origin heads in MHA. It is noteworthy that the significant reduction in the number of similar heads within the DHA architecture indicates that our method effectively reduces redundancy among the heads.

6 Related Work

Advanced Multi-Head Attention. Some efforts have been converting the traditional Multi-Head Attention (MHA) [17] to Multi-Query Attention (MQA) [4], Group-Query Attention (GQA) [5] or GQKVA [6]. These methods achieve a balance between performance and efficiency by reducing the number of head parameters through parameter reuse across grouped heads. DHA is inspired by these methods and has a much higher optimization rate and much less training overhead.

Efficient Pre-training Approaches. In recent years, the ability of incremental training to accelerate large-scale model training by studying how to obtain the optimal initialization point for training has thus attracted much attention [34, 35]. Net2Net [36] uses function-holding transformations to expand the width by duplicating neurons, and uses a unitary layer implementation to expand the depth. LiGO [37] proposes a learnable expansion method that can be used at the initial initialization point of a transformer. DHA is inspired by these methods, but we investigate how to learn to map the parameter matrix from large to small without losing the ability of the larger model itself. For additional related work, please refer to Appendix A.

7 Conclusion

In this paper, we propose an efficient attention architecture and a method for fast converting an MHA checkpoint into an efficient structure. By grouping similar heads and performing controlled linear fusion, we develop an initial DHA architecture that decouples head components at various layers, reducing training overhead while maintaining performance. Experimental results show that our method preserves the knowledge of the original model, improving training acceleration, inference efficiency, and computational cost savings. This transformation paradigm offers research value and potential for broader application with minimal performance loss and reduced computational effort.

Acknowledgments

We would like to thank Yinqi Yang, Jiawei Sheng, Xinhua Zhang, Shicheng Wang, Chuanyu Tang and members of the IIE KDsec NLP group for their valuable feedback and discussions. We are very grateful to Mengzhou Xia for providing the concise and effective ShearingLLaMA experimental code and for her assistance during the reproduction process. Work done during Yilong Chen's internship in Baidu Inc. This research is supported by the National Key Research and Development Program of China (grant No.2021YFB3100600) and the Youth Innovation Promotion Association of CAS (Grant No. 2021153).

References

[1] Anthropic. Introducing claude. 2023.

[2] OpenAI. Gpt-4 technical report. *ArXiv*, page abs/2303.08774, 2023.

[3] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models, July 2023.

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

[5] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[6] Farnoosh Javadi, Walid Ahmed, Habib Hajimolahoseini, Foozhan Ataiefard, Mohammad Hassanpour, Saina Asani, Austin Wen, Omar Mohamed Awad, Kangling Liu, and Yang Liu. Gqkva: Efficient pre-training of transformers by grouping queries, keys, and values, 2023.

[7] Saurabh Agarwal, Bilge Acun, Basil Homer, Mostafa Elhoushi, Yejin Lee, Shivaram Venkataraman, Dimitris Papailiopoulos, and Carole-Jean Wu. Chai: Clustered head attention for efficient llm inference, 2024.

[8] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

[9] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023.

[10] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

[11] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[12] Yujia Qin, Cheng Qian, Jing Yi, Weize Chen, Yankai Lin, Xu Han, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Exploring Mode Connectivity for Pre-trained Language Models, October 2022.

[13] Kaiyan Zhang, Ning Ding, Biqing Qi, Xuekai Zhu, Xinwei Long, and Bowen Zhou. CRaSh: Clustering, Removing, and Sharing Enhance Fine-tuning without Full Large Language Model, October 2023.

[14] Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In Hang Li and Lluís Màrquez, editors, *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Cambridge, MA, October 2010. Association for Computational Linguistics.

[15] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured Pruning of Large Language Models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[16] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning, October 2023.

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

[18] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited, 2019.

[19] TogetherAI. Redpajama: An open source recipe to reproduce llama training dataset, 2023.

[20] The Mosaic ML Team. composer. `https://github.com/mosaicml/composer/`, 2021.

[21] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023.

[22] Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In Leon Derczynski, Wei Xu, Alan Ritter, and Tim Baldwin, editors, *Proceedings of the 3rd Workshop on Noisy User-generated Text, NUT@EMNLP 2017, Copenhagen, Denmark, September 7, 2017*, pages 94–106. Association for Computational Linguistics, 2017.

[23] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7432–7439. AAAI Press, 2020.

[24] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press, 2020.

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

[26] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics, 2019.

[27] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457, 2018.

[28] Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*, 2020.

[29] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

[30] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[31] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35, 2022.

[32] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model, 2023.

[33] Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *arXiv preprint arXiv:2305.14387*, 2023.

[34] Di Xie, Jiang Xiong, and Shiliang Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5075–5084. IEEE Computer Society, 2017.

[35] Lemeng Wu, Dilin Wang, and Qiang Liu. Splitting steepest descent for growing neural architectures. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 10655–10665, 2019.

[36] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[37] Peihao Wang, Rameswar Panda, Lucas Torroba Hennigen, Philip Greengard, Leonid Karlinsky, Rogerio Feris, David Daniel Cox, Zhangyang Wang, and Yoon Kim. Learning to Grow Pretrained Models for Efficient Transformer Training, March 2023.

[38] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020.

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

[40] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

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

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

[43] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

[44] Siyu Ding, Junyuan Shang, Shuohuan Wang, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. Ernie-doc: A retrospective long-document modeling transformer. *arXiv preprint arXiv:2012.15688*, 2020.

[45] Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

[46] Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. *arXiv preprint arXiv:2305.14788*, 2023.

[47] Matanel Oren, Michael Hassid, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

[48] Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. NACL: A general and effective KV cache eviction framework for LLM at inference time. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7913–7926, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[49] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

[50] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[51] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.

[52] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[53] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, March 2021.

[54] Lianshang Cai, Linhao Zhang, Dehong Ma, Jun Fan, Daiting Shi, Yi Wu, Zhicong Cheng, Simiu Gu, and Dawei Yin. Pile: Pairwise iterative logits ensemble for multi-teacher labeled distillation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 587–595, 2022.

[55] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. Deja vu: Contextual sparsity for efficient LLMs at inference time. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22137–22176. PMLR, 23–29 Jul 2023.

[56] Elias Frantar and Dan Alistarh. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot, March 2023.

[57] Minjia Zhang and Yuxiong He. Accelerating training of transformer-based language models with progressive layer dropping. *Advances in Neural Information Processing Systems*, 33:14011–14023, 2020.

[58] Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429, 2023.

[59] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[60] [2401.02415] LLaMA Pro: Progressive LLaMA with Block Expansion.

[61] Yilong Chen, Junyuan Shang, Zhenyu Zhang, Shiyao Cui, Tingwen Liu, Shuohuan Wang, Yu Sun, and Hua Wu. LEMON: Reviving stronger and smaller LMs from larger LMs with linear parameter fusion. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8005–8019, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

Appendix

A Extended Related Works, Discussions, and Limitations

A.1 Extended Related Works

Efficient Transformers. EfficientTransformers[38] have been extensively explored [39, 40, 41, 42, 43, 44, 45, 46] to address the self-attention operation which scales quadratically with the sequence length. For instance, Sparse Transformer [39] uses a dilated sliding window the reduces the attention complexity. Longformer [42] and Bigbird [41] reduced the complexity of self-attention by combining random, window and global attention. Recurrence Transformers [43] maintain a memory bank of past KV cache to process the long text in segments. However, the above methods either result in a loss of model performance or require retraining the model, which is unaffordable for the high computational resources of LLMs. DHA requires very little computation to transform checkpoints into an efficient architecture that balances performance and computational resources.

KV Cache Compression. KV Cache Compression methods emerged for reducing the prominent inference bottleneck caused by KV cache, particularly for long content input. A series of methods [8, 10, 9, 47, 48] explored the sparsity among Transformer's attention block, then evicted unnecessary tokens from KV Cache for efficient inference. However, these methods discard information from the context and use algorithms for inference that are inconsistent with the training phase, which can cause model performance degradation. DHA does not need to discard information from the context and is able to maintain consistent performance for training and inference. Pruning [15, 16], quantization [49, 50, 51] and distillation [52, 53, 54] can reduce the number of model key and value headers, parameter dimensions, and activation to reduce memory bandwidth overhead during model inference. Deja Vu [55] and CHAI [7] prune pruning redundant heads through clustering methods for efficient inference. In the LLM era, this leads to a significant reduction in neuron redundancy as models move from task-specific to generalized [56]. The application of these methods to LLMs is computationally expensive and leads to performance degradation at larger pruning magnitudes.

Model Compression. Our approach is dedicated to obtaining a high-performance lightweight language model, which is the same goal as the task of model compression. Quantization [51] reduces the numerical accuracy of model weights and activations, and speeds up training and inference, but results in a loss of model accuracy and the inability to freely build target-specific models. CRash [13] and LayerDrop [57, 58] methods discard ineffective layers during training, which do not allow for target-specific structuring and come with a large performance loss. Pruning [15] minimizes the impact on performance by cutting out redundant neurons that over-parameterize the model. In the LLM era, this leads to a significant reduction in neuron redundancy as models move from task-specific to generalized [56]. Pruning LLM leads to performance degradation at larger pruning magnitudes. LLMsheairng [16] uses the results of pruning as initialization for continuous pre-training of the model to recover performance, but this approach requires more data and computational overhead. We avoid the information loss caused, by learning the parameter fusion matrix of the model to reach a specific structure, thus obtaining better initialization points and reducing the overhead of continuous pre-training.

A.2 Broader Impact and Limitations

A.2.1 Broader Impact

In this paper, we observe the MHA head mechanism and report the phenomenon of modular clustering of heads in MHA. This paper innovatively proposes linearly fusible parameters within the model, and designs linear fusion operators and related experiments to verify the low-loss fusible nature of the parameters. This helps to advance parameter fusion theory and LLM interpretability studies, which provide a foundation and inspiration for future algorithmic advancements, encouraging further optimization and innovation in LLMs. Our work on Decoupled-Head Attention (DHA) represents an advancement in optimizing the efficiency of Large Language Models (LLMs). By addressing the

substantial computational and memory costs associated with the widely used Multi-Head Attention (MHA), DHA enhances the applicability of LLMs in various domains. The introduction of DHA not only achieves a remarkable balance between performance and efficiency but also significantly reduces the need for extensive pre-training, making the deployment of LLMs more feasible and cost-effective. This efficiency allows for the broader accessibility of advanced LLMs, democratizing technology and fostering innovation across industries. Furthermore, by requiring only 0.25% of the original model's pre-training budgets to achieve near-original performance while saving 75% of KV cache, DHA contributes to significant energy savings, aligning with sustainable and environmentally friendly AI practices. The enhanced performance and reduced training costs accelerate the development of AI applications, enhancing productivity in fields such as natural language processing, healthcare, and finance.

A.2.2 Limitation

There are two limitations to our current approach. Firstly, we have only utilized linear methods for parameter fusion in our model. Future research should explore nonlinear methods, as they may offer a better way to link different parameters and achieve optimal results. Secondly, due to computational resource constraints, we have only experimented with models of 7 billion, 3 billion, and 1.3 billion parameters. However, our method is scalable and can be extended to models of any size in future work.

A.2.3 Ethical Consideration

In our study, we utilize publicly available data and techniques to address privacy concerns. Our approach focuses on improving model parameter efficiency and reducing model size to develop robust, compact, and accessible models, thus promoting the open dissemination and democratization of NLP technologies. By implementing pre-training strategies, we aim to mitigate biases through comprehensive training on large datasets, contributing to ethical AI development that prioritizes transparency, efficiency, and bias reduction. Our work is dedicated to advancing accessible and efficient NLP technologies, fostering a more inclusive and automated future for AI.

B More Implementation Details

B.1 Head Similarity and MHA Redundancy

Centered Kernel Alignment (CKA) is a statistical measure used to quantify the similarity between two sets of data representations. Unlike traditional correlation measures, CKA is designed to be invariant to orthogonal transformations and scaling of the data.

To calculate the similarity between two sets of representations using CKA, we employ a kernel function to map the original data into a higher-dimensional space, where the alignment of their central tendencies can be more easily measured. The CKA value ranges from 0 to 1, where 0 indicates no similarity and 1 indicates identical representations.

The mathematical formulation of CKA, when using a linear kernel, is given by the following equation:

$$\text{CKA}(X, Y) = \frac{\|X^T Y\|_F^2}{\sqrt{\|X^T X\|_F^2 \cdot \|Y^T Y\|_F^2}}$$

Here, X and Y are matrices whose columns are the vectors of the representations to be compared, $\|\cdot\|_F$ denotes the Frobenius norm, and X^T and Y^T are the transposes of X and Y, respectively. To mathematically define the redundancy of each layer based on the average similarity between heads, we follow these steps:

1. Compute the similarity between heads: For each pair of heads within a given layer, calculate the similarity using the CKA formula.
2. Compute the average similarity: Average the similarity scores of all pairs of heads to define the redundancy of the layer.

B.1.1 Compute Similarity Between Heads

Consider a layer with H heads, where the parameters of each head are represented by the matrices $\mathbf{W}_i$ (e.g., $\mathbf{W}_{q1}, \mathbf{W}_{q2}, \ldots, \mathbf{W}_{qH}$ for query weights). For each pair of heads i and j, compute the CKA similarity using the following formula:

$$\text{CKA}(\mathbf{W}_i, \mathbf{W}_j) = \frac{\|\mathbf{W}_i^T \mathbf{W}_j\|_F^2}{\sqrt{\|\mathbf{W}_i^T \mathbf{W}_i\|_F^2 \cdot \|\mathbf{W}_j^T \mathbf{W}_j\|_F^2}}$$

B.1.2 Compute Redundancy

Calculate the similarity for all pairs of heads and then compute the average similarity:

$$\text{Redundancy} = \frac{2}{H(H-1)} \sum_{i=1}^{H-1} \sum_{j=i+1}^{H} \text{CKA}(\mathbf{W}_i, \mathbf{W}_j)$$

The coefficient $\frac{2}{H(H-1)}$ ensures that the average similarity is computed over all pairs of heads. This redundancy measure reflects the degree of similarity between the parameters of different heads within each layer. A higher redundancy indicates that the parameters of different heads are more similar, implying a higher level of redundancy.

B.2 Implementation in LLaMA2 Model

Our method can theoretically transform MHA architecture in any transformer model to efficient DHA architecture. Using LLaMA models as case studies, we implemented DHA transformation with various compression rates on all MHA layers. Only a very small number of additional parameters need to be introduced, DHA significantly accelerates training and improves performance.

DHA adaptively gives search heads and heads connectivity relationship with redundancy in each MHA layer. Thus DHA assigns different group sizes at different layers and aggregates similar heads into one group to speed up fusion and reduce knowledge loss due to noise in fusion. As Shown in Fig 4, the transformation process of MHA to DHA can be divided into three stages.

In order to keep the performance of the DHA model at the fusion start consistent with the MHA model, we initialize the operators of the DHA model to the MHA model with the corresponding scaling factors of query-key, query-value set to 1, and the corresponding scaling factors within the rest of the groups set to 0. At the beginning of every fusion process (e.g. $2\times, 4\times, 8\times$), the algorithm first performs multiple STEPs constrained only by the $\mathcal{L}_{\text{lm}}$ constraints to propagation, computing the $\mathcal{L}_{\text{fusion}}$ but not optimizing the linear fusion operator based on it. Based on the $\mathcal{L}_{\text{fusion}}$ between head and head as a measure of the distance between head and head we perform head clustering with the goal of minimizing the average loss of heads within each group and maximizing the average loss of heads between groups and groups. Afterwards, we select multiple groups with the smallest loss based on the compression rate as the fusion target, and optimize their $\mathcal{L}_{\text{fusion}}$ for back propagation. This algorithm ensures that the most redundant components of the model are fused and compressed during each transformation, while components requiring more parameters retain their original properties.

Our approach is theoretically applicable to transforming parameters across various transformer model designs, focusing on preserving the knowledge within MHA parameters.

Using LLaMA models as a case study, we implement our DHA transformation on all MHA layer. The whole transformation process can be divided into two phases: the Fusion phase with a small training budget and the recovery phase with continuous pre-training. Before Fusion phase, we define the total number of compressed headers budget C then C is split into compression rates at different compression levels. During Fusion phase, we modified the forward propagation path of MHA in the form of DHA refer to Eq. 5 and optimize $\mathcal{L}_{\text{fusion}}$ refer to Eq. 7. At the beginning, the fusion operators of each layer will be initialized making the DHA and the original MHA functionally equivalent. As we antagonistically optimize the fusion operator and upadte Lagrangian multipliers λ, the $\mathcal{L}_{\text{fusion}}$ that marks this DHA fusion process decreases.

When $\mathcal{L}_{\text{fusion}}$ is less than 1e-3 we terminate the fusion algorithm and enter the post-processing phase. The fusion weights within each group are computed by averaging the weights corresponding to each query-key and query-value within the group.We construct new DHA heads' parameters from the original MHA heads based on the fusion operator. After that, the fused model parameters can recover the performance and complete the transformation with a small amount of restorative pre-training.

We implemented the DHA algorithm with different compression ratios on models of different sizes. Experiments show that the DHA algorithm is adapted to models of various sizes. Only a very small number of additional parameters need to be introduced, and DHA preserves parameter knowledge in the model and improves performance.

B.2.1 Attention Module Initialization

In the module initialization process, the input key and value tensors are first reshaped and grouped according to the number of key and value heads, respectively. Given the batch size (bsz), number of heads (num_heads), key length (k_len), and head dimension (head_dim), the key tensor is reshaped into keys_grouped of shape [bsz, num_key_heads, num_heads // num_key_heads, k_len, head_dim]. Similarly, the value tensor is reshaped into values_grouped of shape [bsz, num_value_heads, num_heads // num_value_heads, k_len, head_dim]. These grouped tensors are then expanded by repeating them along the group size dimension, resulting in keys_expanded and values_expanded. Correspondingly, the weight tensors weights_k and weights_v are reshaped to match the expanded dimensions and are then multiplied element-wise with the expanded key and value tensors.

Algorithm 1 Attention Module Initialization

Require: K ▷ key tensor
Require: V ▷ value tensor
Ensure: K' ▷ weighted key tensor
Ensure: V' ▷ weighted value tensor

$b \leftarrow$ batch size
$H \leftarrow$ number of heads
$L_k \leftarrow$ key length
$D \leftarrow$ head dimension
$K_s \leftarrow$ self.num_key_heads
$V_s \leftarrow$ self.num_value_heads
$K_g \leftarrow$ self.key_group_size
$V_g \leftarrow$ self.value_group_size
$K'_g \leftarrow$ self.weights_k
$V'_g \leftarrow$ self.weights_v
$K_g \leftarrow K$.view$(b, K_s, H/K_s, L_k, D)$
$V_g \leftarrow V$.view$(b, V_s, H/V_s, L_k, D)$
$K_e \leftarrow K_g$.repeat_interleave$(K_g, \text{dim} = 1)$
$V_e \leftarrow V_g$.repeat_interleave$(V_g, \text{dim} = 1)$
$K_w \leftarrow K'_g$.view$(1, H, K_g, 1, D)$
$V_w \leftarrow V'_g$.view$(1, H, V_g, 1, D)$
$W_K \leftarrow K_e \times K_w$
$W_V \leftarrow V_e \times V_w$
$K' \leftarrow W_K$.sum$(\text{dim} = 2)$
$V' \leftarrow W_V$.sum$(\text{dim} = 2)$
return K', V'

B.2.2 Attention Forward Pass

During the forward pass, the reshaping and expansion of the key and value tensors are performed in a similar manner as in the initialization process but with parameters specific to the DHA fusion phase. The key tensor is reshaped into keys_grouped of shape [bsz, dha_warmup_group_num, num_heads // dha_warmup_group_num, k_len, head_dim] and the value tensor into values_grouped of shape [bsz, dha_warmup_group_num, num_heads // dha_warmup_group_num, k_len, head_dim]. These

grouped tensors are then expanded by repeating them according to the dha_warmup_group_size. The weights weights_k and weights_v are reshaped and expanded to align with the dimensions of the expanded key and value tensors. Element-wise multiplication is performed between the expanded tensors and their corresponding weights, and the resulting weighted tensors are summed along the appropriate dimension.

Algorithm 2 Attention Forward Pass

Require: K ▷ key tensor
Require: V ▷ value tensor
Ensure: K' ▷ weighted key tensor
Ensure: V' ▷ weighted value tensor

$b \leftarrow$ batch size
$H \leftarrow$ number of heads
$L_k \leftarrow$ key length
$D \leftarrow$ head dimension
$G_q \leftarrow$ self.dha_warmup_group_num
$G_s \leftarrow$ self.dha_warmup_group_size
$K'_g \leftarrow$ self.weights_k
$V'_g \leftarrow$ self.weights_v
$K_g \leftarrow K$.view$(b, G_q, H/G_q, L_k, D)$
$V_g \leftarrow V$.view$(b, G_q, H/G_q, L_k, D)$
$K_e \leftarrow K_g$.repeat_interleave$(G_s, \text{dim} = 1)$
$V_e \leftarrow V_g$.repeat_interleave$(G_s, \text{dim} = 1)$
$K_w \leftarrow K'_g$.view$(1, H, G_s, 1, D)$
$V_w \leftarrow V'_g$.view$(1, H, G_s, 1, D)$
$W_K \leftarrow K_e \times K_w$
$W_V \leftarrow V_e \times V_w$
$K' \leftarrow W_K$.sum$(\text{dim} = 2)$
$V' \leftarrow W_V$.sum$(\text{dim} = 2)$
return K', V'

B.2.3 DHA Loss Calculation

The calculation of the loss function in this model involves the adaptive DHA loss. This loss is computed based on the global step, warmup steps, and a base value. The DHA margin is calculated as the product of an exponential decay term and a linear decay term, ensuring it is non-negative. The adaptive DHA loss is derived by comparing the mean squared error (MSE) with the DHA margin and summing the positive differences.

Formally, the DHA margin M_{dha} is calculated as:

$$M_{\text{dha}} = \max\left(0, \left(\text{base}^{\text{global\_step}}\right) \times \left(1.0 - \frac{\text{global\_step}}{\text{dha\_warmup\_step}}\right)\right)$$

MSE Loss are defined in Eq. 7. The adaptive DHA loss L_{dha} is then:

$$L_{\text{dha}} = \sum \max(\text{mse} - M_{\text{dha}}, 0.0)$$

The overall loss L is the adaptive DHA loss:

$$L = L_{\text{dha}} + L_{\text{lm}}$$

This combined loss function effectively utilizes the adaptive component to optimize the attention mechanism in the model. The calculation process ensures that the model adapts dynamically during training, reducing the loss progressively as the training steps increase.

Algorithm 3 Adaptive DHA Loss Calculation

Require: B ▷ blocks
Require: mse ▷ Mean Squared Error tensor
Ensure: L ▷ loss_dha_diversity

$\lambda \leftarrow$ self.lambda_mse
$s \leftarrow$ self.mse_scale
$L \leftarrow 0.0$
$global\_step \leftarrow 1000$
$dha\_warmup\_step \leftarrow 200$
$base \leftarrow 0.999$
for each $b \in B$ **do**
  $L_l \leftarrow 0.0$ ▷ loss_dha_diversity_layer
  $A \leftarrow b.attn$ ▷ attn_layer
  $W_k \leftarrow A.weights\_k$
  $W_v \leftarrow A.weights\_v$
  $G_s \leftarrow A.dha\_warmup\_group\_size$
  $G_n \leftarrow A.dha\_warmup\_group\_num$
  $H_{kv} \leftarrow A.num\_key\_value\_heads$
  $H \leftarrow A.num\_heads$
  $D \leftarrow A.head\_dim$
  $W_k \leftarrow \text{reshape}(W_k, [H_{kv}, -1, G_s, D])$
  $W_v \leftarrow \text{reshape}(W_v, [H_{kv}, -1, G_s, D])$
  for each $r \in W_k$ **do**
    for each $o \in W_k$ after r **do**
      $L_l \leftarrow L_l + \text{MSE\_Loss}(r, o)$
    end for
  end for
  for each $r \in W_v$ **do**
    for each $o \in W_v$ after r **do**
      $L_l \leftarrow L_l + \text{MSE\_Loss}(r, o)$
    end for
  end for
  $N \leftarrow H \times G_s \times D \times 2 \times \frac{(G_s-1)}{2}$
  $L_l \leftarrow \frac{s \times L_l}{N}$
  $L \leftarrow L + L_l$
end for
$L \leftarrow \frac{L}{\text{len}(B)}$
$L \leftarrow L \times \lambda$
$exponent \leftarrow base^{global\_step}$
$linear\_decay \leftarrow 1.0 - \frac{global\_step}{dha\_warmup\_step}$
$margin \leftarrow \max(0, exponent \times linear\_decay)$
$adaptive\_loss \leftarrow \sum \max(mse - margin, 0.0)$
$L \leftarrow L + adaptive\_loss$
return L

B.3 Head Grouping Based on Fusion Loss

This algorithm uses simulated annealing to optimize group scores based on a given score matrix. It begins by defining the number of groups and distributing the points among them randomly. The initial score for these groups is calculated using the 'calculate_score' function, which sums the scores from the matrix for each group, considering each connection twice and dividing by two.

The algorithm starts with a high temperature (T=100) and gradually cools down (T_min=0.001) using a cooling rate (alpha=0.9). During each iteration, two random points from different groups are swapped, creating a new grouping. The score for this new grouping is calculated, and the difference in score (delta) is evaluated.

If the new score is higher, or if a randomly generated number is less than the exponential of delta divided by the temperature, the new grouping is accepted. This allows the algorithm to escape local optima. The temperature is then reduced according to the cooling rate. This process continues until the temperature reaches the minimum threshold. The algorithm returns the final group configuration and its corresponding score, which represents an optimized grouping based on the initial score matrix.

In practice, we use the MSE computed by the head and the head as scores, and compute the matrix of scores between the head and the head for head clustering after forward.

Algorithm 4 Head Grouping Optimization on Fusion Loss

Require: M ▷ score matrix
Require: $G \leftarrow 8$ ▷ number of groups
Ensure: $best\_groups, best\_score$ ▷ final groups and their score

function CALCULATE_SCORE($M, groups$)
    $score \leftarrow 0$
    for each $group \in groups$ **do**
        for each $i \in group$ **do**
            for each $j \in group$ **do**
                $score \leftarrow score + M[i][j]$
            end for
        end for
    end for
    return $score/2$ ▷ each connection counted twice
end function
function SIMULATED_ANNEALING(M, G)
    $P \leftarrow$ length(M) ▷ number of points
    $N \leftarrow P/G$ ▷ number of points per group
    $points \leftarrow$ array$(range(P))$
    shuffle$(points)$
    $groups \leftarrow points.reshape(G, N)$
    $current\_score \leftarrow$ CALCULATE_SCORE$(M, groups)$
    $T \leftarrow 100.0$ ▷ initial temperature
    $T_{min} \leftarrow 0.001$ ▷ minimum temperature
    $\alpha \leftarrow 0.9$ ▷ cooling rate
    while $T > T_{min}$ **do**
        $i, j \leftarrow$ random integers in $[0, G)$
        if $i \neq j$ **then**
            $a, b \leftarrow$ random integers in $[0, N)$
            $new\_groups \leftarrow groups.copy()$
            $temp \leftarrow new\_groups[i][a]$
            $new\_groups[i][a] \leftarrow new\_groups[j][b]$
            $new\_groups[j][b] \leftarrow temp$
            $new\_score \leftarrow$ CALCULATE_SCORE(M, new_groups)
            $\Delta \leftarrow new\_score - current\_score$
            if $\Delta > 0$ **or** $\exp(\Delta/T) >$ random() **then**
                $groups \leftarrow new\_groups$
                $current\_score \leftarrow new\_score$
            end if
        end if
        $T \leftarrow T \times \alpha$ ▷ cooling down
    end while
    return $groups, current\_score$
end function
$best\_groups, best\_score \leftarrow$ SIMULATED_ANNEALING(M, G)

B.4 Layer Allocation Based on Fusion Loss

This algorithm efficiently allocates resources to different layers based on their respective losses to optimize system performance. Initially, it assigns a minimum allocation to each layer. Then, it calculates weights for each layer based on their losses, prioritizing layers with higher losses. The algorithm determines the number of times 16 can be allocated based on the remaining allocation. It allocates 16s to layers with the highest weights until reaching a predetermined limit. Next, it redistributes the remaining allocation to layers with the highest loss-to-allocation ratios, assigning resources in multiples of 8 or 4. This process ensures that layers with higher losses receive more resources, optimizing the overall system performance. Finally, the algorithm returns the final allocation for each layer, resulting in an efficient distribution of resources across the system. The total search process for the LLaMA2 model requires 42 minutes.

Algorithm 5 Layer Allocation Based on Losses

Require: L ▷ losses for each layer
Require: $A \leftarrow [4, 8, 16]$ ▷ possible allocations
Require: $T \leftarrow 256$ ▷ total allocation
Ensure: $alloc \leftarrow [a_1, a_2, \ldots, a_n]$ ▷ final allocations for each layer
$n \leftarrow \text{length}(L)$
$alloc \leftarrow [4] \times n$ ▷ initial allocation
$W \leftarrow \frac{L}{\sum L}$ ▷ weights proportional to losses
$R \leftarrow T - \sum alloc$ ▷ remaining allocation
$k \leftarrow 1$ ▷ initial number of 16's to allocate
$M_{16} \leftarrow R//16$ ▷ maximum number of 16's that can be allocated
$k \leftarrow \min(k, M_{16})$
for $i \leftarrow 1$ to k **do**
    $idx \leftarrow \text{argmax}(W)$
    $alloc[idx] \leftarrow alloc[idx] + 12$
    $W[idx] \leftarrow 0$ ▷ prevent reallocation
end for
$R \leftarrow T - \sum alloc$
while $R > 0$ **do**
    if $R \geq 8$ **then**
        $idx \leftarrow \text{argmax}(\frac{L}{alloc})$
        $alloc[idx] \leftarrow alloc[idx] + 4$
        $R \leftarrow R - 4$
    else if $R \geq 4$ **then**
        $idx \leftarrow \text{argmax}(\frac{L}{alloc})$
        $alloc[idx] \leftarrow alloc[idx] + 4$
        $R \leftarrow R - 4$
    end if
end while
return $alloc$

B.5 Training Details

The hyperparameters used in our experiments are presented in Tab. 4. We employ fully sharded data parallel to efficiently train our models in parallel, and we utilize FlashAttention V1 [59] to accelerate the training process. A cosine learning rate scheduler is used, with the learning rate decaying to a minimum of 10% of the peak value. Preliminary experiments were conducted to determine the optimal peak learning rate for learning the fusion variables and Lagrange multipliers.

Table 4: Training hyper-parameters

	Fusion	Contined Pre-training
Training budget	0.2B	5B
Learning rate of ω, λ	0.05	-
Learning Rate of θ	0.0001	0.0001
LR warmup ratio	10%	3%
Batch size (tokens)	262K	1M
Evaluation interval m (steps)	40	40
Steps	800	5,000
# GPUs	8	8

C Extended Experiments

C.1 Instruction Tuning Evaluation.

Instruction Tuning Evaluation. To assess our models' capabilities in downstream application after instruct tuning [31, 32], we fine-tune both DHA and the baseline models on 10,000 instruction-response pairs drawn from the initial round of multi-turn chat histories in the ShareGPT dataset[7]. For evaluation, we select another 1,000 instructions from ShareGPT, generate responses using our fine-tuned models and other baseline models and employ GPT-4 as an evaluator to compare these responses [33]. We report the win rate of our model relative to the baseline model.

Instruction Tuning. As shown in Fig. 10, the tuned DHA model outperforms all GQA baselines of comparable scale . This demonstrates that the DHA model effectively retains the foundational capabilities of the MHA model and can be activated through instruction tuning to produce long, coherent, and informative responses.

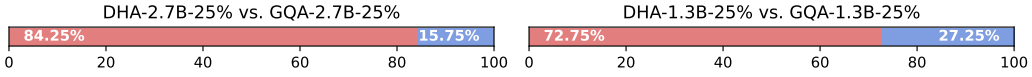


Figure 10: In model scale of 7B, 3B, and 1.3B, DHA significantly outperforms GQA and achieves comparable performance with MHA after instruction tuning .

Combination with KV Cache Compression Techniques. In Sections 2 and 4, we demonstrated that DHA is a more efficient GQA architecture, so it has similarly good compatibility. We tested the compatibility of the DHA model with the KVCache eviction method NACL [48]. NACL 25% indicates retaining only 25% of the KVCache. The experiment results are shown in the Tab. 5. DHA and GQA exhibit equally good compatibility with KV cache compression techniques.

Compare with Advance GQA Initialization. It's a common and effective approach to convert MHA to GQA using mean pooling instead of training from scratch. The author of GQA tested several methods for the initialization of GQA and found it works best using simple mean pooling from MHA. Indeed, training GQA from scratch will cost trillions tokens budget to match the performance of MHA which is inefficient and costly.Inspired by the similarity of head parameters, we improved the initialization method of GQA: instead of direct grouping, we first cluster similar heads using CKA

[7] `https://sharegpt.com`

Table 5: Comparison of log(PPL) between DHA and GQA with NACL.

Method	log(PPL)
GQA-7b-25%	2.89
DHA-7b-25%	2.84
GQA-7b-25% (NACL 25%)	3.01
DHA-7b-25% (NACL 25%)	2.93

Table 6: Comparison of Avg ACC and PPL between different methods at 7B-25% (5B).

Method	Avg ACC	PPL
DHA-7B-25% (5B)	62.4	7.29
GQA-7B-25% (5B)	60.3	7.54
GQA (CKA-Grouping)-7B-25% (5B)	60.4	7.51

and then perform mean-pooling initialization within each cluster. We compare this approach with the Vanilla GQA and DHA.

Tab. 6 shows that GQA(CKA-Grouping)-7B-25% (5B) achieved comparable performance to the original implementation in Vanilla GQA. We believe the reason for this is that the head grouping learned by DHA is based on the fusible nature between heads, which cannot be completely equated with CKA similarity. More importantly, DHA not only groups heads based on similarity but also learns the fusible parameters. This allows it to eliminate the influence of redundant parameters and retain more important information during the initialization process, which is not possible with mean initialization.

D Extend Analysis

How Merging Weights Change. Refer to Fig. 3a, where we show the weight variation diagram. In the fusion process of heads 0-3, head 0 initially constitutes 100% as the starting head of the MHA. As the fusion process progresses, the parameters of the important heads increase, and the proportions of all heads become more balanced. This indicates that the algorithm attempts to retain information from different heads by balancing the parameter proportions of each head. This process results in a slight increase in loss, but not significantly.

DHA's Compatibility on GQA Model. DHA is primarily designed for models based on the Transformer Decoder architecture and can be adapted to all models with this architecture. We chose LLaMA [60] as the experimental baseline because it is a classic model using the decoder architecture in LLMs. Other open-source LLM models differ from LLaMA only in certain details (such as activation functions and training methods), which do not affect DHA's training. Successfully applying DHA to LLaMA indicates that it can be used in most decoder-only models. GQA [5] is an efficient variant of MHA, which optimizes the inference process through head grouping and sharing. Due to its simplicity and efficiency, GQA is widely used. DHA can be similarly constructed based on GQA, requiring only minor adjustments to the construction process. Here, we provide two feasible methods to convert GQA to DHA.

- Easiest method in less than 1 minute. GQA can be losslessly converted into MHA by simply replicating the GQA' KV heads. Then, we can perform the DHA transformation on the MHA architecture.
- Minor modification by grouping KV. DHA only needs to group and fuse the Key and Value heads. When constructing DHA on GQA, we initially group the Key and Value, maintaining alignment with GQA functionality. During the training phase, the fused head parameters can replace the original GQA heads for sharing.

Inter-layer Grouping of Heads or Only Intra-layer Grouping? Only intra-layer grouping and fusion is conducted in DHA. Fig. 1 meant to illustrate the decoupled-heads where the number of key

and value heads can be different among layers. The DHA method employs parameter fusion within each layer for three reasons:

- Higher redundancy of heads within layer for fusion. The heads within a layer exhibit high similarity and redundancy, which provides a good starting point for parameter fusion.
- More complex optimization for inter-layer fusion. The optimization process between layers is very complex and requires memory operations for cross-layer calls, which inherently increases the inference cost.
- Promising future work by introducing inter-layer fusion [61]. This paper represents an early exploration of applying parameter fusion methods within model parameters. The inter-layer fusion approach is indeed a valuable direction for future exploration.

Accuracy Loss after Transformation. The performance gap between the results shown in the paper and MHA is primarily due to the following two reasons:

- The gap of pre-training data. The MHA model was not trained on the same data used for DHA. Since LLaMA's training data is not directly open-sourced, we used an experimental open-sourced pre-training data following Sheared-LLaMA (Xia et al., 2024). The improved pre-training data will close the gap between DHA and MHA.
- Parameter size difference. Compared to MHA, DHA compresses 50% or 25% of attention heads, requires only 0.05% of pre-training data and achieves approximately 5% loss. The number of parameters of MHA is much larger than that of DHA, so performance loss is inevitable during conversion. Compared with GQA, a strong baseline with the same number of parameters, DHA has shown higher training efficiency and performance advantages. Due to the high efficiency of DHA, DHA can use more heads than MHA with the same number of parameters, and has the opportunity to achieve better performance.

E Extend Observation

E.1 Header parameter characteristics in MHA

We show more of our head similarity observations in the LLaMA2-7b model MHA. Each subfigure represents the similarity between heads within the same layer for three different types of attention mechanisms: WQ (query), WK (key), and WV (value). The matrices are arranged in a 3x4 grid layout, with each row corresponding to a specific layer and each column corresponding to a type of attention mechanism. Note: Layer numbers start from 1.

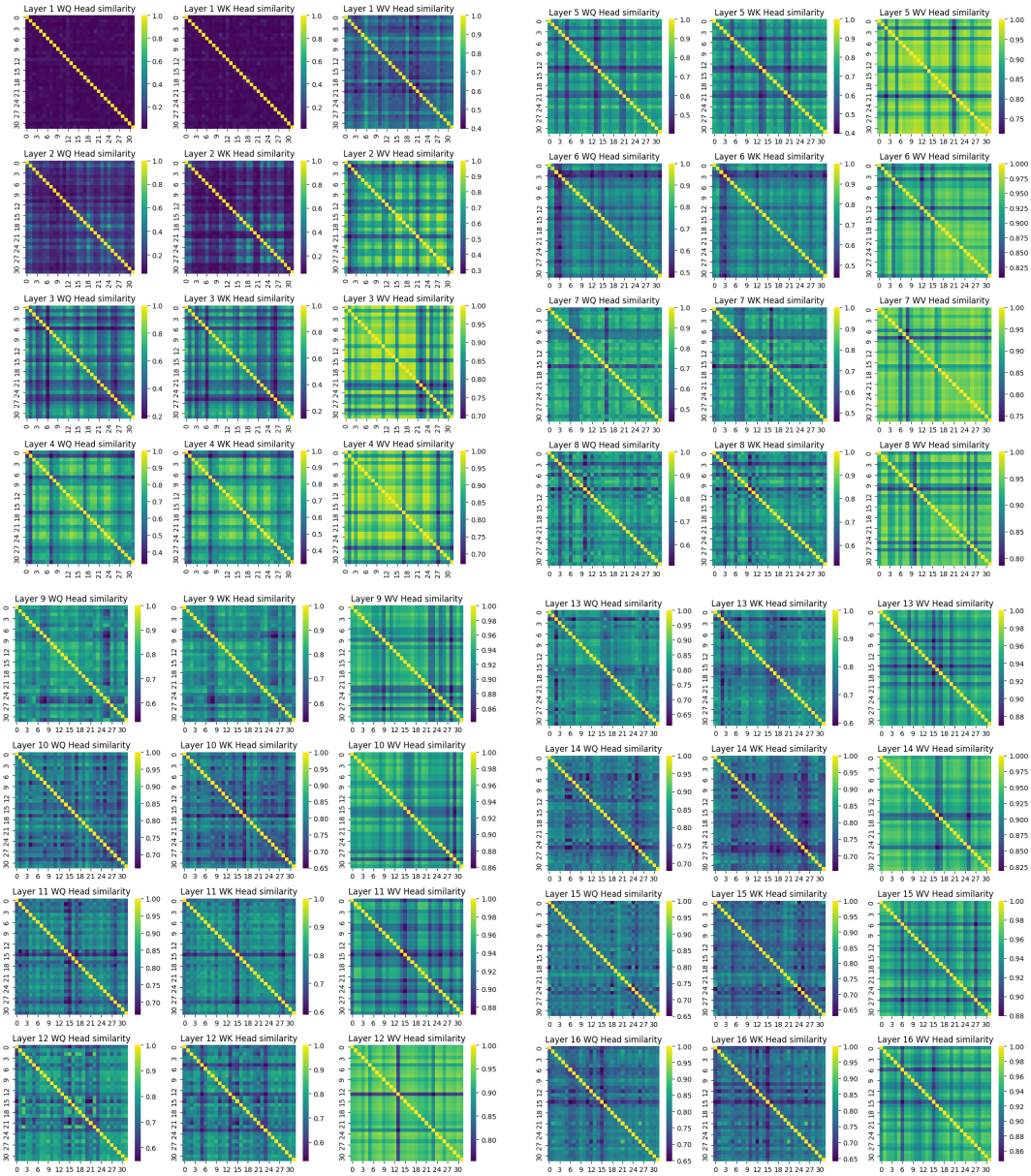


Figure 11: Visualization of query, key, value head parameters similarity from layer 1 to layer 1b in LLaMA2-7B.

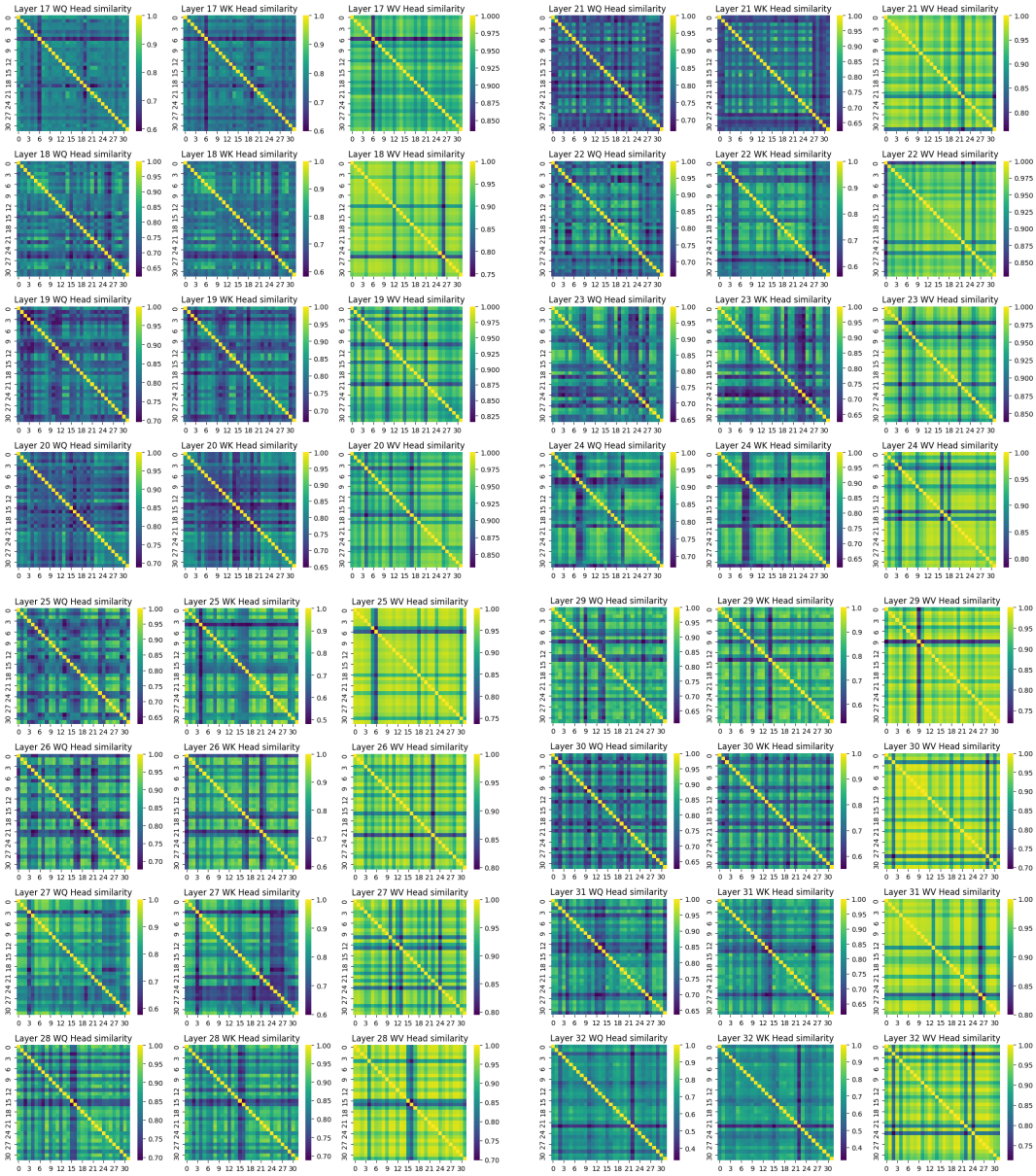


Figure 12: Visualization of query, key, value head parameters similarity from layer 17 to layer 32 in LLaMA2-7B.

E.2 Header parameter characteristics in DHA

The DHA parameter distribution of shows consistency with MHA's. It indicates that DHA effectively aggregates multiple similar functional heads within clusters and new fused heads successfully reconstruct the functionalities of multiple origin heads in MHA. It is noteworthy that the significant reduction in the number of similar heads within the DHA architecture indicates that our method effectively reduces redundancy among the heads.

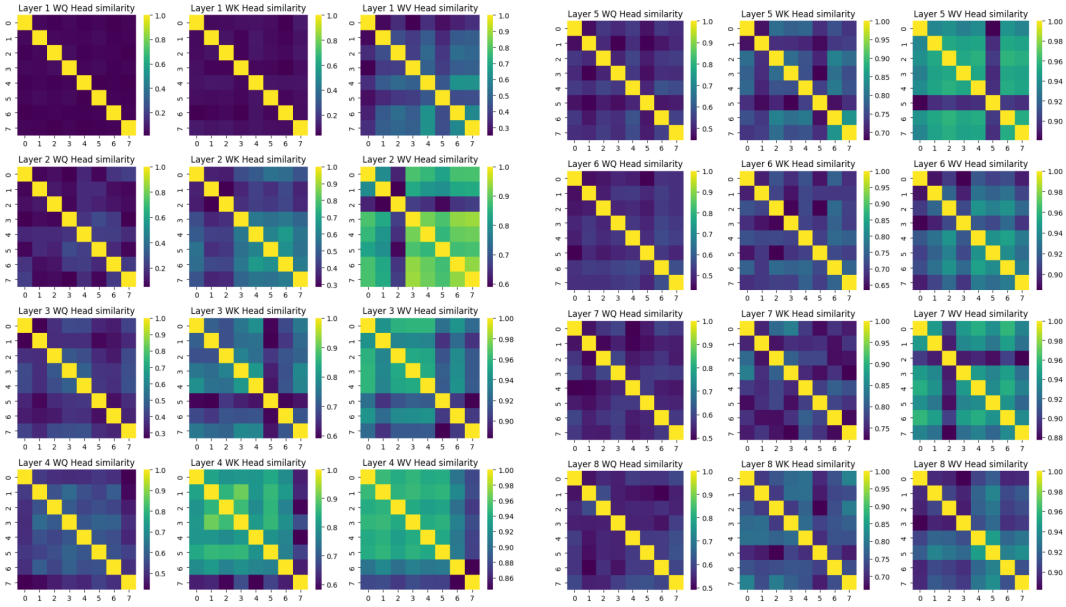


Figure 13: Visualization of query, key, value head parameters similarity from layer 1 to layer 8 in DHA-7B-25%.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract provides a concise summary of the key findings and experiment results. The introduction in Sec. 1 outlines the research questions and objectives in paragraph 3,4 and contribution in paragraph 5.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The paper discusses the limitations of the work performed by the authors in detail in Appendix Appendix. A.2, highlighting two specific limitations and the broader impact.

   Guidelines:

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

3. **Theory Assumptions and Proofs**

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

Answer: [Yes]

Justification: This paper is mainly based on observation, making conjectures and methods and proving the effects through experiments. The paper defines the background in Sec. 2, presents the conjecture in Sec. 3, and provides a detailed derivation of the form and optimization process in Sec. 4. All assumptions made in the paper are thoroughly validated through experiments in Sec. 5.

Guidelines:

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

4. **Experimental Result Reproducibility**

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

Answer: [Yes]

Justification: The paper provides a detailed description of the experimental data setup and hyperparameter settings in Sec. 5. Additionally, in Sec. 4 and in Appendix. B.2 sections we thoroughly explain the derivation and implementation process, ensuring all necessary information for reproducing the main experimental results is disclosed.

Guidelines:

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

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [No]

   Justification: The datasets, baseline methods, and models used in the paper are fully open-source and available on Hugging Face. The paper includes the key implementation steps and code in Sec. 4 and the Appendix. B.2. However, the complete code is still being organized and is under consideration for open sourcing.

   Guidelines:

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

6. **Experimental Setting/Details**

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

   Answer: [Yes] ,

   Justification: The paper provides a detailed description of the experimental data setup and hyperparameter settings in Sec. 5. Additionally, in Sec. 4 and in Appendix. B.2 sections we thoroughly explain the derivation and implementation process.

   Guidelines:

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

7. **Experiment Statistical Significance**

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

   Answer: [Yes]

Justification: All results are averaged over multiple tests, and we report the mean accuracy along with the standard deviation (acc_norm) as a measure of error bars.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

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

   Answer: [Yes]

   Justification: In Sec. 5.1, we report the GPUs we used, the memory, and detailed training information. For more information you can refer to the Appendix B.2.

   Guidelines:

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

9. **Code Of Ethics**

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

   Answer:[Yes]

   Justification: The discussion of the ethics and impact can be consulted in Appendix. A.2. We are open and transparent throughout the study and do not design for human subjects, privacy data bias, or other issues.

   Guidelines:

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

10. **Broader Impacts**

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

    Answer: [Yes]

    Justification: The discussion of the broader impacts can be consulted in Appendix. A.2.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [NA]

    Justification: This paper presents an improved approach based on the existing model architecture, but does not release any new models. The paper poses no such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: This article uses assets reasonably in compliance with the license, and the assets used are cited in the article.

Guidelines:

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

13. **New Assets**

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

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

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

14. **Crowdsourcing and Research with Human Subjects**

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

    Answer: [NA]

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

    Guidelines:

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

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

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

    Answer: [NA]

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

Guidelines:

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