Improving Transformers with Dynamically Composable Multi-Head Attention

Da Xiao[1] Qingye Meng[2] Shengping Li[2] Xingyuan Yuan[2]

Abstract

Multi-Head Attention (MHA) is a key component of Transformer. In MHA, attention heads work independently, causing problems such as low-rank bottleneck of attention score matrices and head redundancy. We propose Dynamically Composable Multi-Head Attention (DCMHA), a parameter and computation efficient attention architecture that tackles the shortcomings of MHA and increases the expressive power of the model by dynamically composing attention heads. At the core of DCMHA is a `Compose` function that transforms the attention score and weight matrices in an input-dependent way. DCMHA can be used as a drop-in replacement of MHA in any transformer architecture to obtain the corresponding DCFormer. DCFormer significantly outperforms Transformer on different architectures and model scales in language modeling, matching the performance of models with ~1.7×–2.0× compute. For example, DCPythia-6.9B outperforms open source Pythia-12B on both pretraining perplexity and downstream task evaluation. The code and models are available at `https://github.com/Caiyun-AI/DCFormer`.

1. Introduction

Transformers (Vaswani et al., 2017) have become the state-of-the-art model for various domains and tasks and the de facto backbone for foundation models. Multi-head attention (MHA) is a key component of Transformer responsible for information exchange between tokens. MHA allows the model to jointly attend to information from different representation subspaces at different positions. An important feature of MHA is that multiple attention heads work in parallel and independent of each other. While being simple and empirically successful, this choice leads to some drawbacks, such as the low-rank bottleneck of attention score matrices (Bhojanapalli et al., 2020; 2021) which decreases expressive power, and the issue of redundant heads (Voita et al., 2019; Michel et al., 2019) which causes waste of parameters and computation.

There has been a number of works in the literature that try to improve MHA by introducing some form of interaction or collaboration between heads. We categorize these works and motivate ours from the viewpoint of *head composition*: composing new heads from a fixed number of "base heads".

Head composition can be performed in different places in the computation graph of MHA. A common form of head composition is to use more sophisticated ways of combining / selecting the outputs of multiple heads to replace the simple concatenate-then-project approach of MHA, either statically (Ahmed et al., 2017) or dynamically (Li et al., 2019; Zhang et al., 2022). Operating at the uppermost level of MHA computation, this is only a "surface" form of composition: heads still operate on their own and the underlying information flow between tokens is unchanged. Due to this nature, it is generally lightweight and efficient, but at the same time the expressive power gain is limited.

Operating at the lowest level, another contrasting approach is to compose the linear projections W^Q, W^K and W^V of heads in MHA (Cordonnier et al., 2020; Liu et al., 2022). Projection composition allows genuinely new heads to be composed with actually changed information flow, which is a prerequisite for fundamental improvement of expressive power. Though theoretically being more parameter-efficient by sharing parameters among projections, this approach usually incurs large computation cost in practice. Besides, the composition is *static*, lacking adaptability to inputs. The potential of head composition can not be fully realized.

We opt for a third middle ground approach of composing the attention score and/or attention weight matrices (collectively referred to as *attention matrices* in this paper) (Shazeer et al., 2020; Wang et al., 2022; Nguyen et al., 2022). Attention matrix composition has some equivalent relationship with projection composition (see Section 2), ensuring fundamental improvement in expressive power compared with head output composition. Thanks to the smaller computation cost compared with projection composition and a careful design,

[1]School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China [2]ColorfulClouds Technology Co.,Ltd., Beijing, China. Correspondence to: Da Xiao <xiaoda99@bupt.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Table 1. Comparison between head composition approaches. (*: parameter-efficient and computation-inefficient)

comp. level	methods	attn. scores	attn. weights	dyn-amic	effic-iency
attn. outputs	Weighted TFM (Ahmed et al., 2017)	✗	✗	✗	✓
	Routing-by-Agreement (Li et al., 2019) MoA(Zhang et al., 2022)	✗	✗	✓	✓
projections	Collab. Attn. (Cordonnier et al., 2020) Tuformer(Liu et al., 2022)	✓	✗	✗	✓✗*
attn. matrices	MHDC(Wang et al., 2022) FiSHformer(Nguyen et al., 2022)	✓	✗	✗	✓
	THA (Shazeer et al., 2020)	✓	✓	✗	✓
	THA dynamic	✓	✓	✓	✗
	DCMHA (ours)	✓	✓	✓	✓

we are able to make the composition *dynamic*: new heads are composed on-the-fly dependent on the inputs, further increasing model expressiveness. In contrast to existing work, we seek to meet the requirements of true compositionality, dynamicity and efficiency simultaneously (see Table 1).

In this work, we propose Dynamically Composable Multi-Head Attention (DCMHA), a parameter and computation efficient attention architecture that tackles the shortcomings of MHA and increases the expressive power of the model by dynamically composing attention heads. At the core of DCMHA is a `Compose` function that transforms the attention score and weight matrices in an input-dependent way. DCMHA can be used as a drop-in replacement of MHA in any transformer architecture to obtain the corresponding DCFormer. We implement DCMHA / DCFormer and conduct analysis and extensive experiments to evaluate its effectiveness, efficiency and scalability. Experimental results show that DCFormer significantly outperforms Transformer on different architectures (original or the advanced LLaMA architecture) and model scales (from 405M to 6.9B) in language modeling, matching the performance of models with ~1.7×-2× compute. For example, DCPythia-6.9B outperforms open source Pythia-12B on both pretraining perplexity and downstream task evaluation. We also apply DCMHA to vision transformers for image classification and do some preliminary analysis on the DCPythia-6.9B model with a synthetic dataset to better understand why and how DCMHA works.

2. Head Composition by Transforming Attention Matrices

As mentioned in Section 1, DCMHA achieves head composition by composing attention matrices. In this section we introduce the concept of attention matrix composition, show its role in increasing model expressiveness and discuss its relationship with projection composition.

Notation Suppose T and S are query and key sequence lengths. We use $A_h \in \mathbb{R}^{T\times S}$ to denote the *attention matrix* of hth head, which can be either the attention score (before softmax) or weight (after softmax) matrix computed by an MHA module with H heads. We can stack the H attention matrices into a tensor $A = \text{Stack}(\{A_h\}_{h=1}^{H}) \in \mathbb{R}^{H\times T\times S}$. We use $A_{:ij} = A[:, i, j] \in R^H$ to denote the *attention vector* between a pair query vector Q_i and key vector K_j.

By *attention matrix composition*, we can compose H new attention matrices $\{A'_h\}_{h=1}^{H}$ as follows: the hth composed matrix A'_h is a linear combination of H base matrices: $A'_h = \sum_{j=1}^{H} C_{hj} A_j$, where $C \in \mathbb{R}^{H\times H}$ is the *composition map*.

We illustrate the functionality of matrix composition through several simplified and prototypical composition maps in Figure 1 (here assume we are composing attention weight matrices). Different patterns of composition maps have different functions. Figure 1 (a) shows the case of mutual excitation (between Heads 3 and 8) and inhibition (between Heads 2 and 5). Mutual excitation is helpful when two heads are positively related: if one head is active, the other should also be active. Mutual inhibition is the opposite. Figure 1 (b) shows one-to-many sharing, where Head 6 shares its attention weights to Heads 4 and 7. Figure 1 (c) shows many-to-one sharing, where Heads 3 and 7 shares their weights to Head 1. Assume that Head 1 has an OV circuit[1] that can transform a concrete noun to its hypernym/superclass (e.g. 'apple' -> 'fruit'), but it may have difficulty in attending to the right token in the context to apply the transformation due to inefficacy of its QK circuits.[2] With the help of attention weight composition, it can now "borrow" attention weights from Heads 3 and 7 to function correctly. Namely, a new head is composed with Heads 3 and 7's QK circuit and Head 1's OV circuit. Figure 1 (d) shows self excitation (Heads 3 and 6) and inhibition (Head 4). It is useful when a head is considered beneficial/detrimental in a particular context. Here no cross-head interaction occurs and it is more often called gating. Note that the $H \times H$ composition map applies to attention vectors $A_{:ij}$ between all pairs of Q_i and K_j uniformly, just like a 1x1 convolution with input and output channel dim of H (Figure 1 (e)).

Now that we know what we can do with attention matrix composition, we could in fact compose the $W^{Q/K/V/O}$ projections of MHA to achieve the same effects. Theorem 2.1 (proof in Appendix B) shows that composing attention score matrices is equivalent to composing H new query

[1]The OV circuit of head i is defined as $W_i^V W_i^O$, responsible for transforming the attended tokens before adding it to the residual stream; The QK circuit as $W_i^Q W_i^{K\top}$, responsible for forming the attention distribution. Terminology from Elhage et al. (2020).

[2]We indeed systematically observed such cases in our preliminary mechanistic interpretability study. See Table 11 in Appendix F for concrete examples.

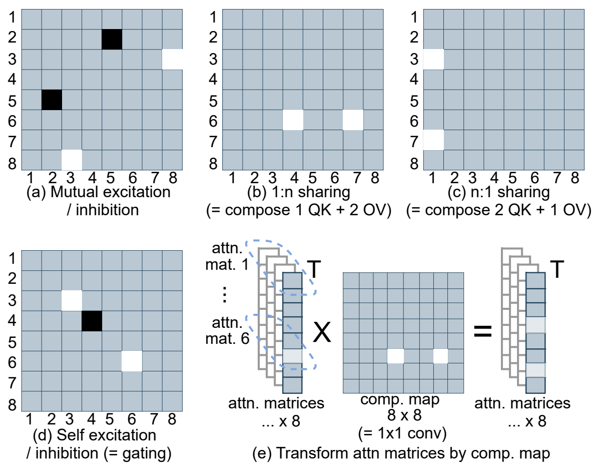

(a) Mutual excitation / inhibition
(b) 1:n sharing (= compose 1 QK + 2 OV)
(c) n:1 sharing (= compose 2 QK + 1 OV)
(d) Self excitation / inhibition (= gating)
attn. mat. 1
attn. mat. 6
T
X
=
attn. matrices ... x 8
comp. map 8 x 8 (= 1x1 conv)
attn. matrices ... x 8
(e) Transform attn matrices by comp. map

Figure 1. Simplified and prototypical composition maps for 8 heads and their functions. Lighter color denotes larger value.

and key projections $\{\tilde{W}_i^Q, \tilde{W}_i^K \in \mathbb{R}^{D_m \times HD_h}\}_{i=1}^H$ with H times larger head dim by concatenating exiting projections $\{W_i^Q, W_i^K \in \mathbb{R}^{D_m \times D_h}\}_{i=1}^H$ as follows (D_m is model dim. D_h is head dim. Assume no bias is used for the projections):

$$\tilde{W}_i^Q = \underset{j\in[H]}{\text{Concat}}\left[C_{ij}W_j^Q\right],\ \tilde{W}_i^K = \underset{j\in[H]}{\text{Concat}}\left[W_j^K\right] \quad (1)$$

Theorem 2.1. *Composition of attention scores $\{A_i\}_{i=1}^H$ by composition map $C \in \mathbb{R}^{H\times H}$ is equivalent to QK projection composition with H-fold expansion defined in Eqn. (1).*

Similarly, we have a theorem for the equivalence between attention weight matrix composition and the following OV projection composition:

$$\tilde{W}_i^V = \underset{j\in[H]}{\text{Concat}}\left[C_{ij}W_j^V\right],\ \tilde{W}^O = \text{Tile}(W^O, (H,1)) \quad (2)$$

where $\tilde{W}_i^V \in \mathbb{R}^{D_m\times HD_h}, \tilde{W}^O \in \mathbb{R}^{HHD_h\times D_m}$ are composed and expanded projection matrices. Tile$(W^O,(H,1))$ tile W^O along its first dim H times.

Theorem 2.2. *Composition of attention weights $\{A_i\}_{i=1}^H$ by composition map $C \in \mathbb{R}^{H\times H}$ is equivalent to OV projection composition with H-fold expansion defined in Eqn. (2).*

Attention matrix composition's relationship with expansion-based projection composition also supports its effectiveness: Bhojanapalli et al. (2020) has shown that enlarging head dim of QK projections mitigates the low-rank bottleneck of attention score matrices. We posit that enlarging head dim of OV projections can increase the cross-token information transmission bandwidth of heads. Therefore, both attention score and attention weight compositions can fundamentally improve model expressiveness.

It is worth noting that the analysis above is based on the assumption that the attention matrix composition is *static*, i.e. a shared composition map C (a trainable parameter which can be seen as a 1x1 convolution kernel) is applied to all $T \times S$ attention vectors $\{A_{:ij}\}$. When the composition becomes *dynamic* for additional expressive power gain, i.e. there is a map C for each pair of query and key with their attention vector (namely a local convolution with input dependent kernels) it has no equivalent projection composition. This reveals that attention matrix composition is more general and flexible than projection composition.

3. Dynamically Composable Multi-Head Attention

In MHA, the attention vector $A_{:ij}$ governs the information flow between query Q_i and key K_j. At the core of DCMHA is a `Compose` function which, given Q_i and K_j, transforms their attention vector $A_{:ij} \in \mathbb{R}^H$ into a new vector $A'_{:ij}$ with trainable parameter θ:

$$A'_{:ij} = \text{Compose}(A_{:ij}, Q_i, K_j; \theta) \quad (3)$$

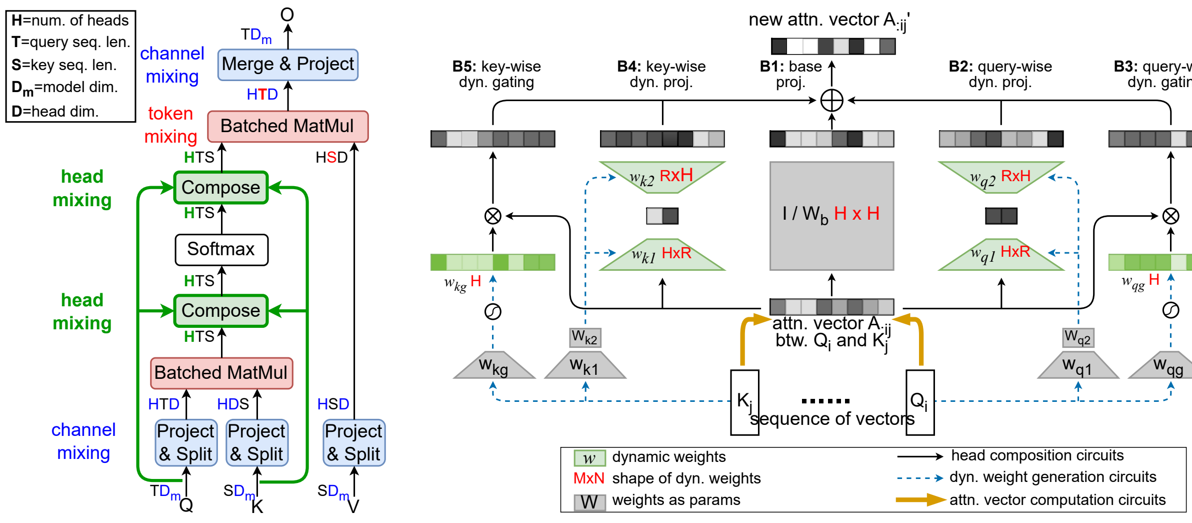
At a high level, to get DCMHA we just insert into the computation of MHA two `Compose` functions, where input-dependent cross-head interaction happens, one applied to the attention score tensor A^S before softmax, the other applied to the attention weight tensor A^W after softmax (Figure 2 (a)):

$$\begin{aligned} A_i^S &= \frac{QW_i^Q(KW_i^K)^T}{\sqrt{D_h}};\ A^S = \text{Stack}(A_1^S, ..., A_H^S) \\ A^S &= \text{Compose}(A^S, Q, K; \theta_{pre}) \\ A^W &= \text{Softmax}(A^S, \text{dim}=-1) \\ A^W &= \text{Compose}(A^W, Q, K; \theta_{post}) \\ O_i &= A_i^W(VW_i^V);\ O = \text{Concat}(O_1, ..., O_H)W^O \end{aligned} \quad (4)$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{D_m\times D_h}$ are projection matrices of the ith head, $W^O \in \mathbb{R}^{HD_h\times D_m}$ is the output projection matrix. We `Stack` tensors along the first dim and `Concat` them along the last dim. Here we use a "batched version" of `Compose` in Eqn. (3) which, given T queries and S keys, both packed into matrices as $Q \in \mathbb{R}^{T\times D_m}$ and $K \in \mathbb{R}^{S\times D_m}$, transforms their attention tensor $A \in \mathbb{R}^{H\times T\times S}$ to a new tensor of the same shape.

Now we describe the computation inside `Compose` (Figure 2 (b)). $A_{:ij}$ is transformed respectively through five branches before summing up together. $A_{:ij}$ is first projected by a weight matrix W_b independent of Q_i or K_j. This can be seen as a base composition on which dynamic compositions are superimposed. In the second branch, $A_{:ij}$ is first projected down by $\mathcal{W}_{q1} \in \mathbb{R}^{H\times R}$ to a lower dim R, then projected up by $\mathcal{W}_{q2} \in \mathbb{R}^{R\times H}$ to the original dim H to get $A_{:ij}\mathcal{W}_{q1}\mathcal{W}_{q2}$. The dynamic weights $\mathcal{W}_{q1}$ and $\mathcal{W}_{q2}$ are computed from Q_i.[3] This models how heads share their

[3]In this paper, we use calligraphic font for dynamic weights.



(a) Overall Architecture of DCMHA (b) Computation of Compose

Figure 2. Illustration of DCMHA. (a) Scale and optional mask operations are omitted. Each linear projection's input and output are denoted by their dims and the projected (i.e. mixed) dims are colored. (b) Attention vector $A_{:ij}$ can be either attention scores or weights.

attention scores/weights with each other. By letting $R \ll H$ (in this work we set $R = 2$), we assume that, despite many possible ways in which heads can share with each other, a small number of sharing patterns suffice for *any particular* pair of query and key. In the third branch, $A_{:ij}$ is element-wise multiplied by a gate $w_{qg} \in \mathbb{R}^H$ which is also computed from Q_i. This branch controls how much each head retains or forgets its original score given the query.

To compute the dynamic projection weights w_{q1} and w_{q2} from Q_i, we use an FFN with a single hidden layer and GELU activation function, parameterized by $W_{q1} \in \mathbb{R}^{D_m \times I}$ and $W_{q2} \in \mathbb{R}^{I \times I}$, where $I = 2HR$. We apply RMSNorm without scaling to w_{q1} along the number-of-head dim before multiplying it with $A_{:ij}$ to stabilize training:

$$\begin{aligned} w_{q1}, w_{q2} &= \text{Chunk}(\text{GELU}(Q_i W_{q1}) W_{q2}, \text{dim} = 1) \\ w_{q1} &= \text{Rmsnorm}(\text{Reshape}(w_{q1}, (H, R)), \text{dim} = 0) \quad (5) \\ w_{q2} &= \text{Reshape}(w_{q2}, (R, H)) \end{aligned}$$

To compute the dynamic gating weight w_{qg} from Q_i, we simply use a linear projection parameterized by $W_{qg} \in \mathbb{R}^{D_m \times H}$, followed by a tanh nonlinearity:

$$w_{qg} = \tanh(Q_i W_{qg}) \quad (6)$$

There are also two symmetric branches for K_j following the same computation as Q_i. The outputs of the five branches are summed up to obtain the final updated vector:

$$\begin{aligned} A'_{:ij} = A_{:ij} W_b + & A_{:ij} w_{q1} w_{q2} + A_{:ij} \otimes w_{qg} \\ & + A_{:ij} w_{k1} w_{k2} + A_{:ij} \otimes w_{kg} \end{aligned} \quad (7)$$

The trainable parameters for DCMHA are $\theta = \{W_b, W_{q1}, W_{q2}, W_{qg}, W_{k1}, W_{k2}, W_{kg}\}$. They are learned end-to-end together with the other parameters of the model.

3.1. A Tensor Decomposition Perspective

To do dynamic head composition, we need $T \times S$ transformation matrices (i.e. composition maps) of shape $H \times H$ for each pair of Q_i and K_j. In other words, we need to compute an input-dependent 4-D transformation tensor $W \in \mathbb{R}^{T \times S \times H \times H}$ and apply it to the 3-D attention tensor $A \in \mathbb{R}^{H \times T \times S}$. While there are theoretically many ways to do this, different approaches may lead to vast difference in efficiency. The computation of `Compose` described above is equivalent to applying a two-level decomposition of W for parameter and computation efficiency:

$$A'_{:ij} = A_{:ij} W_{ij} \quad i \in [1, T], j \in [1, S]$$

$$\begin{aligned} \overset{T \times S \times H \times H}{W} &= \overset{H \times H}{W_b} + \underbrace{\overset{T \times 1 \times H \times H}{\mathcal{W}_q}}_{row-wise} + \underbrace{\overset{1 \times S \times H \times H}{\mathcal{W}_k}}_{column-wise} \\ &= \overset{H \times H}{W_b} + \text{ED}(\overset{T \times H \times R}{\mathcal{W}_{q1}} \overset{T \times R \times H}{\mathcal{W}_{q2}} + \overset{T \times H \times 1}{\mathcal{W}_{qg}} \otimes \overset{H \times H}{\mathbb{1}}, 1) \\ & \quad + \text{ED}(\underbrace{\overset{S \times H \times R}{\mathcal{W}_{k1}} \overset{S \times R \times H}{\mathcal{W}_{k2}}}_{low-rank} + \underbrace{\overset{S \times H \times 1}{\mathcal{W}_{kg}} \otimes \overset{H \times H}{\mathbb{1}}}_{diagonal}, 0) \end{aligned} \quad (8)$$

where the function $\text{ED}(\cdot, \text{dim})$ stands for ExpandDims. Eqn. (8) can be seen as a "batched version" of (7). First, W is decomposed as the sum of a 2-D tensor $W_b \in \mathbb{R}^{H \times H}$, which is a static weight (as parameters) independent of input and plays the role of bias, and two 3-D tensors $\mathcal{W}_q \in \mathbb{R}^{T \times H \times H}$ and $\mathcal{W}_k \in \mathbb{R}^{S \times H \times H}$, which are dynamic weights dependent on queries (row-wise) and keys (column-wise) respectively.[4] We call this *row plus column decomposition*. Then each 3-D tensor is further decomposed as the sum of a low-rank tensor computed as the product of two tensors

[4]Size-1 dims are added for broadcasting, i.e. repeating along the missing dim, for point-wise summation / multiplication.

$\mathcal{W}_{q/k1} \in \mathbb{R}^{T/S \times H \times R}$ and $\mathcal{W}_{q/k2} \in \mathbb{R}^{T/S \times R \times H}$, and a diagonal tensor filled by a 2-D tensor $\mathcal{W}_{q/kg} \in \mathbb{R}^{T/S \times H}$. This form of decomposition is also used elsewhere (Zhao et al., 2016; Gu et al., 2021a) and is called *low-rank plus diagonal decomposition*. The row plus column decomposition allows applying $\mathcal{W}_q$ and $\mathcal{W}_k$ separately on the attention tensor A without materializing the large 4-D tensor W. The low-rank plus diagonal decomposition reduces the size of the transformation matrix on attention vectors from H^2 to $2HR + H$. Thanks to these decompositions, the resulting tensors are all much smaller than W and can be efficiently computed from input queries and keys. For example, $\mathcal{W}_{qg} = \tanh(QW_{qg})$ (a batched version of Eqn. (6)).

We empirically found that with the presence of the four dynamic branches, the static base projection (Branch 1 in Figure 2) can be replaced by a simple skip connection (i.e. directly adding $A_{:ij}$ to the final sum) with little degradation in performance (see Section 4.6). We do so in practice for a simpler and more efficient design. Complete pseudo-code for DCMHA is given in Appendix D.

3.2. Grouped Composition for Tensor Parallel Training

In Megatron-style tensor parallel (TP) training, attention heads of a layer are partitioned into groups, and each group of heads are placed on a node where the computation of these heads are performed. In a typical TP setting (e.g. H = 32, TP = 4), instead of composing across all 32 heads, one could compose heads within each group of 8 heads (The dynamic projection rank R for each group of heads could be set to a smaller value of 1). As composition takes place locally within each node, there is no cross-group interaction between heads and thus no additional cross-node communication introduced by DCMHA. This grouped composition can be implemented by simple modifications to `Compose`. We implemented grouped DCMHA and tested it with TP training. Empirically, we found that there is little difference between grouped composition and all-heads composition on performance as well as speed.

3.3. Complexity Analysis

Table 2 shows the ratios of extra parameters and computation introduced by DCMHA with both analytical results and typical concrete values.[5]The derivations are in Appendix E. The ratio of extra parameters, i.e. parameter count of θ_{pre} and θ_{post} of DCMHA divided by that of the whole model, is inverse proportional to the head dim D_h, and is negligible for commonly used D_h values, e.g. 128. The ratio of extra computation, i.e. FLOPs of the two compose functions divided by FLOPs of the whole forward pass, besides inverse proportional to D_h, increases with $\rho = S/D_m$, where S is sequence length, and is also very small for large enough models (e.g. $\geq$ 6.9B) with $D_m \geq 4096$ and typical values of S, e.g. $2048 \leq S \leq 8192$.

[5]The analysis assumes that we use all-heads composition. Grouped composition has lower complexity.

Table 2. Ratios of extra parameters and computation introduced by DCMHA: (last row) analytical results and (upper rows) concrete values for typical model architectures and hyperparamters. L = number of layers, S = sequence length.

Model Size	$\mathrm{R}_{\Delta\mathrm{params}}$	$\mathrm{R}_{\Delta\mathrm{FLOPs}}$	R	L	H	$\mathrm{D_h}$	S	$\rho = \frac{\mathrm{S}}{\mathrm{D_m}}$
1.4B	2.6%	4.8%	2	24	32	64	2048	1
6.9B	1.3%	1.9%	2	24	32	128	2048	0.5
		2.4%					4096	1
		3.3%					8192	2
Formula	$\frac{2R+1}{3D_h}$	$\frac{4(2R+1)(1+\rho)}{(12+\rho)D_h}$						

4. Experiments

Implementation Details We implement DCFormer model and training in JAX. Like several other works (Brown et al., 2020; Beltagy et al., 2020; Black et al., 2021), we use local attention with a sliding attention window of 256 tokens for each other layer of DCMHA. This improves efficiency without degrading results. We use normal initializers with standard deviations of $0.02/(\sqrt{2HR}(H + R))$ and $0.05\sqrt{2/(D_m + H)}$ for the dynamic projection/gating generating parameters W_{q2}, W_{k2}, and W_{qg}, W_{kg} respectively. This ensures that the dynamic projection and gating weights have small enough values at the beginning of training, which is found to be critical for successful training. For the other parameters of DCMHA, we use Xavier normal initializer.

Organization As language modeling is arguably the most important task for today's foundation models, we mainly focus on evaluating DCFormer on autoregressive language modeling with a decoder-only architecture, on both scaling curves of pretraining metrics (loss and perplexity) (Section 4.1) and downstream task evaluation with large scale training (Section 4.2). We use the Pile dataset (Gao et al., 2020) for all our language modeling experiments. Then to gain a better understanding of why and how DCFormer works, we evaluate the trained model on a synthetic dataset that partly motivated the design of DCMHA and analyze its projection matrices in Section 4.3. Then Section 4.4 quantifies DCFormer's training and inference overhead incurred by the compose operations. To verify if the advantages of DCMHA hold across different transformer architectures and modalities, in Section 4.5 we evaluate the performance of DCFormer when applied to encoder-only vision transformers for ImageNet-1K classification. Finally, Section 4.6 ablates various components of DCMHA. Unless otherwise specified, DCFormer and the counterpart Transformer always use the same experimental settings.

4.1. Scaling Laws

Settings Table 3 specifies the model sizes and hyperparameters for scaling experiments. The model architectures, learning rates and batch sizes are mostly taken from GPT-3 specifications (Brown et al., 2020). Unlike GPT-3, we untie input and output embedding matrices. We train with context length 2048 and set the number of training tokens to roughly match Chinchilla scaling laws (Hoffmann et al., 2022), which specifies that training tokens should increase proportionally to model size. The other hyperparameters are in Appendix C. We apply DCMHA to two Transformer architectures: the original architecture used by GPT-3 (Transformer) and an improved architecture used by LLaMA (Touvron et al., 2023) with rotary positional encodings (RoPE) (Su et al., 2024) and SwiGLU MLP (Shazeer, 2020) (Transformer++), the strongest transformer architecture we know.

Table 3. Model sizes and hyperparameters for scaling experiments.

params	n_{layers}	d_{model}	n_{heads}	learning rate	batch size (in tokens)	tokens
405M	24	1024	16	3e-4	0.5M	7B
834M	24	1536	24	2.5e-4	0.5M	15B
1.4B	24	2048	32	2e-4	0.5M	26B

Results Figure 3 (top) plots Pile validation loss scaling curves of Transformer(++) and DCFormer(++) models. DCMHA improves both Transformer and Transformer++ significantly on models from 405M to 1.4B. For example, by fitting a straight line for Transformer and Transformer++ data points, it can be estimated that DCFormer-834M matches the loss of Transformer with 1.87× compute, while DCFormer++-834M matches the loss of Transformer++ with 1.67× compute. Figure 3 (bottom) shows that, compared with the RoPE and SwiGLU MLP combination from Transformer++, DCMHA's magnitude of relative improvement (Δloss) over baseline models decreases less as the compute scales, showing the favorable scaling property of DCMHA as an architectural improvement.

4.2. Large Scaling Training and Downstream Evaluations

Settings We compare DCFormer with the well-known Pythia model suit (Biderman et al., 2023) at large scale training (300B tokens on Pile). Specifically, we train two models, DCPythia-2.8B and DCPythia-6.9B, and compare them with Pythia-2.8B, 6.9B and 12B. Except replacing MHA with DCMHA and adding QKNorm (Dehghani et al., 2023) to stablize training, DCPythia uses exactly the same architecture choices (e.g. parallel attention and MLP, rotary embedding with 1/4 head dim) and training hyperparamters (e.g. optimizer settings, learning rate schedule, batch size, context length, initialization methods) as Pythia (refer Biderman et al. (2023) Appendix E for details). In making

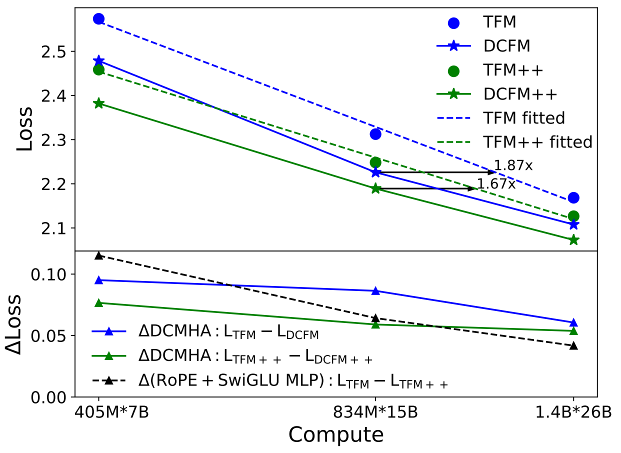


Figure 3. (top) Scaling curves of Transformers and DCFormers. (bottom) Scaling curves of relative improvement of RoPE + SwiGLU MLP and DCMHA. TFM++ = TFM + RoPE + SwiGLU MLP; DCFM = TFM + DCMHA; DCFM++ = TFM++ + DCMHA.

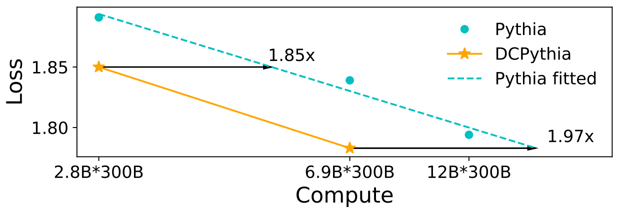


Figure 4. Scaling curves of Pythia and DCPythia.

these restrictions, our aim is not to obtain SoTA results, but to clearly quantify the gain brought by DCMHA.

Evaluation Datasets Besides the datasets used by Pythia for downstream evaluation (LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), ARC easy and challenge (Clark et al., 2018), SciQ (Welbl et al., 2017), LogiQA (Liu et al., 2020)), we also include BoolQ (Clark et al., 2019) and HellaSwag (Zellers et al., 2019) for commonsense reasoning, RACE (Lai et al., 2017) for reading comprehension, all of which are widely used benchmarks. We evaluate zero-shot and five-shot learning using LM evaluation harness (Gao et al., 2023).

Results Besides lower Pile validation loss and perplexity as shown in Figure 4 and Table 4, DCPythia also significantly outperforms Pythia at 2.8B and 6.9B scales on downstream task accuracies. Notably, DCPythia-6.9B outperforms Pythia-12B on both ppl and downstream task evaluation. Table 4 also reports perplexities on a randomly sampled subset of the Flan Collection dataset (Longpre et al., 2023). We sample 320K examples and compute loss on the target span. This dataset features data of instructing following, in-context few-shot learning, chain-of-thought, etc. The ppl differences between DCPythia and Pythia on Flan are significantly larger than on Pile, showing that DCMHA has more advantage in improving these valued emergent abilities of large language models (Wei et al., 2022).

Table 4. Zero-shot and five-shot evaluations on downstream NLP tasks.

Model	Pile ppl↓ /Δ ppl	Flan ppl↓ /Δ ppl	Lambada	PIQA	Wino Grande	ARC -E	ARC -C	SciQ	Logi QA	BoolQ	Hella Swag	RACE -M	RACE -H	Avg acc↑ /Δacc
			0-shot											
Pythia-2.8B	6.63	8.16	64.7	73.9	59.4	**64.4**	**29.5**	88.2	21.2	64.5	45.4	38.1	34.9	53.1
DCPythia-2.8B	**6.36** /-0.27	**7.68** /-0.48	**67.9**	**74.8**	**61.4**	64.3	28.8	**89.7**	**22.3**	**65.4**	**46.5**	**42.0**	**36.5**	**54.5** /+1.4
DCFM++2.8B	6.11	7.24	69.7	73.9	63.6	68.6	32.4	89.9	24.9	69.6	48.9	41.4	37.3	56.4
Pythia-6.9B	6.29	7.85	67.3	75.2	60.9	67.3	31.3	89.7	**25.3**	63.7	48.0	40.6	37.0	55.1
DCPythia-6.9B	**5.95** /-0.34	**7.13** /-0.72	**70.7**	**76.1**	**62.7**	**68.5**	**32.6**	**92.0**	24.0	**67.0**	**49.5**	**42.8**	**37.9**	**56.7** /+1.6
Pythia-12B	6.01	7.26	70.5	76.0	63.9	70.2	31.8	90.2	22.4	67.4	50.3	40.6	38.3	56.5
			5-shot											
Pythia-2.8B	-	-	60.5	73.6	60.6	67.3	32.3	94.3	21.7	65.6	45.1	38.4	35.6	54.1
DCPythia-2.8B	-	-	**64.6**	**74.3**	**64.2**	**69.3**	**33.3**	**95.1**	**22.4**	**68.0**	**46.5**	**42.0**	**37.1**	**56.1**
DCFM++2.8B	-	-	66.2	74.8	65.2	70.5	36.0	96.0	23.4	68.7	49.2	43.1	38.6	57.4
Pythia-6.9B	-	-	63.8	75.5	**63.7**	70.2	**35.6**	95.1	27.0	65.7	48.1	39.0	36.5	56.4
DCPythia-6.9B	-	-	**65.6**	**76.5**	63.1	**70.4**	34.3	**95.9**	**27.8**	**69.1**	**49.8**	**43.4**	**38.9**	**57.7**
Pythia-12B	-	-	67.3	76.0	64.2	71.0	36.5	95.3	21.8	68.0	50.3	40.1	38.8	57.2

It can also be observed that on both perplexities and downstream evaluation, the relative improvements brought by DCMHA (ΔPile ppl, ΔFlan ppl, Δavg acc) with 6.9B model is generally larger than those with 2.8B model, again showing that DCMHA scales well. To show that DCMHA also works well with Transformer++ at 2.8B scale, we also train a 2.8B DCFormer++ model DCFM++2.8B, which get significantly better results than DCPythia-2.8B.

4.3. Synthetic Tasks and Weight Analysis of Trained Models

To stress the dynamic head composition ability of models we construct a synthetic dataset consisting of a set of tasks. To successfully predict the answer (the last word) for an example in some task, e.g. "*John has an apple. Mary has a dog. So John has a kind of __*", the model must simultaneously accomplish two things: attending to the right source token '*apple*' by looking up for the same person, and applying the right transformation to the token embedding (object−>superclass) when moving it to the residual stream of the destination token '*of*'. By making various combinations of attention patterns (e.g. same-person, the-other, different-in-set) and transformations (e.g. obj−>superclass, city−>country), we construct a total of 74 tasks and 888 examples. See Appendix F for more examples in the dataset. We conjecture that MHA-based models, where heads have fixed QK (for attending the source token) and OV (for transforming the token) circuits, would have difficulty in solving this kind of tasks.

We test Pythia-6.9B and DCPythia-6.9B on the synthetic dataset and report the results in Table 5.[6] We find that: 1) this dataset, though looks simple, is challenging for both models; 2) the advantage of DCPythia-6.9B over Pythia-6.9B on this dataset is much greater than on Pile and Flan. Presumably, the gain comes from DCMHA's ability to dynamically combine the QK and OV circuits of existing heads according to a given task. Appendix G gives mechanistic interpretability analysis and visualization on how DCPythia-6.9B solves an example in this dataset.

[6]The results are averaged over a mixture of K-shot examples where $2 \leq K \leq 7$. For more results see Figure 6 in Appendix F.

Table 5. Evaluation results on the synthetic dataset.

Model	PPL	Acc
Pythia-6.9B	10.05	31.9%
DCPythia-6.9B	7.36	39.0%

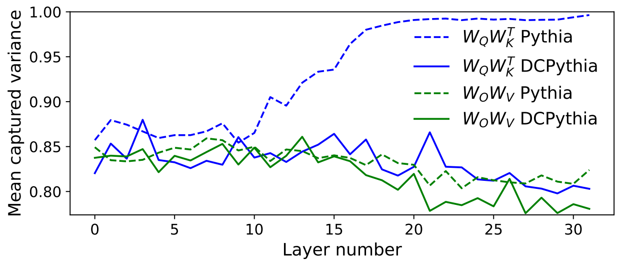


Figure 5. Mean cumulative captured variance of concatenated QK and OV heads of Pythia-6.9B and DCPythia-6.9B.

Head Diversity In Figure 5, We measure head diversity (Cordonnier et al., 2020) of Pythia-6.9B and DCPythia-6.9B by concatenating $W_QW_K^T$ and W_OW_V matrices layer-wisely, and calculating mean cumulative captured variance by their principle components. A lower value indicates higher diversity of the QK and OV circuits of attention heads in a layer. Figure 5 demonstrates DCMHA enhances QK circuit diversity significantly and improves OV circuit diversity moderately. The gain of head diversity is also a supportive evidence that DCMHA reduces head redundancy and enhances model expressiveness.

Due to the large difference in head projection statistics between MHA and DCMHA, it would be difficult to adapt a pretrained Transformer model to DCFormer by continual pretraining. We did not obtain significant improvement by finetuning 1/10 steps on a 1.4B model with Llama architecture, which was pre-trained by us from scratch on the C4 dataset (Raffel et al., 2020). Through integrated gradients attribution analysis, we observed that head composition of lower layers are more important than upper layers. On the other hand, the gradient of lower layers when finetuning the pretrained model is relatively small, hindering substantial update of lower MHA layers in the pretrained model, which is consistent with the previous observations (e.g. Houlsby et al. (2019)). The difficulty of adaptation to existing models also shows the fundamental difference between DCFormer and Transformer.

4.4. Training and Inference Overhead

As DCMHA introduces additional operations in the compose function, we quantify its overhead compared with MHA in real training and inference settings.

Models Though we use Pythia's architecture in large scale training, the evaluation in this section is done on untrained models with Transformer++/DCFormer++ architecture, which has more practical value due to better performance. We measure on 4 model sizes: The 2.8B and 6.9B models are "LLaMAed" version of Pythia 2.8B and 6.9B; The 13B and 33B models use the same architecture as LLaMA (Touvron et al. (2023), Table 2).

Settings We train on TPU v3 pods with context length of 2048, batch size of 2M tokens and measure DCFormer's throughput numbers and percentages as compared with Transformer++'s throughput. The 2.8B and 6.9B models are trained on 256 TPU v3 chips while the 13B and 33B models are trained on 512 chips. For inference we use A100 80G GPU and set the prompt length and generation length to 1024 and 128 respectively. We use a batch size of 1 and measure the speed to generate 128 tokens. We repeat the measurements 3 times and take the average. As mentioned previously, all DCFormer++ models use a local attention window of 256 for every other layer. Besides comparing it with standard Transformer++, we also compare with Transformer++ with the same attention window and report the ratio as the second percentage after slash.[7] To speedup inference, we cache the results of key-wise computation of `Compose` along with the KVCache. This is made feasible by the row plus column decomposition (Section 3.1) because query-wise and key-wise computation are independent. Besides, we use `torch.compile` to accelerate both Transformer++ and DCFormer++.

Table 6. Training throughput and inference speed comparison between Transformer++ and DCFormer++.

Model	Training (K tokens/s)		Inference (tokens/s)	
Size	TFM++	DCFM++	TFM++/win	DCFM++
2.8B	396	295 ×**74.5%**	151/164	133 ×**87.9%**/81.1%
6.9B	201	167 ×**83.1%**	83.2/88.7	78.6 ×**94.5%**/88.7%
13B	203	171 ×**84.4%**	45.5/48.1	43.3 ×**95.1%**/89.9%
33B	84	75 ×**89.2%**	19.9/21.0	18.9 ×**94.8%**/89.7%

Results As shown in Table 6, The training overheads are generally larger than the inference overheads, and both decrease as the model scales. These overheads, though larger than the theoretical estimates in Table 2 and not negligible, are acceptable considering the performance gain, especially at larger scales. The overheads are mainly due to the I/O bottleneck caused by the series of operations on the attention matrices introduced by `Compose`, which are I/O bound instead of compute bound. Currently we implement training in pure JAX and inference in pure PyTorch without writing any custom kernels. We believe there is room for acceleration using FlashAttention-like tiling and kernel fusion techniques (Dao et al., 2022) and leave it for future work.

4.5. Image Classification

Besides decoder-only transformer or language modeling, we apply DCMHA to Vision Transformer (ViT, an encoder-only transformer) (Dosovitskiy et al., 2020) for image classification on the Imagined-1k dataset (ILSVRC-2012). Implementation and experimental settings are based on the Big Vision code base[8]. We use ViT-S/16 as the baseline model and equip it with DCMHA to obtain DCViT-S/16. We also compare with a 1.7x larger model ViT-M/16 (Table 7). We report top-1 and top-5 accuracy results in Table 8. DCViT-S/16 outperforms ViT-S/16, on par with ViT-M/16 (though the accuracy differences at Epoch 300 between the three models are relatively small).

Table 7. ViT Model architectures for ImageNet-1k classification.

Model	n_{layers}	d_{model}	d_{mlp}	n_{heads}	params
(DC)ViT-S/16	12	384	1536	6	22M
ViT-M/16	12	512	2048	8	39M

4.6. Ablation Studies and Tradeoffs

We ablate and compare various components of DCMHA, focusing on the settings of scaling law experiments for lan-

[7]Note that all the Transformer(++) models used as baselines in this paper, including Pythia models, are trained using global attention *without* window.

[8]https://github.com/google-research/big_vision. Detailed hyperparamters for training are in Appendix C.

Table 8. ViT for ImageNet-1k classification results.

Model	Epoch 90 Top-1	Epoch 90 Top-5	Epoch 300 Top-1	Epoch 300 Top-5	Relative size
ViT-S/16	65.2	86.8	79.8	94.7	1
DCViT-S/16	**68.0**	**88.6**	80.1	**95.0**	1.03
ViT-M/16	67.1	87.9	**80.3**	94.9	1.72

Table 9. Ablations of DCMHA's components. a = Talking-Heads Attention (Shazeer et al., 2020); b = all $-$ a = dyn. proj. + gate

Config	ppl	Config	ppl	Config	ppl
TFM++	11.68				
+static proj.[a]	11.17	+query-wise	10.89	R=1	10.87
+dyn. proj.	10.95	+key-wise	10.91	R=2	10.83
+dyn. gate	11.31	+pre comp.	11.54	R=4	10.89
+all	10.79	+post comp.	11.05		
DCFM++[b]	10.83				

Table 10. Performance and speed trade-offs for different speedup configs and models.(QW: Query-Wise, *: default config, ^ : query-wise config in Table 9)

Local:Global Attn.	1:1*	3:1	7:1	1:1 QW^	3:1 QW
Pile Validation ppl					
DCFM++ 405M	10.83	10.78	10.83	10.89	10.92
DCPythia-6.9B (1/10 steps)	8.17	8.00	8.04	7.98	8.03
Training(K tokens/s)-6.9B					
DCFM++	167	174	177	184	186
pct. vs TFM++	83.1%	86.6%	88.1%	91.5%	92.5%
Inference(tokens/s)-6.9B					
DCFM++	78.6	82.39	84.46	82.89	85.97
pct. vs TFM++/ w/ window	94.5%/ 88.7%	99.0%/ 89.6%	101.5%/ 90.3%	99.6%/ 93.2%	103.3%/ 93.5%

guage modeling with Transformer++/DCFormer++ 405M models in Section 4.1 (see Table 3). We add each (groups of) component(s) separately to Transformer++ to study its effect and report the perplexity results in Table 9.

Dynamic vs Static While static composition (static proj., Branch 1 in Figure 2 (b), also equivalent to Talking-Heads Attention (Shazeer et al., 2020)) is effective, the dynamic composition used by DCFormer++ (dyn. proj. + gate) improves much more, getting very close to +all Config, showing the critical role of dynamicity in increasing expressive power. Among dynamic composition components, low-rank projection (Branch 2 and 4) is more effective than gating (Branch 3 and 5), showing the importance of cross-head sharing.

Query-wise vs Key-wise When acting alone, both query-wise (Branch 2 and 3) and key-wise (Branch 4 and 5) compositions work surprisingly well, showing that query-wise and key-wise branches can work independently with little interaction between them, and that there may be some overlaps in their functions.

Pre-compose vs Post-compose When acting alone, post-compose on attention weights is significantly more effective than pre-compose on attention scores, presumably because attention weights have a more direct impact on the final output of DCMHA module. This also reveals the shortages of existing works that only consider attention score composition (Wang et al., 2022; Nguyen et al., 2022; Cordonnier et al., 2020).

Impact of ranks There is slight performance gain when increasing the dynamic project rank R from 1 to 2, but further increasing the rank has no positive effect, validating the choice of $R = 2$ in our work.

Tradeoffs We explore two performance-efficiency tradeoffs that can further improve the efficiency of DCMHA: 1) increasing the ratio of local:global attention layers and 2) only using query-wise composition. We train two models across scales (DCFormer++ 405M and DCPythia-6.9B) with different configs to quantify their impact on performance by measuring Pile validation ppl as shown in Table 10. For DCPythia-6.9B, we train only 13K steps to save compute cost. We use Transformer++/DCFormer++ 6.9B in Table 6 to study the impact on training and inference efficiency. For inference speed we compare DCFormer++6.9B with two Transformer++6.9B baselines: one with all global attn and one with the same local:global attn ratio as DCFormer++. It can be observed from the table that increasing the local:global attn ratio from 1:1 to 7:1 improves training and inference efficiency without hurting performance. Only using query-wise composition also improves efficiency while slight degrading performance. The two approaches can also be combined, offering a spectrum of trade-offs. Specifically, combining 3:1 local:global attn with query-wise composition increases DCFormer++ 6.9B's training throughput ratio from 83.1% to 92.5%, increases inference speed ratio from 94.5%/88.7% to 103.3%/93.5%, while the ppl is slightly worse than the default DCFormer but still significantly better than the Transformer baseline.

5. Conclusion

We introduce a dynamic head composition mechanism to improve the MHA module of Transformers. Experimental results show that DCFormer is effective, efficient and scalable, significantly outperforming strong Transformer baselines, especially on the important language modeling task for foundation models. In the future, we would like to apply the idea of dynamic head composition to more architectures and domains, and to do more interpretability studies on DCMHA to gain a deeper understanding of its working mechanism.

Acknowledgements

We are grateful to Google Cloud for providing the compute for model training, and to Shun Wang for his technical support and help in troubleshooting TPU resource allocation and training.

Impact Statement

This paper presents work on improving Transformer architecture by dynamically composing multi-head attention, which can be used to boost performance of large language models with slight overhead. It can help save pretraining cost and reduce carbon footprint in the era of LLMs when adopted in practice. In addition, our open source code and models can advance research of architecture innovation and promote downstream application of large language models. However, it is also possible that models are used maliciously to generate harmful contents, which may have negative societal impact.

References

Ahmed, K., Keskar, N. S., and Socher, R. Weighted transformer network for machine translation. *arXiv preprint arXiv:1711.02132*, 2017.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Bhojanapalli, S., Yun, C., Rawat, A. S., Reddi, S., and Kumar, S. Low-rank bottleneck in multi-head attention models. In *International conference on machine learning*, pp. 864–873. PMLR, 2020.

Bhojanapalli, S., Chakrabarti, A., Jain, H., Kumar, S., Lukasik, M., and Veit, A. Eigen analysis of self-attention and its reconstruction from partial computation. *arXiv preprint arXiv:2106.08823*, 2021.

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Black, S., Gao, L., Wang, P., Leahy, C., and Biderman, S. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. URL `https://doi.org/10.5281/zenodo.5297715`. If you use this software, please cite it using these metadata.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Cordonnier, J.-B., Loukas, A., and Jaggi, M. Multi-head attention: Collaborate instead of concatenate. *arXiv preprint arXiv:2006.16362*, 2020.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dar, G., Geva, M., Gupta, A., and Berant, J. Analyzing transformers in embedding space. *arXiv preprint arXiv:2209.02535*, 2022.

Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A. P., Caron, M., Geirhos, R., Alabdulmohsin, I., et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pp. 7480–7512. PMLR, 2023.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Du, Z., Qian, Y., Liu, X., Ding, M., Qiu, J., Yang, Z., and Tang, J. Glm: General language model pretraining with autoregressive blank infilling. *arXiv preprint arXiv:2103.10360*, 2021.

Elhage, N., Neel, N., Olsson, C., et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2020. URL https://transformer-circuits.pub/2021/framework/index.html.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021a.

Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., and Ré, C. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34: 572–585, 2021b.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., et al. An empirical analysis of compute-optimal large language model training. *Advances in Neural Information Processing Systems*, 35: 30016–30030, 2022.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

Hua, W., Dai, Z., Liu, H., and Le, Q. Transformer quality in linear time. In *International Conference on Machine Learning*, pp. 9099–9117. PMLR, 2022.

Kasai, J., Peng, H., Zhang, Y., Yogatama, D., Ilharco, G., Pappas, N., Mao, Y., Chen, W., and Smith, N. A. Finetuning pretrained transformers into rnns. *arXiv preprint arXiv:2103.13076*, 2021.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Li, J., Yang, B., Dou, Z.-Y., Wang, X., Lyu, M. R., and Tu, Z. Information aggregation for multi-head attention with routing-by-agreement. *arXiv preprint arXiv:1904.03100*, 2019.

Liu, J., Cui, L., Liu, H., Huang, D., Wang, Y., and Zhang, Y. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*, 2020.

Liu, X., Su, J., and Huang, F. Tuformer: Data-driven design of transformers for improved generalization or efficiency. In *The Tenth International Conference on Learning Representations (ICLR 2022)*, 2022.

Longpre, S., Hou, L., Vu, T., Webson, A., Chung, H. W., Tay, Y., Zhou, D., Le, Q. V., Zoph, B., Wei, J., et al. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*, 2023.

Michel, P., Levy, O., and Neubig, G. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

Narang, S., Chung, H. W., Tay, Y., Fedus, W., Fevry, T., Matena, M., Malkan, K., Fiedel, N., Shazeer, N., Lan, Z., et al. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

Nguyen, T., Nguyen, T., Do, H., Nguyen, K., Saragadam, V., Pham, M., Nguyen, K. D., Ho, N., and Osher, S. Improving transformer with an admixture of attention heads. *Advances in neural information processing systems*, 35: 27937–27952, 2022.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., and Kong, L. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

Qin, Z., Han, X., Sun, W., Li, D., Kong, L., Barnes, N., and Zhong, Y. The devil in linear transformer. *arXiv preprint arXiv:2210.10340*, 2022.

Qin, Z., Li, D., Sun, W., Sun, W., Shen, X., Han, X., Wei, Y., Lv, B., Yuan, F., Luo, X., et al. Scaling transnormer to 175 billion parameters. *arXiv preprint arXiv:2307.14995*, 2023.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Shazeer, N., Lan, Z., Cheng, Y., Ding, N., and Hou, L. Talking-heads attention. *arXiv preprint arXiv:2003.02436*, 2020.

Smith, J. T., Warrington, A., and Linderman, S. W. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. Retentive network: A successor to transformer for large language models. *URL http://arxiv.org/abs/2307.08621 v1*, 2023.

Sundararajan, M., Taly, A., and Yan, Q. Axiomatic attribution for deep networks. In *International conference on machine learning*, pp. 3319–3328. PMLR, 2017.

Tay, Y., Dehghani, M., Tran, V. Q., Garcia, X., Bahri, D., Schuster, T., Zheng, H. S., Houlsby, N., and Metzler, D. Unifying language learning paradigms. *arXiv preprint arXiv:2205.05131*, 2022.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

Wang, H., Shen, X., Tu, M., Zhuang, Y., and Liu, Z. Improved transformer with multi-head dense collaboration. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:2754–2767, 2022.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

Welbl, J., Liu, N. F., and Gardner, M. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, X., Shen, Y., Huang, Z., Zhou, J., Rong, W., and Xiong, Z. Mixture of attention heads: Selecting attention heads per token. *arXiv preprint arXiv:2210.05144*, 2022.

Zhao, Y., Li, J., and Gong, Y. Low-rank plus diagonal adaptation for deep neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5005–5009. IEEE, 2016.

A. Related work

We overview some prior works related to our DCMHA in the following subsections.

A.1. Architecture Modifications to Transformers

Since being introduced seven years ago, many modifications to the Transformer architecture have been proposed. However, relatively few of them generalize well across domains and scales and have seen widespread adoption (Narang et al., 2021) Some notable successful ones include Transformer-XL (Dai et al., 2019) and Rotary Position Encoding (Su et al., 2024) for improving long-context handling and position encoding, GLU MLP (Shazeer, 2020) and Sparse Mixture-of-Experts (MoE) MLP (Lepikhin et al., 2020; Fedus et al., 2022) for more expressive or efficient MLP nonlinearty and architecture, UL2 (Tay et al., 2022) and GLM (Du et al., 2021) for better training objectives. Among these, RoPE and SwiGLU MLP have been adopted by recent well-known foundation models such as Palm (Chowdhery et al., 2023) and LLaMA (Touvron et al., 2023), and are also used as our strong baseline (Transformer++).

A.2. Improving MHA by Head Collaboration

Noticing the problems caused by the independent working of attention heads, various forms of cross-head collaboration or interaction mechanisms have been proposed (Li et al., 2019; Zhang et al., 2022; Cordonnier et al., 2020; Liu et al., 2022; Shazeer et al., 2020; Wang et al., 2022; Nguyen et al., 2022). While some of these works mainly focus on improving parameter or computation efficiency of MHA by reducing head redundancy (Cordonnier et al., 2020; Nguyen et al., 2022; Zhang et al., 2022), we aim to improve model performance. Sharing the same goal as ours, Wang et al. (2022) proposed a Multi-Head Dense Collaboration (MHDC) mechanism and evaluate it primarily on Neural Machine Translation and some other small NLP tasks. MHDC is essentially the same as the static projection of attention scores in pre-compose of DCMHA, although they enhance it with cross-layer collaboration. We propose a more comprehensive head composition framework which supports dynamic composition of both attention scores and weights with pre- and post-compose, evaluate on large scale language model pretraining as well as downstream tasks.

The work most closely related to ours is Talking-Heads Attention (THA) (Shazeer et al., 2020), which proposed to use two learned cross-head projections before and after softmax to transform the attention score and attention weight tensor respectively, which is same as pre- and post-compose with only static projections in DCMHA. They showed the effectiveness of THA in T5-style pretraining and downstream evaluation. We more clearly motivate head composition by relating it to projection composition, propose dynamic composition to further increase model expressiveness significantly, and offer a parameter and computation efficient design and implementation based on two-level tensor decomposition. The authors of THA also proposed a dynamic variant of THA in Appendix A of the paper, but compared with ours, the parameter and computation overhead is too large for practical use (see Table 8 in Appendix A of Shazeer et al. (2020)).

A.3. Linear Attention Transformers

As another line of work, linear attention Transformers (Katharopoulos et al., 2020; Choromanski et al., 2020; Kasai et al., 2021; Peng et al., 2021; Qin et al., 2022) address MHA's quadratic complexity with respect to context length T and slow $O(T)$ inference by using various kernels to replace the softmax function and re-arranging the order of matrix multiplications. The models can be switched to recurrent mode for efficient $O(1)$ inference but sacrificing training parallelism. Besides, the modeling capability and performance are usually worse than Transformers, which hinders their popularity. Recently, several improved linear transformer variants (Sun et al., 2023; Qin et al., 2023) have reported performance matching or exceeding Transformer (but not Transformer++). Inspired by Hua et al. (2022) and similar to the query-wise dynamic gating in post-compose of DCMHA, they also add gating on the attention outputs to improve performance, though operating at a finer granularity of neurons instead of heads. It is interesting to explore if DCMHA can be combined with these linear attention transformer variants.

A.4. Non-Transformer Architectures

In recent years, several non-attention architectures with subquadratic complexity have been proposed, mainly based on RNNs (Peng et al., 2023) or State Space Models (SSMs) (Gu et al., 2021b;a; Smith et al., 2022; Gu & Dao, 2023), in an attempt to replace Transformer as the backbone for foundation models. Most recently, Gu & Dao (2023) proposed an improved SSM variant Mamba which can match the performance of Transformer++. Coincidentally, they also use an input-dependent selection mechanism as the source of improvement in model expressiveness. Our work shows that Transformer can also be improved by an input-dependent head composition mechanism. While works such as RWKV and Mamba advocate *replacing* Transformer with completely different architectures, we take the approach of *enhancing* Transformer, including the most advanced Transformer++, by improving its core MHA module. Our work shows that the Transformer, as a powerful and mature architecture, still has a lot of potential for improvement, especially in its attention mechanism.

B. Proofs of Two Theorems on Equivalency

Proof of Theorem 2.1

Let $\{\tilde{A}_i\}_{i=1}^H$ be the attention score matrices computed with the composed and expanded projections $\{\tilde{W}_i^Q, \tilde{W}_i^K\}_{i=1}^H$.

$$\begin{aligned}
\tilde{A}_i &= Q\tilde{W}_i^Q(K\tilde{W}_i^K)^\top \\
&= (Q[C_{i1}W_1^Q \quad | \dots | \quad C_{iH}W_H^Q \quad])\cdot \\
&\quad (K[\underbrace{\qquad W_1^K}_{local\ dot-product} \quad | \dots | \quad \underbrace{\qquad W_H^K}_{local\ dot-product} \quad])^\top \\
&= QC_{i1}W_1^Q(KW_1^K)^\top + \dots + QC_{iH}W_H^Q(KW_H^K)^\top \\
&= C_{i1}A_1 \qquad\qquad + \dots + C_{iH}A_H \\
&= A_i' \quad \square
\end{aligned}$$

where "|" is the concatenate operator. The proof is straightforward, based on the observation that concatenate-then-dot-product and dot-product-then-sum lead to the same result, so the dot-product between the expanded (concatenated) query and key can be decomposed into the sum of H local dot-products between original queries and keys.

Proof of Theorem 2.2

Following the reformulation of MHA by Elhage et al. (2020), we split $\tilde{W}^O$ along the first dim into H parts such that each head has its own $W_i^O \in \mathbb{R}^{D_h \times D_m}$, which is expanded to the same $\tilde{W}_i^O = W^O \in \mathbb{R}^{HD_hD_m}$ (There is a better symmetry with QK projection expansion with this reformulation). Then the attention output computed with the composed and expanded projections $\{\tilde{W}_i^V, \tilde{W}_i^O\}_{i=1}^H$ are $\sum_{i=1}^H A_i(V\tilde{W}_i^V)\tilde{W}_i^O$. We show that it equals to the attention output computed with original projections and composed attention weight matrices $\{A_i'\}_{i=1}^H$.

$$\begin{aligned}
\sum_{i=1}^H A_i(V\tilde{W}_i^V)\tilde{W}_i^O &= \sum_{i=1}^H A_i(V\operatorname*{Concat}_{j\in[H]}\big[C_{ij}W_j^V\big]\operatorname*{Concat}_{j\in[H]}\big[W_j^O\big]) \\
&= \sum_{i=1}^H A_i(V\operatorname*{Concat}_{j\in[H]}\big[C_{ij}W_j^VW_j^O\big]) \\
&= \sum_{i=1}^H A_i\sum_{j=1}^H C_{ij}VW_j^VW_j^O \\
&= \sum_{j=1}^H(\sum_{i=1}^H C_{ij}A_i)VW_j^VW_j^O \\
&= \sum_{j=1}^H A_j'(VW_j^V)W_j^O \quad \square
\end{aligned} \tag{9}$$

C. Hyperparameters for Experiments

Hyperparamters for language model scaling experiments We use the AdamW optimizer with β_1 = 0.9, β_2 = 0.95, gradient clip value of 1.0, weight decay of 0.1, 1% learning rate warmup steps followed by cosine decay to 10% of its maximal value, and no dropout.

Hyperparamters for ImageNet-1k classification experiments We use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.9$, gradient clip value of 1.0, weight decay of 0.0001, learning rate of 0.001 with 10000 warmup steps followed by cosine decay to 10% of its maximal value, and no dropout.

D. Pseudo-code for DCMHA

```
# B = batch size; S = key/value len; T = query len
# D_m = model dim; H = num. of heads; D = head dim; R = rank

def dw_proj(
    X,    #  B * T * D_m
    W_1, #  D_m * (H*R*2)
    W_2  #  (H*R*2) * (H*R*2)
    ):
    dw = gelu(X @ W_1) @ W_2
    dw1, dw2 = dw.chunk(2, dim=-1)
    dw1 = rmsnorm(rearrange(dw1, 'BT(RH)->BTRH'), dim=-1)
    dw2 = rearrange(dw2, 'BT(RH)->BTRH')
    return dw1, dw2

def compose(
    a, # B * H * T * S
    Q, # B * T * D_m
    K, # B * S * D_m
    theta
    ):
    W_q1, W_q2, W_k1, W_k2 = theta.W_q1, theta.W_q2, theta.W_k1, theta.W_k2
    W_qg, W_kg = theta.W_qg, theta.W_kg # D_m * H

    dw1, dw2 = dw_proj(Q, W_q1, W_q2)
    h = einsum('BHTS,BTRH->BRTS', a, dw1)
    o_qp = einsum('BRTS,BTRH->BHTS', h, dw2)

    dw1, dw2 = dw_proj(K, W_k1, W_k2)
    h = einsum('BHTS,BSRH->BRTS', a, dw1)
    o_kp = einsum('BRTS,BSRH->BHTS', h, dw2)

    o_qg = einsum('BHTS,BTH->BHTS', a, tanh(Q @ W_qg))
    o_kg = einsum('BHTS,BSH->BHTS', a, tanh(K @ W_kg))
    return a + o_qp + o_kp + o_qg + o_kg

def DCMHA(
    Q, K, V, W_q, W_k, W_v, W_o, causal,
    theta_lc, # params for pre-composition
    theta_pc  # params for post-composition
    ):
    q, k, v = [rearrange(x, 'BT(HD)->BHTD') for x in
               [Q @ W_q, K @ W_k, V @ W_v]]
    logits = einsum('BHTD,BHSD->BHTS', q, k)
    logits = compose(logits, Q, K, theta_lc)
    if causal: logits = causal_mask(logits)
    probs = logits.softmax(-1)
    probs = compose(probs, Q, K, theta_pc)
    o = einsum('BHTS,BHSD->BHTD', probs, v)
    return rearrange(o, 'BHTD->BT(HD)') @ W_o
```

E. Details of Complexity Analysis

To simplify, we calculate $R_{\Delta params}$ and $R_{\Delta FLOPs}$, the ratios of extra parameters and computation introduced by DCMHA in one transformer layer by ignoring the batch and layer dimension. We only consider the self-attention case where $T = S$. We assume $D_h \gg 2R$.

Ratio of extra parameters

$$\begin{aligned} R_{\Delta params} &= \frac{\overbrace{D_m 4(2RH)}^{W_{q1/k1}} + \overbrace{4(2RH)^2}^{W_{q2/k2}} + \overbrace{D_m 4H}^{W_{qg/kg}}}{12D_m^2} \\ &= \frac{4(2HR)}{12HD_h} + \frac{16R^2}{12D_h^2} + \frac{4H}{12HD_h} \\ &= \frac{2R}{3D_h} + \frac{4R^2}{3D_h^2} + \frac{1}{3D_h} \\ &\approx \frac{2R+1}{3D_h} \quad \text{(ignore 2nd term assuming } D_h \gg 2R) \end{aligned} \tag{10}$$

Ratio of extra FLOPs.

$$\begin{aligned} R_{\Delta FLOPs} &= \frac{\overbrace{2(T+S)(D_m(2RH)+2RH)^2)}^{\substack{generate\ w_{q1/q2}, w_{k1/k2} \\ \text{Code Line 9}}} + \overbrace{2(T+S)D_m H + 4TSH}^{\substack{generate\ w_{qg}, w_{kg}\ and\ apply \\ \text{Code Line 32-33}}} + \overbrace{8TSHR}^{\substack{apply\ w_{q1/q2}, w_{k1/k2} \\ \text{Code Line 25-26,29-30}}}}{D_m T(12D_m + S)} \\ &= \frac{2(T+S)(2D_h RH^2 + 4R^2H^2 + D_h H^2) + 4TSH(2R+1)}{HD_h T(12D_m + S)} \\ &= \frac{4H((2R+1)D_h + 4R^2)}{D_h(12D_m + S)} + \frac{4(2R+1)S}{D_h(12D_m+S)} \quad \left(T = S, \rho = \frac{S}{D_m}\right) \\ &\approx \frac{4(2R+1)}{D_h(12+\rho)} + \frac{4(2R+1)\rho}{D_h(12+\rho)} \quad \text{(assume } D_h \gg 2R) \\ &= \frac{4(2R+1)(\rho+1)}{(12+\rho)D_h} \end{aligned} \tag{11}$$

F. Examples of Synthetic Dataset and Few shot Evaluation Results

Table 11 shows more one-shot examples in the synthetic dataset.

Table 11. One-shot examples from the synthetic dataset. Source tokens for predicting the right answer is in bold.

QK pattern	OV transformation	Question	Answer
same-person	obj→superclass	< Ruth has a violin. Carol has a beetle. Ronald has **blueberries**. >. Ronald has a kind of fruit < Deborah has a **shirt**. Daniel has strawberries. Edward has a grenade. >. Deborah has a kind of ____	clothing
the-other-person	copy	< Ronald has a grenade. Ronald has a costume. Maria has a **lime**. >. Ronald does **not** own lime < Kenneth has T-shirt. Kenneth has pizza. Nancy has a **bee**. >. Kenneth does **not** own ____	bee
same-obj	copy	< **Jeff** has shoes. Laura has a baseball. Joseph has pizza. >. Who possesses shoes? Jeff < **Michelle** has a jersey. Helen has an elephant. Joseph has a pineapple. >. Who possesses jersey? ____	Michelle
same-person	city→country	< Madrid attracts Kenneth. **Mumbai** attracts John. Marseille attracts Susan. >. John yearns for the city in India < **Vancouver** attracts Steven. Paris attracts Deborah. London attracts Sarah. >. Steven yearns for the city in ____	Canada
same-person	obj→capability	< Sarah has a car. Steven has spray. Mark has swim **fins**. >. Mark possesses the thing that can swim < Mark has a violin. Christopher has spray. Barbara has **chalk**. >. Barbara possesses the thing that can ____	write
different-adj	adj→opposite	There are **big**, unhealthy, ill. Which is different? The opposite of small There are heavy, **dishonest**, weighty. Which is different? The opposite of ____	honest
different-ordinal	ordinal→prev	There are 2017, **2011**, 2017. Which is different? The year just after 2010 There are 2012, 2017, 2017. Which is different? The year just after ____	2011
the-other-class	copy	There are a sweater, an **elephant**, a shirt. Which is not clothing? The elephant There are a pear, a banana, a **car**. Which is not fruit? The ____	car

Figure 6 shows the evaluation results on the synthetic dataset with differenct number of shots.

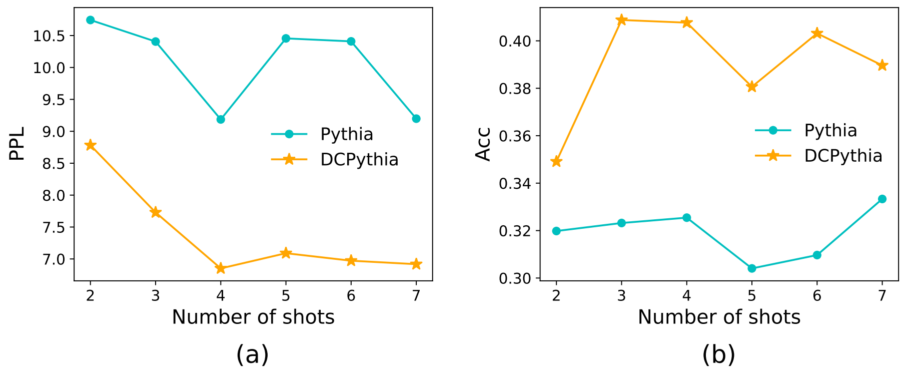


Figure 6. Few shot evaluation of Pythia-6.9B and DCPythia-6.9B on the synthetic dataset.

G. Interpretation and Visualization

We try to unveil the working mechanism of DCMHA by analyzing how DCPythia-6.9B tackles the (same-person, object−>superclass) task in our synthetic dataset. One example is "*Michelle has a* ***burger****. Helen has a papaya. Carol has a cannon. Michelle owns a kind of* ***food***", the last word of which need to be predicted. To solve it, the model need attend to the source **object** token '*burger*' among three candidates ('*burger*', '*papaya*' and '*cannon*') from the destination token '*of*' and transform it to its **superclass** '*food*', which requires for an attention head whose QK circuit attends the right candidate and OV circuit completes the transformation. The interpretation case study goes as follows.

1. **OV Circuit Identification**: Using Integrated Gradients attribution (Sundararajan et al., 2017) on all attention heads' outputs, we can locate head L16-H11 (Layer 16, Head 11) whose output is the most important for making the right prediction. Further weight analysis (Dar et al., 2022) of L16-H11 also confirms that its OV circuit can accomplish transformation from object to superclass (see G.1).

2. **Breakdown of Post-Composition**: Although the QK circuit of L16-H11 can sometimes attend to the source token, when failed, it can also reuse attention weights of the other heads via composition. Figure 7 displays the attention weights $A_{:,of,burger}$, $A'_{:,of,burger}$ of all heads in Layer 16 before and after post-composition (the leftmost and rightmost vector respectively). It is clear that the attention weight of L16-H11 from '*of*' to '*burger*' increases significantly after composition. We further break the attention weight into five parts: $A_{:,of,burger}$, O_{qp}, O_{qg}, O_{kp} and O_{kg}, which corresponds to the five terms in Eqn. 7. The breakdown indicates that O_{kp} contributes most, which is the output of key-wise low-rank projection by $\mathcal{W}_{k1}$ and $\mathcal{W}_{k2}$.

3. **Locating Helpful QK Circuit**: The left part of Figure 7 visualizes the forward pass of the key-wise projection: $\mathcal{W}_{k1,of,burger}$ gathers attention weights from all heads (mainly made up of L16-H2 and L16-H17) into the inner low-rank components and $\mathcal{W}_{k2,of,burger}$ shares them to several heads including L16-H11. As a result, the projection composes the QK circuits of L16-H2, L16-H17 with the OV circuit of L16-H11 (recall Figure 1 (c)).

4. **Enhancing QK Circuit by Dynamic Composition**: The reason why L16-H11 should combine the QK circuits of L16-H2 and L16-H17 lies in the attention distributions demonstrated in Figure 8 (bottom). These two heads can attend to the source token '*burger*' correctly. So with help of them, L16-H11 is able to revise its most attended token from '*cannon*' to '*burger*', effectively enhancing its own QK circuit. Figure 8 (top) shows variation of the second component (Column 1) of $\mathcal{W}_{k1,of}$ along the sequence (for clear visualization, we sample several representative rows as opposed to Figure 7 which shows the whole matrices.). One interesting observation is that $\mathcal{W}_{k1,of,1,17}$ and $\mathcal{W}_{k1,of,1,2}$ is generally larger at positions of nouns ('*burger*', '*Michelle*') than that of other tokens ('*kind*', '*of*').

5. **The Role of Query-wise Gating**: As found in G.1, the OV circuit of L16-H2 performs copying. While its attention weight from '*of*' to '*burger*' is also enhanced by the key-wise projection, it is decreased by O_{qp}, the output of the query-wise gating branch (the purple boxes in Figure 7). This makes sense because the query '*of*', as the token after '*a kind*', is in a better position of knowing that the appropriate OV transformation is object−>superclass instead of copying, so it inhibits L16-H2 by giving a negative value in $\mathcal{W}_{qg,of}$.

G.1. OV circuit Analysis

We analyse functions of the OV circuits of L16-H11, L16-H2 and L16-H17 by using projection weight analysis. Specifically, given a head (Layer l, Head h) and a token x, we obtain the top 10 tokens converted by the OV circuit following Eqn.12. Instead of using the token embedding directly, we use the output of the first MLP layer as input to the OV circuit of the head, which gives better results. By inspecting the input and output tokens, we find that L16-H11 can transform an object to its superclass, while the other two heads fail. In addition, L16-H2 tends to copy input tokens.

$$\begin{aligned}
&x = \text{Embed}(x)\\
&x = x + \text{MLP}_0(\text{Layernorm}_0^{mlp}(x)) \qquad (12)\\
&logits = \text{Unembed}(\text{Layernorm}_l^{attn}(x)W_h^V W_h^O))
\end{aligned}$$

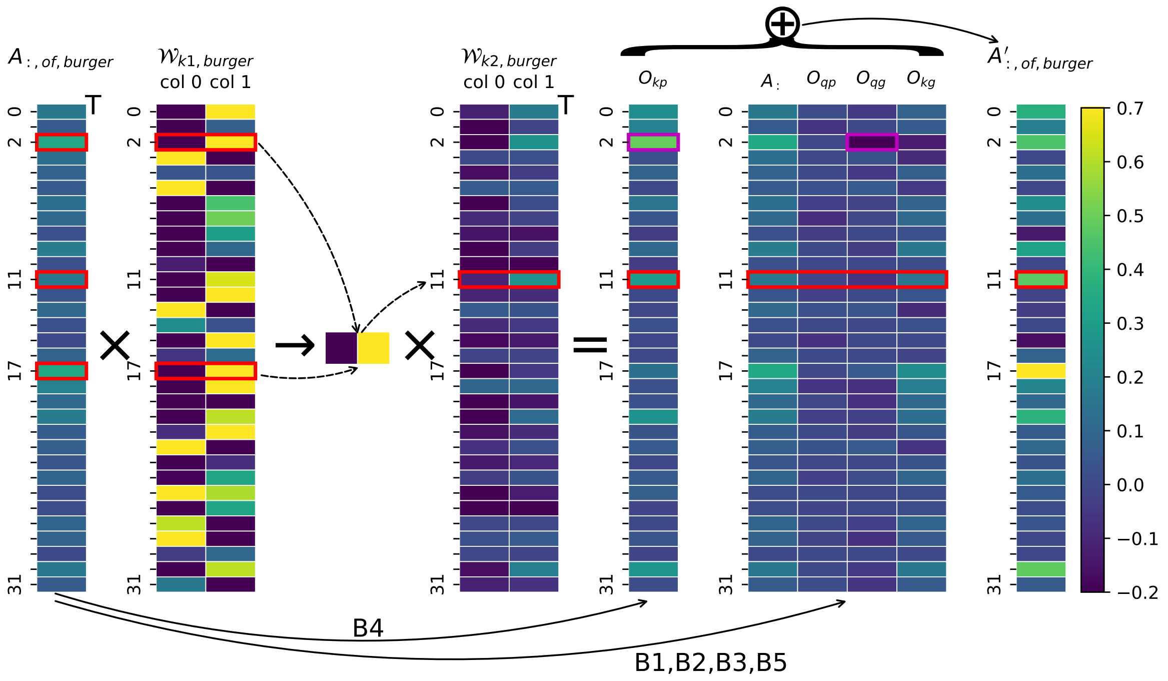


Figure 7. Composition of attention weights by post-compose in a case study. Seen as an instantiation of Figure 2 (b) where the processing of Branch 4 is depicted in detail.

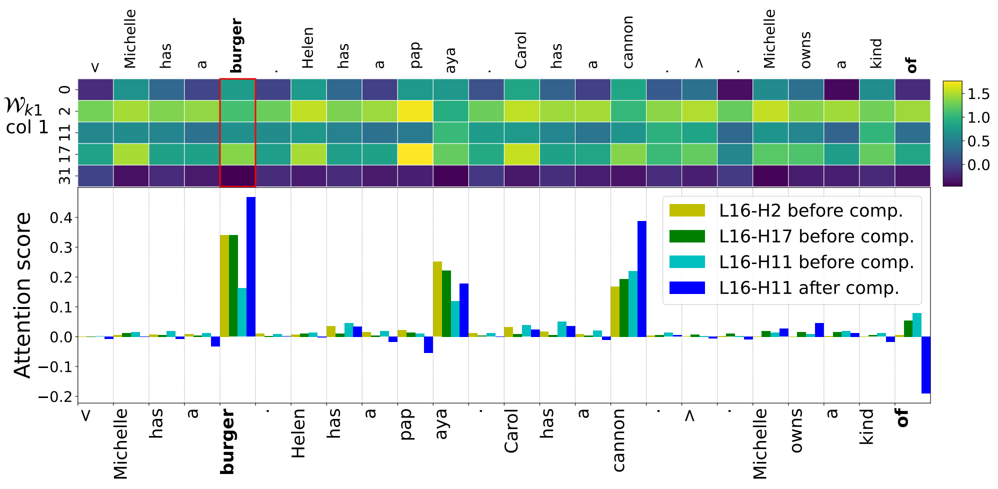


Figure 8. Attention distributions and sampled $\mathcal{W}_{k1}$ visualization in the case study.