Multi-Token Attention

Olga Golovneva **Tianlu Wang** **Jason Weston** **Sainbayar Sukhbaatar**

FAIR at Meta

Abstract

Soft attention is a critical mechanism powering LLMs to locate relevant parts within a given context. However, individual attention weights are determined by the similarity of only a single query and key token vector. This "single token attention" bottlenecks the amount of information used in distinguishing a relevant part from the rest of the context. To address this issue, we propose a new attention method, *Multi-Token Attention* (MTA), which allows LLMs to condition their attention weights on multiple query and key vectors simultaneously. This is achieved by applying convolution operations over queries, keys and heads, allowing nearby queries and keys to affect each other's attention weights for more precise attention. As a result, our method can locate relevant context using richer, more nuanced information that can exceed a single vector's capacity. Through extensive evaluations, we demonstrate that MTA achieves enhanced performance on a range of popular benchmarks. Notably, it outperforms Transformer baseline models on standard language modeling tasks, and on tasks that require searching for information within long contexts, where our method's ability to leverage richer information proves particularly beneficial[1].

1 Introduction

The attention mechanism (Bahdanau et al., 2014; Vaswani et al., 2017) is a critical component of Large Language Models (LLMs) that enables them to retrieve and combine information from different parts of the context. Attention is especially useful when the context contains a large number of tokens, as focusing on the relevant part while disregarding distractions becomes crucial. However, numerous works have shown that standard attention can suffer from suboptimal performance in this setting (Kamradt, 2023; Kuratov et al., 2025).

Standard multi-head attention works by comparing the similarity of the current query vector and the key vectors corresponding to the context tokens using their dot products. The keys similar to the query obtain higher attention weights, and subsequently their value vectors dominate the output vector. For example, a query vector corresponding to a token "Alice" is capable of locating all mentions of "Alice" in the context. However, each attention weight conditions only on a single key and query vector (besides the normalization to sum to one).

We argue that the dependency on single token vector similarity brings a fundamental limitation to the attention mechanism. In many cases, the relevant part of the context cannot be identified by a single token. For example, looking up a sentence that mentions both "Alice" and "rabbit" requires the query vector to encode both tokens. Looking up "Alice" with one attention head and using another head for "rabbit" could find their mentions separately, but is not sufficient to pinpoint where both are mentioned together. While it is possible to encode multiple tokens into a single vector via the layers of the Transformer, this requires increased dimension, and the model to use lots of its capacity for this task.

In this paper, we propose a novel attention mechanism that goes beyond this "single token" bottleneck, which we call Multi-Token Attention (MTA). The high level goal is to make it possible to use the similarities of *multiple vector pairs* to determine where attention must

[1]Code is available at https://github.com/facebookresearch/RAM/tree/main/projects/mta

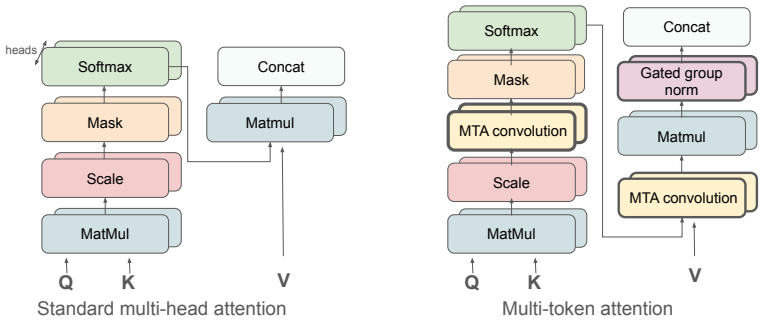

Figure 1: **Multi-Token Attention (MTA)** (right), compared to standard attention (left). In MTA, within each head we apply a key-query convolution on the attention scores and a head convolution across groups of heads. We repeat this operation after the softmax. Finally, we apply a group normalization with scalar gating before final concatenation.

focus. We achieve this by making straight-forward modifications to the existing attention mechanism. In particular, we design convolution operations over attention weights that operate on three dimensions: keys, queries, and attention heads. This allows its attention weights to condition on neighboring keys, previous queries, and other heads. Intuitively, following our previous example, MTA can find mentions of "Alice" and "rabbit" separately first, and then combine those attentions together to focus only on where both exist.

We first experiment with a motivating toy task that reveals the shortcoming of standard attention and demonstrate that MTA can easily solve it. Next, we test our method at scale by pre-training 880M parameters models on 105B tokens on a standard language modeling task. There we see MTA bring improvements in terms in validation perplexity as well as standard benchmark tasks, while only increasing the number of parameters by 0.001%. Further, we evaluate the resulting models on long-context tasks such as Needle-in-the-Haystack and BabiLong where MTA outperforms the baselines by a significant margin. Finally, we investigate scaling laws of the proposed model and compare them with baselines.

2 Background on multi-head attention

We first describe the standard multi-head attention (Vaswani et al., 2017) and define the notation we use. In decoder-only Transformer architectures, the model receives a sequence of tokens $[x_1, ..., x_T]$ of length T as an input. These tokens then undergo a series of transformations into hidden states $H = [h_1, \dots h_T]^\top \in \mathbb{R}^{T \times D}$ through an embedding layer and repeated transformer layers. Each layer consists of a multi-head attention submodule, a feedforward network, and normalization operations. Multi-head attention includes M heads with dimensions $d = D/M$ working in parallel. Each head uses key, value and query projections $W_k, W_v, W_q \in \mathbb{R}^{D \times d}$ to construct key, value and query vectors:

$$K = HW_k, \quad V = HW_v, \quad Q = HW_q$$

The attention logits $\hat{A}$ and weights A (i.e. probabilities) are then calculated as follows:

$$\hat{A} = QK^\top / \sqrt{d}, \quad A = \text{Softmax}(\text{Mask}_{-\infty}(\hat{A})), \tag{1}$$

where the softmax operates over the key dimension, and the mask function replaces values at (i, j) with $-\infty$ for $i < j$ to prevent information leaking from future tokens. Finally, the attention output $AV \in \mathbb{R}^{T \times d}$ from all heads is then concatenated, and multiplied by the output projection $W_o \in \mathbb{R}^{D \times D}$ which is then passed to the normalization and feedforward components. This standard approach is summarized in Figure 1 (left).

3 Multi-Token Attention

Each attention value in standard multi-head attention, see Equation 1, depends solely on a single key and query vector. That means all the necessary information for finding and

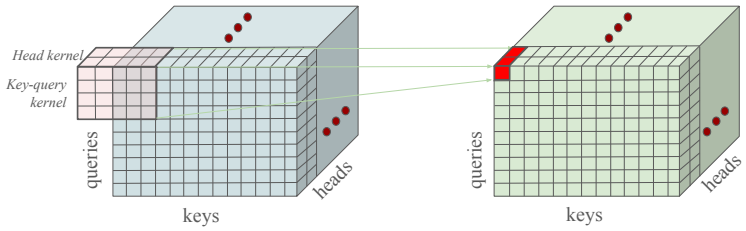

Figure 2: MTA applies key-query and head convolution over attention values.

attending to a relevant part of the context must be compressed into these single vectors. This might not be ideal if we are looking for a sentence containing multiple elements. Consider for example the sentence "Where did Alice see the rabbit?". We could try to find instances of "Alice" and "rabbit" independently and then check if there is a sentence that has both. Let q_a and q_r be query vectors encoding "Alice" and "rabbit" respectively (assuming a word tokenizer), then their attention weights are computed as follows:

$$a_a = \text{Softmax}(q_a K^\top / \sqrt{d}), \quad a_r = \text{Softmax}(q_r K^\top / \sqrt{d}) \tag{2}$$

By doing normal attention with those queries, we can attend where "Alice" and "rabbit" are mentioned in the context. All we have to do then is to check if both attention weights a_a and a_r have higher probabilities at the same nearby locations, e.g. in the same sentence, which will indicate that the sentence mentions both "Alice" and "rabbit". Unfortunately, normal attention lacks such interaction between attention maps, and instead only uses them to compute output values. Even if we use different attention heads to find "Alice" and "rabbit", there is no mechanism to combine these attention weights. This motivates us to modify the attention mechanism to allow combining different attention maps from nearby locations (both in terms of query and key locations), or between different attention heads.

As shown in Figure 1 (right), our proposed Multi-Token Attention consists of three important components built on top of multi-head attention: key-query convolution, head mixing convolution, and group normalization with gating mechanism. The overall MTA convolution applies the key-query convolution to combine multiple keys and queries within heads, and the head convolution to share knowledge between heads and amplify important information. Finally, we apply group normalization with scalar gating to push back against residual streams and improve gradient flow. In this section, we will describe each component of MTA in detail.

3.1 Key-query convolution

Pre-softmax convolution MTA applies a convolution operation on attention logits to combine information from multiple query and key tokens:

$$A = \text{Softmax}\left(\text{Conv2d}_\theta(\hat{A})\right), \tag{3}$$

where Conv2d_θ is a 2-dimensional convolution operation with kernel weights θ, and kernel sizes (c_q, c_k). Convolution is applied to the key and query length dimensions, while the batch and head dimensions remain independent. More precisely, the attention weight a_{ij} from query q_i to key k_j is computed as follows:

$$a_{ij} = \text{Softmax}\left(\sum_{i'=0}^{c_q-1} \sum_{j'=-\lfloor c_k/2 \rfloor}^{\lceil c_k/2 \rceil - 1} \mathbf{1}_{i \geq j - j'} \, \theta_{i',j'} \, q_{i-i'} \, k_{j-j'}^\top \, / \, \sqrt{d}\right) \tag{4}$$

Given that the query position is i, any information from position $> i$ should not be used to prevent information leakage from the future. That is why only past queries are used in Equation 4. For keys, we use indicator function $\mathbf{1}_{i \geq j - j'}$ to zero out future keys. However such masking is too complex to implement (it is necessary to modify the convolution cuda kernels), thus we propose a simpler version that applies the existing causal masking twice:

$$A = \text{Softmax}\left(\text{Mask}_{-\infty}\left(\text{Conv2d}_\theta\left(\text{Mask}_0(\hat{A})\right)\right)\right). \tag{5}$$

Here, the first masking uses 0 so that those values will not affect the output from the convolution. Although this version masks out a little more than necessary, it is simpler to implement and prevents information leakage, so we use it as our default.

Post-softmax convolution Similarly, we can apply the convolution on top of the attention weights instead of the logits

$$A = \text{Mask}_0\left(\text{Conv2d}_\theta\left(\text{Softmax}\left(\text{Mask}_{-\infty}(\hat{A})\right)\right)\right). \tag{6}$$

This makes interaction between attention weights additive instead of multiplicative. We will experiment with both versions, but will use the pre-softmax version by default.

Each attention head has separate θ parameters, so they can perform different convolution operations. The chosen kernel dimensions dictate how far away tokens can be combined together. In the above example, the question "Where did Alice see the rabbit?" will make q_a and q_r queries separated by two tokens (assuming a word tokenizer), so we need $c_q = 4$ to cover both queries. Similarly, the target sentence "Alice saw the white rabbit under the tree" for example produces keys k_a and k_r separated by three tokens, so $c_k = 5$ is sufficient for combining them. Before applying the convolution, we pad the input with an appropriate amount of zeros so that each convolution operation has a valid input.

3.2 Head mixing convolution

Unlike the key-query convolution, which allows mixing of attention weights from different time steps, we further propose to use a head convolution over groups of heads, so attention weights from different heads can be combined. In particular, for a head convolution kernel of size c_h, all heads are divided in M/c_h groups. Within each group, we apply a non-overlapping convolution operation[2]. This way, MTA allows not only to condition attention weights on multiple query and key vectors within each head, but also shares attention information across heads. For example, consider splitting all heads into groups of two, such that the kernel size is $c_h = 2$. Let us use superscript to denote head indices, so A^1 and A^2 are attention weights from two different heads. Then the new attention weights are:

$$A^1_{\text{new}} = w_{11}A^1 + w_{12}A^2, \quad A^2_{\text{new}} = w_{21}A^1 + w_{22}A^2, \tag{7}$$

where $w_{11}, w_{12}, w_{21}, w_{22}$ are kernel weights. Here the mixing occurred after the softmax, but we can also mix logits before the softmax:

$$\hat{A}^1_{\text{new}} = w_{11}\hat{A}^1 + w_{12}\hat{A}^2, \quad \hat{A}^2_{\text{new}} = w_{21}\hat{A}^1 + w_{22}\hat{A}^2,$$

Like the key-query convolution, we experiment with both versions.

3.3 Putting everything together

In the previous sections, we introduced two different ways of mixing attention weights, one across key-query time steps and another across different heads. These two can be implemented together in a single MTA module. Because each has pre and post-softmax versions, there are multiple different ways of combining them.

If both mixing methods are pre-softmax, then they can be implemented by a single 3-dim convolution operation as shown in Figure 2. Two of these dimensions will span key and query dimensions as described in Section 3.1. The third dimension will be across attention heads, but on groups of c_h heads. The same 3-dim convolution can be applied after softmax to perform both mixing methods post-softmax. The third possibility is to apply the key-query convolution before softmax, and then head mixing after softmax. This is also straightforward by applying Equation 7 on the attention weights obtained from Equation 5.

Finally, we follow Ye et al. (2024) who also apply normalization, but in our case we replace the layer-dependent scaling with a sigmoid gating mechanism. Sigmoid gating allows the model to switch heads on and off across layers and better adapt to different tasks. We also perform ablations across different normalization approaches.

[2]Since the kernel size c_h and group size c_h are the same, it can also be viewed as fully-connected.

	Block size $N = 5$			*Block size* $N = 8$		
Model	**All**	**First**	**Last**	**All**	**First**	**Last**
Transformer	51.6 ± 43.1	78.2 ± 1.5	1.3 ± 0.4	31.2 ± 50.3	28.9 ± 24.9	58.4 ± 46.8
MTA	0.1 ± 0.1	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 1: **Error rates (%) on the motivating toy task** where the model needs to locate a block of letters containing the given $L = 2$ letters. The output should contain *all*, *first*, or *last* tokens of the target block. MTA perfectly solves this task while a standard Transformer struggles to learn it. We report average error rate over 3 seeds and their standard deviations.

4 Experiments

We conduct experiments with the MTA architecture, comparing to several baselines on a set of standard and long-range dependency tasks, starting with a toy task. Unless otherwise specified, we apply the MTA convolution both pre-softmax and post-softmax.

4.1 Motivating toy task

We start with a simple toy task to demonstrate the effectiveness of our method over standard multi-head attention. In this task, the model is given a sequence of blocks where each block consists of N random letters. This is followed by L question letters ($L < N$). The objective is to find the block that contains all question letters in any order. Then the model must output *all* letters of the target block, or only its *first* or *last* token (as three separate task variants).

Despite its simplicity, this task poses a challenge to standard attention because it requires L pieces of information (i.e. question letters) to identify the target block. To succeed, standard soft attention must encode all L question letters into a single query vector. In contrast, MTA can first find the locations of each question letter, then use its convolution operation to increase the attention to the locations where all L letters are found together.

For this task, we train a small Decoder-only Transformer model with 4 layers, 2 heads and 256 hidden dimensions. The results are shown in Table 1. As expected, Transformer with standard multi-head attention struggles to solve this task, often completely failing to find the target blocks even though the questions have only $L = 2$ letters. This highlights the inherent limitation of standard attention conditioned on single token vectors. For MTA, we set the query kernel size $c_q = 2$ to match the number of question letters, and the key kernel $c_k = 2N - 1$ so it can cover a whole block on both sides. To simplify the experiments, the key-query convolution is only applied before softmax, and no head convolution is used. This way, it is sufficient for each query vector to encode only a single question letter while the convolution operation can aggregate their locations to find the target block. As a result, MTA successfully solves all the versions of the task with near zero error rate. See Appendix C for more training details.

4.2 Large language modeling

For language modeling experiments, we perform pre-training of 880M-size models and compare the Transformer model (Vaswani et al., 2017), Differential Transformer (DIFF Transformer) (Ye et al., 2024), Transformer with Talking heads attention (Shazeer et al., 2020), and Transformer with MTA. DIFF Transformer calculates attention scores as the difference between two separate softmax attention maps, while Talking heads attention applies linear projections across heads before and after softmax operations, thus being related to our head convolution approach, so we use them as baselines. All models are trained in the same setup on the SlimPajama (Soboleva et al., 2023) dataset for 105B tokens using the Lingua framework (Videau et al., 2024). Training details are provided in Appendix D. To improve training efficiency, we apply the key-query convolution on every 4th layer, while head convolution is applied on all layers. Kernel dimensions are fixed at $c_q = 6, c_k = 11$, and we mix groups of $c_h = 16$ heads within each layer. We ablate these parameters in Section 4.6.

For each model, we conduct training twice and report average validation perplexity in Table 2. We observe consistent improvements across all validation datasets for the model

Model	arxiv	book	c4	cc	github	se	wiki	Avg PPL ↓
Pretraining								
Transformer	4.65	13.47	20.20	14.41	4.28	10.13	11.64	11.25 (0.00)
DIFF transformer	4.62	13.33	19.99	14.28	4.25	10.04	11.54	11.15 (0.02)
Talking heads	4.59	13.26	19.82	14.15	4.20	9.93	11.33	11.04 (0.00)
MTA	**4.54**	**13.09**	**19.63**	**14.00**	**4.12**	**9.76**	**11.18**	**10.91 (0.01)**
Long context finetuning								
Transformer	4.32	13.18	20.14	14.08	3.96	9.84	11.63	11.02
DIFF transformer	4.28	13.01	19.87	13.93	3.90	9.72	11.49	10.89
Talking heads	4.29	13.25	19.95	13.90	3.86	9.66	11.29	10.88
MTA	**4.21**	**12.77**	**19.51**	**13.66**	**3.79**	**9.46**	**11.14**	**10.65**

Table 2: Validation perplexity for 880M Transformer model on SlimPajama dataset after training for 105B tokens, and finetuning with 2048 → 4096 context extension for another 10.5B tokens. Pretraining perplexity was averaged across two runs.

Model	BoolQ	PIQA	SIQA	HellaS	WinoG	ARCe	ARCc	OBQA	MMLU	Avg ↑
Transformer	56.2	70.2	39.9	38.5	56.4	57.9	**25.9**	**23.8**	24.5	43.7 (0.3)
DIFF transformer	59.6	70.5	39.7	38.9	56.4	57.7	25.6	21.4	24.9	43.9 (0.5)
Talking heads	61.9	71.4	**40.6**	39.4	54.5	58.2	25.2	23.6	24.7	44.4
MTA	**62.1**	**71.7**	40.4	**39.7**	**57.2**	**58.9**	24.7	23.6	**25.8**	**44.9**

Table 3: Pretrained models' evaluation results on standard benchmarks. Results are averaged over two model training runs for each method.

trained with MTA, even though the key-query convolution was only applied to a quarter of the layers, keeping the total number of parameters on-par with the Transformer baseline (see Appendix Table 8). We also note that group normalization with layer scaling is an important component that drives superior performance for both the DIFF Transformer and MTA architectures.

We further evaluate our models on a set of popular benchmarks in a zero-shot setup as shown in Table 3. The model trained with MTA outperforms the baselines on most of them and achieves a higher average score, despite these not being long-context tasks.

4.3 Long context finetuning

We further finetune our models on the same mix of datasets for another 10.5B tokens, but increase the context length from 2048 to 4096. We increase RoPE's theta to 500000, change the weight decay to 0, and reduce warm up steps to 50. Other parameters remain the same as during pretraining (see Appendix D). The resulting Transformer model with MTA similarly outperforms the new baselines in perplexity evaluations as shown in Table 2.

4.4 Long-range dependency tasks

It was previously shown that Transformers struggle to find relevant information especially in the middle of long context (Liu et al., 2024; 2025). To test MTA in this setting, we evaluate trained models on three tasks: LAMBADA (Paperno et al., 2016; Radford et al., 2019), Needle-In-A-Haystack (Kamradt, 2023) and BabiLong (Kuratov et al., 2025). All these tasks require models to almost sharply attend to the long-range tokens buried in the context.

LAMBADA is a collection of texts that test the model's ability to search long-range dependencies in the context. In particular, this dataset is designed such that humans can correctly predict the next word only if they have the whole text, but not if they only see the last sentence preceding the target word. We observe models trained with MTA are better at correctly guessing the next word (Table 4), significantly outperforming the baseline Transformer model.

Needle-In-A-Haystack We probe our models by inserting 2, 4, and 6 needles in 2k and 4k context windows at varying depths. We additionally ensure that for each new sample, the

Model	LAMBADA standard ↓	LAMBADA OpenAI ↓
Transformer	17.6	9.5
DIFF transformer	14.9	9.3
Talking heads	15.1	8.9
MTA	**13.2**	**8.4**

Table 4: Perplexity evaluations on the LAMBADA standard (Paperno et al., 2016) and LAMBADA OpenAI (Radford et al., 2019) datasets.

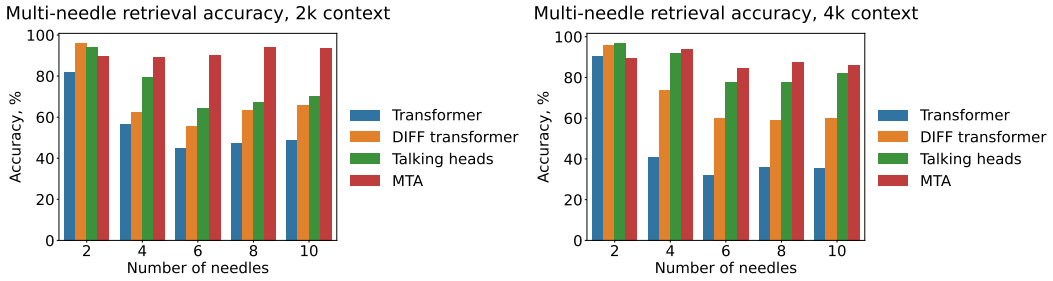

Figure 3: Multi-needle retrieval accuracy averaged over different needle insertion depths. Pretrained models are evaluated with 2K context (left), while finetuned models are evaluated with 4K context (right).

needles are shuffled, thus removing bias the models might have toward extracting the needle that was inserted first or last. Each setup is evaluated on 500 samples. Accuracy evaluations are averaged across the depths of insertion. As reported in Figure 3, we observe a significant improvement in needle extraction abilities for models trained with MTA across all needle counts and varying context lengths. Breakdown by depth is reported in Appendix G.

BabiLong This benchmark tries to to assess a language models' ability to reason across facts scattered in long documents. We focus on tasks QA1-5 (Weston et al., 2015) where a correct response requires various numbers of facts or argument relations. Examples of input and target output can be found in Table 6. We present average accuracy in Figure 5(left) and per-task in Appendix Figure 7. We observe that the MTA model performs well compared to other models, especially when there is more distraction text (4K tokens) in the input.

4.5 Kernel patterns

Key-query and head convolution kernels across different layers and heads are shown in Appendix H. We observe that a number of of the key-query kernels are close to identity, with near-one weight learned for targeted query and key, and near-zero weights learned for all other positions. However, there are numerous kernels that are more intricate.

One such kernel is shown in Figure 4(left), which has a diagonal structure. This kernel will amplify attention if a sequence of query tokens match a sequence of keys. Such ability is useful in searching sentences that matches the current sentence. Indeed, we observe that this particular kernel focuses attention on the target needle (see Figure 4(right)) in the Needle-in-the-Haystack task, where we must locate the magic number of the matching city.

Other kernels, for example, can be viewed as priming, amplifying if the same key was attended by previous queries (head 5 on Figure 10), or edge detecting, amplifying the first or last of multiple contiguous keys with high attention (heads 8 and 11 on Figure 12). We leave further exploration of key-query kernel maps and their meaning for future research.

Head kernel patterns, on the contrary, are simpler due to their small size, as shown in Appendix Figure 15. Besides an identity with scaling, a common pattern is contrasting: subtracting one attention weight from another. We also observe that the kernel scales increase with layers when there is no group normalization (see Appendix Figure 16). This is probably to compete with the residual stream, which gets larger with the model's depth.

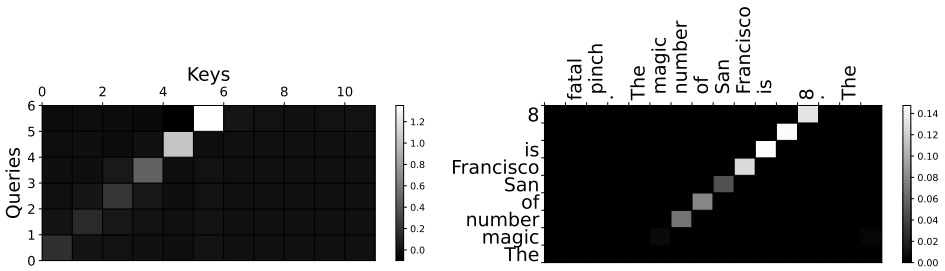

Figure 4: Kernel pattern (left) and corresponding attention map (right), which has the highest attention scores on the targeted needle "The magic number of San Francisco is 8". This kernel amplifies attention if a query token sequence matches a key sequence – useful for searching for related sentences that match the current sentence.

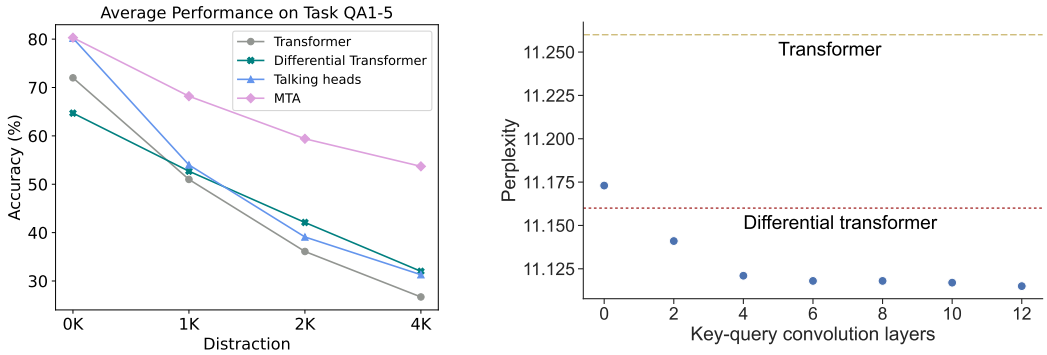

Figure 5: **(left)** Average accuracy on QA1-5 tasks in BabiLong. The models are all pretrained with 2K context and then finetuned with 4K context. Distraction text length varies from 0K (no distraction) to 4K tokens. MTA consistently outperforms the baseline models. **(right)** Ablation on the number of MTA layers with key-query convolutions (head convolution is applied to all layers). We report average validation perplexity on SlimPajama.

However, this pattern is not present with group normalization because it will undo the effect of such scaling.

4.6 Ablations

Key-query convolution To understand how key-query convolution affects the model performance, we run ablation studies on the number of layers where key-query convolution is added to the attention. The results shown in Figure 5(right) demonstrate that when only 2 layers are enhanced with MTA, the model can outperform strong baselines, while 6 layers with MTA strikes a balance between performance and additional complexity (see Appendix F for more about computational complexity).

Head convolution An ablation study on the head kernel size shows that increasing kernel size improves average validation perplexity, as shown in Figure 6 (left), allowing for better communication across heads.

Kernel initialization Kernel initialization can affect the convergence rate and stability of training. We evaluated MTA models initialized with zero, constant (0.3), and identity kernels. The latter corresponds to initialization with a regular Transformer without convolution in the attention. We found that identity initialization leads to better convergence and final performance, while models initialized with zero and 0.3 values reduce average validation perplexity by 0.02 and 0.08 correspondingly.

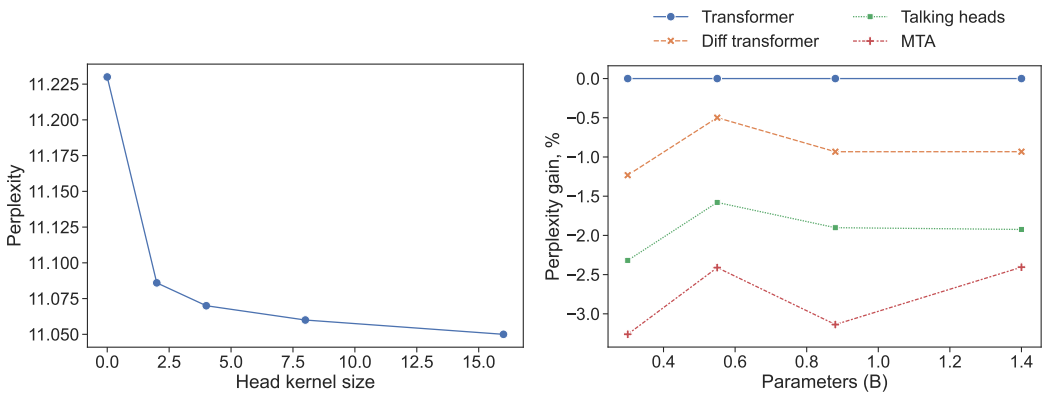

Figure 6: Ablation on the head kernel size in MTA convolution **(left)**, and scaling laws **(right)**. We report average validation perplexity on SlimPajama.

Key-query conv		Head conv			
Pre-sm	**Post-sm**	**Pre-sm**	**Post-sm**	**Group norm**	**PPL ↓**
✓	✓	✓	✓	scalar gating	**10.90**
✓	✓	✓	✓	depth scaling	10.92
✓	×	✓	✓	scalar gating	10.95
✓	×	✓	✓	×	10.99
✓	×	✓	✓	depth scaling	10.99
✓	×	✓	✓	no scaling	11.03
✓	×	×	✓	depth scaling	11.09
✓	×	×	✓	×	11.16
✓	×	×	✓	no scaling	11.13
✓	×	×	✓	layer-norm scaling	11.41
$c_q = 4, c_k = 9$	×	×	×	depth scaling	11.23
$c_q = 6, c_k = 11$	×	×	×	depth scaling	11.23
$c_q = 8, c_k = 13$	×	×	×	depth scaling	11.31
×	✓	×	✓	depth scaling	11.11
×	✓	×	✓	×	11.19
✓	×	✓	×	depth scaling	11.10
✓	×	✓	×	×	11.20

Table 5: Ablation on MTA components: validation perplexity over SlimPajama dataset.

Ablation on MTA components We further experiment with MTA kernels of different sizes, changing the order of convolution operations with respect to softmax, and different normalizations, such as layer-norm scaling (Sun et al., 2025). Results are summarized in Table 5. We found that different kernel sizes display similar kernel patterns, while resulting in slightly different evaluation results (middle rows). Group normalization and exponential depth scaling are both important factors each bringing improvement over vanilla MTA (top rows), while square root layer-norm scaling surprisingly underperforms. Finally, changing the order of convolutions wrt softmax operations increases perplexity only by 0.01-0.04 points (bottom rows).

Scaling laws We experiment with models of various sizes: 300M, 550M, and 1B, with hyperparameters listed in Table 7. As shown in Figure 6 (right), we observe a consistent pattern across models of different sizes, where MTA provides superior performance.

5 Related work

Focusing attention There has been a number of attempts to focus the soft attention mechanism on important context. For example, modifications have been proposed to make

softmax sharper. Martins & Astudillo (2016) propose to replace softmax with sparsemax for sparse activations. Veličković et al. (2024); Nakanishi (2025) propose Adaptive temperature and Scalable-Softmax that adjust exponential base in softmax attention. Irrelevant tokens can be altogether removed from memory (Sukhbaatar et al., 2021) to make attention easier. In Golovneva et al. (2024); Desrochers et al. (2024) the authors incorporate contextual information into position encodings to allow positions to be amplified by context. More recently, several noise canceling mechanisms have been proposed for attention. Talking heads attention (Shazeer et al., 2020) adds linear projections across the attention-heads dimension before and after the softmax operation. DIFF transformer (Ye et al., 2024) uses differential amplifiers to focus attention to the relevant context, which is related to our head mixing step. Cang et al. (2025) further extends it by introducing normalization steps. OpAmp adaptation (Wu et al., 2025) propose adapters, which are fine-tuned with noisy context to enhance an LLMs' denoising capability. Xu et al. (2024) modified attention so that keys and values can shift by one time step. This can be viewed as a special case of MTA where the convolution performs shifting in the key dimension.

Convolution in Attention Convolution layers have been used with attention (Gehring et al., 2017; Wu et al., 2019), especially in vision Transformers, to decompose each image into a sequence of tokens with fixed length, and then apply multiple standard Transformer layers (Xiao et al., 2021; Wu et al., 2021). There have been some attempts to include convolution in language modeling as well. For example, Liu et al. (2018) use convolution to *compress* the keys and values in the multi-headed attention by a factor of 3 thus allowing to process sequences 3x in length. Liu & Lapata (2019) later augment this architecture with the ability to encode multiple documents in a hierarchical manner, and further improve on long-context summarization tasks. Later Subramanian et al. (2020) propose three hierarchical models to learn different levels of abstraction in language by interchanging casual transformer layers with convolutions or pooling, or compressing representations for tokens further away in the past. Gulati et al. (2020) proposed a Conformer architecture for speech recognition, where a convolution module is applied to multi-head self-attention output. This architecture was later used to encode speech in the Llama-3 herd of models (Grattafiori et al., 2024). (Zheng et al., 2024) apply a convolution on the attention weights in the key dimension that enhances Transformer's extrapolation capabilities to long context. However, none of these methods apply convolutions across multiple queries, keys and heads to the attention weights.

6 Conclusion

In this paper, we focused on a limitation of the standard soft attention mechanism that stems from conditioning on the similarity of a single vector pairs. This makes it challenging for Transformers to precisely locate relevant information based on richer distinguishing information. As a remedy, we proposed a novel Multi-Token Attention mechanism that combines attention weights from multiple queries, keys and heads, making it possible to attend using more fine-grained information. From a simple motivating toy task to large-scale LLM experiments on a variety of popular benchmarks we demonstrate that models equipped with MTA achieve enhanced performance, especially when tested on tasks that require the precise location of relevant information within a long context.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, 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.

Yueyang Cang, Yuhang Liu, Xiaoteng Zhang, Erlu Zhao, and Li Shi. Dint transformer. *arXiv preprint arXiv:2501.17486*, 2025.

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.

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.

Sarah Desrochers, James Wilson, and Matthew Beauchesne. Reducing hallucinations in large language models through contextual position encoding, 2024.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pp. 1243–1252. PMLR, 2017.

Olga Golovneva, Tianlu Wang, Jason Weston, and Sainbayar Sukhbaatar. Contextual position encoding: Learning to count what's important. *arXiv preprint arXiv:2405.18719*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Gregory Kamradt. Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023.

Yury Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems*, 37:106519–106554, 2025.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Xiaoran Liu, Ruixiao Li, Mianqiu Huang, Zhigeng Liu, Yuerong Song, Qipeng Guo, Siyang He, Qiqi Wang, Linlin Li, Qun Liu, et al. Thus spake long-context large language model. *arXiv preprint arXiv:2502.17129*, 2025.

Yang Liu and Mirella Lapata. Hierarchical transformers for multi-document summarization. *arXiv preprint arXiv:1905.13164*, 2019.

Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pp. 1614–1623. PMLR, 2016.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Ken M Nakanishi. Scalable-softmax is superior for attention. *arXiv preprint arXiv:2501.19399*, 2025.

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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

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

Noam Shazeer, Zhenzhong Lan, Youlong Cheng, Nan Ding, and Le Hou. Talking-heads attention. *arXiv preprint arXiv:2003.02436*, 2020.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://cerebras.ai/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, June 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Sandeep Subramanian, Ronan Collobert, Marc'Aurelio Ranzato, and Y-Lan Boureau. Multi-scale transformer language models. *arXiv preprint arXiv:2005.00581*, 2020.

Sainbayar Sukhbaatar, Da Ju, Spencer Poff, Stephen Roller, Arthur Szlam, Jason Weston, and Angela Fan. Not all memories are created equal: Learning to forget by expiring. In *International Conference on Machine Learning*, 2021. URL https://api.semanticscholar.org/CorpusID:234681615.

Wenfang Sun, Xinyuan Song, Pengxiang Li, Lu Yin, Yefeng Zheng, and Shiwei Liu. The curse of depth in large language models. *arXiv preprint arXiv:2502.05795*, 2025.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Petar Veličković, Christos Perivolaropoulos, Federico Barbero, and Razvan Pascanu. softmax is not enough (for sharp out-of-distribution). *arXiv preprint arXiv:2410.01104*, 2024.

Mathurin Videau, Badr Youbi Idrissi, Daniel Haziza, Luca Wehrstedt, Jade Copet, Olivier Teytaud, and David Lopez-Paz. Meta Lingua: A minimal PyTorch LLM training library, 2024. URL https://github.com/facebookresearch/lingua.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019.

Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 22–31, 2021.

Haoyuan Wu, Rui Ming, Haisheng Zheng, Zhuolun He, and Bei Yu. Efficient opamp adaptation for zoom attention to golden contexts. *arXiv preprint arXiv:2502.12502*, 2025.

Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick. Early convolutions help transformers see better. *Advances in neural information processing systems*, 34:30392–30400, 2021.

Mingyu Xu, Wei Cheng, Bingning Wang, and Weipeng Chen. Kv shifting attention enhances language modeling. *ArXiv*, abs/2411.19574, 2024. URL https://api.semanticscholar.org/CorpusID:274422840.

Tianzhu Ye, Li Dong, Yuqing Xia, Yutao Sun, Yi Zhu, Gao Huang, and Furu Wei. Differential transformer. *arXiv preprint arXiv:2410.05258*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Chuanyang Zheng, Yihang Gao, Han Shi, Jing Xiong, Jiankai Sun, Jingyao Li, Minbin Huang, Xiaozhe Ren, Michael Ng, Xin Jiang, et al. Dape v2: Process attention score as feature map for length extrapolation. *arXiv preprint arXiv:2410.04798*, 2024.

A Limitations

To the best of our knowledge, the Multi-Token Attention method is not currently compatible with the popular optimized attention kernels. Optimizing the runtime performance was not the goal of this work, thus we leave further optimization of the MTA implementation for future research.

B MTA Architecture

Head convolution vs normal attention with higher rank Let us consider mixing of two heads after softmax. If W_o^1 and W_o^2 are the output projections corresponding to those heads, their output can written as

$$\begin{aligned}
O &= (A^1_{\text{new}} V^1) W_o^1 + (A^2_{\text{new}} V^2) W_o^2 \\
&= (w_{11} A^1 V^1 + w_{12} A^2 V^1) W_o^1 + (w_{21} A^1 V^2 + w_{22} A^2 V^2) W_o^2 \\
&= A^1 (w_{11} V^1 W_o^1 + w_{21} V^2 W_o^2) + A^2 (w_{12} V^1 W_o^1 + w_{22} V^2 W_o^2) \\
&= A^1 (w_{11} H W_v^1 W_o^1 + w_{21} H W_v^2 W_o^2) + A^2 (w_{12} H W_v^1 W_o^1 + w_{22} H W_v^2 W_o^2) \\
&= A^1 H (w_{11} W_v^1 W_o^1 + w_{21} W_v^2 W_o^2) + A^2 H (w_{12} W_v^1 W_o^1 + w_{22} W_v^2 W_o^2)
\end{aligned}$$

If we compare the first term to a normal attention output $A^1 H W_v^1 W_o^1$, we note that the only difference is that now we have two rank-d matrix additions instead of one. Actually we can rewrite

$$w_{11} W_v^1 W_o^1 + w_{21} W_v^2 W_o^2 = \hat{W}_1^v \hat{W}_1^o$$

where $\hat{W}_v^1 \in \mathbb{R}^{D \times 2d}$ and $\hat{W}_o^1 \in \mathbb{R}^{2d \times D}$. Thus, post-softmax head convolution can be replicated by a normal attention head with twice the rank. This may help in understanding why head mixing is useful, as it may increase the expressive power using a higher rank. However, it is not truly identical to $2\times$ rank because the parameters of the two heads are not independent.

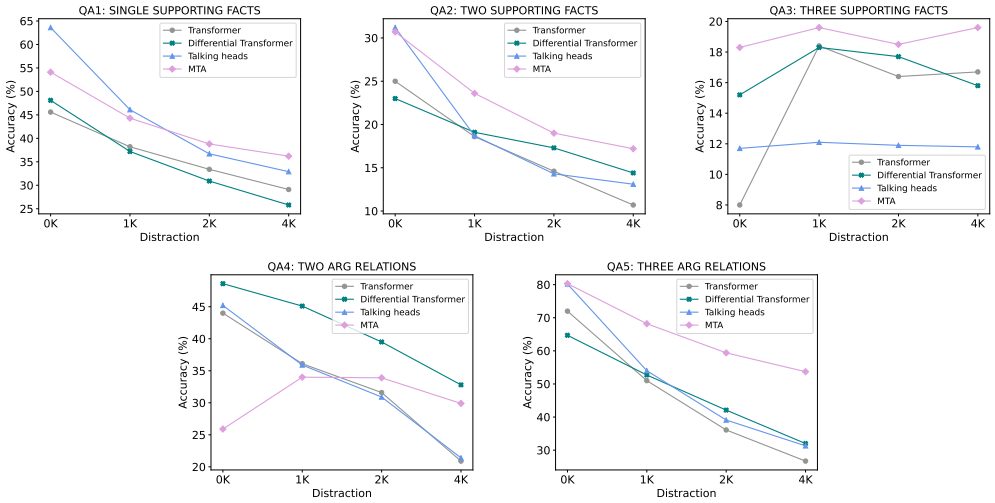

Figure 7: Accuracy (%) on QA1-5 tasks in BabiLong benchmark. Note the random performance on QA3 is 16.67%, thus all models perform poorly on QA3. On other tasks, MTA demonstrates strong performance compared to baselines especially when there is lengthy distraction text (4K).

Let us see if the same is true for the pre-softmax version. The attention logits after mixing is

$$\begin{aligned}\hat{A}^1 &= w_{11}Q^1K^{1\top} + w_{12}Q^2K^{2\top} \\ &= w_{11}HW_q^1W_k^{1\top}H^\top + w_{12}HW_q^2W_k^{2\top}H^\top \\ &= H(w_{11}W_q^1W_k^{1\top} + w_{12}W_q^2W_k^{2\top})H^\top\end{aligned}$$

Again, this can be rewritten like a normal attention logits

$$\hat{A}^1 = H\hat{W}_q^1\hat{W}_k^{1\top}H^\top$$

where $\hat{W}_q^1, \hat{W}_k^1 \in \mathbb{R}^{D\times 2d}$. As before, it can be replicated by a normal attention with twice the rank in the key and query projections.

C Toy task details

The blocks are built by randomly choosing $N \in \{5, 8\}$ lowercase alphabets and concatenating them. We join up to 50 blocks into a single sequence separated by ".", followed by "#" and $L = 2$ question letters. Here is an example sequence:

`hjnvt.qfjgt.whftb.bjtpq.` *...(many blocks)...* `.pxjvf.ulhik.qoiax#pb`

Here the 4th block "`bjtpq`" is the target because it contains all query letters "`pb`". During data generation, we make sure there is only one target block in each sequence. We use 1M such sequences for training and test on held-out 1K sequences.

We train a small Transformer model with 4 layers, 2 heads and 256 hidden dimensions. The batch size is 64 and the training continues for a total of 100K steps. We run each experiment three times with different random seeds and report the average performance.

D Language model training details

Training setup follows LLaMa architecture (Touvron et al., 2023), and includes RMSNorm pre-normalization (Zhang & Sennrich, 2019), SwiGLU activation function (Shazeer, 2020),

Task	Name	Input	Target	Question
QA1	single supporting fact	John travelled to the hallway. Mary journeyed to the bathroom. Daniel went back to the bathroom. John moved to the bedroom.	bathroom	Where is Mary?
QA2	two supporting fact	Mary journeyed to the bathroom. ... Daniel grabbed the football there. Sandra grabbed the milk there. Daniel went to the kitchen.	kitchen	Where is the football?
QA3	three supporting fact	Daniel moved to the bathroom. ... Mary got the football. Mary went back to the kitchen. Mary journeyed to the garden.	kitchen	Where was the football before the garden?
QA4	two arg relations	The hallway is east of the bathroom. The bedroom is west of the bathroom.	bedroom	What is the bathroom east of?
QA5	three arg relations	Fred picked up the football there. Fred gave the football to Jeff. ... Jeff gave the football to Fred. Fred gave the football to Jeff.	Jeff	Who did Fred give the football to?

Table 6: Examples of QA1-5 tasks in BabiLong benchmark when there is no distraction context.

and Rotary Embeddings (Su et al., 2024). Table 7 shows a detailed hyperparameter breakdown for Transformer model pretraining. All models were trained in the same setup, except Diff Transformer, which had twice less heads to account for doubled dimension.

We evaluate on the following tasks: BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC easy and challenge (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018). We also report 5-shot performance on the aggregated MMLU benchmark (Hendrycks et al., 2020).

E BabiLong details

We include examples of input and output from BabiLong benchmark in Table 6. The details of per-task evaluation results on QA1-5 are shown in Figure 7.

F Complexity evaluation

Table 8 shows estimated additional number of parameters and their actual counts. We note that DIFF transformer (Ye et al., 2024) introduces four learnable vectors per layer

Parameter	300M model	550M model	880M model	1B model
Dimension, D	1024	1280	1536	2048
Layers	20	24	24	24
Heads, M	16	10	16	16
RoPE theta	100,000	100,000	100,000	100,000
Batch size (tokens)	262,144	262,144	262,144	524,288
Context length	2048	2048	2048	2048
Learning rate	$1.5e-4$	$1.5e-4$	$1.5e-4$	$1.5e-4$
Weight decay	0.05	0.05	0.05	0.05
Warm up steps	375	375	375	187

Table 7: Hyperparameters for Transformer training.

Model	Additional parameters, estimates	Actual number of parameters
Transformer		876,553,728
DIFF transformer	$4 * L * d + 2 * L * d$	876,567,552
Talking heads	$2 * L * M * M$	876,566,016
MTA	$2 * L * M(c_q * c_k/4 + c_h) + L * (2 * d + 1)$	876,583,320

Table 8: Estimation of the parameter counts for each model architecture.

Model	Memory, GB ↓	FLOPS, 10^{13} ↑	TPS, 10^3 ↑
Transformer	17.5	25.0	54.3
DIFF transformer	21.6	8.4	17.6
Talking heads	63.0	6.9	14.6
MTA	73.8	2.6	5.7

Table 9: Average memory consumption and training speed for models trained on 32 NVIDIA H200 GPUs with our unoptimized code.

$(\lambda_{q1}, \lambda_{k1}, \lambda_{q2}, \lambda_{k2})$, plus group normalization weights. In total, that results in more parameters than Multi-Token Attention with key-query convolution added to the quarter of layers, even though all kernel weights in MTA models are different between all heads and layers.

Table 9 shows actual runtime and memory consumption recorded during 880M model training. Note that the baseline and DIFF transformer utilize *scaled dot product attention*[3] function implemented in Torch, that calls optimized CUDA kernels for improved performance. In contrast, our MTA implementation does not take advantage of such efficient kernels, which is the major reason behind its lower FLOPS.

G Multi-needle evaluation results

We report on the detailed evaluation results in Figure 8. We observe that models trained with MTA are better at finding needles hidden deeper in the context.

H MTA kernel patterns.

Key-query convolution kernels across different layers and heads are reported in Figure 9-Figure 14.

Head kernel patterns for the model with c_h=2 are shown in Figure 15. They are simpler due to their small size. Besides an identity with scaling, a common pattern is contrasting: subtracting one attention weight from another. We also observe that the kernel scales

[3] https://pytorch.org/docs/stable/generated/torch.nn.functional.scaled_dot_product_attention.html

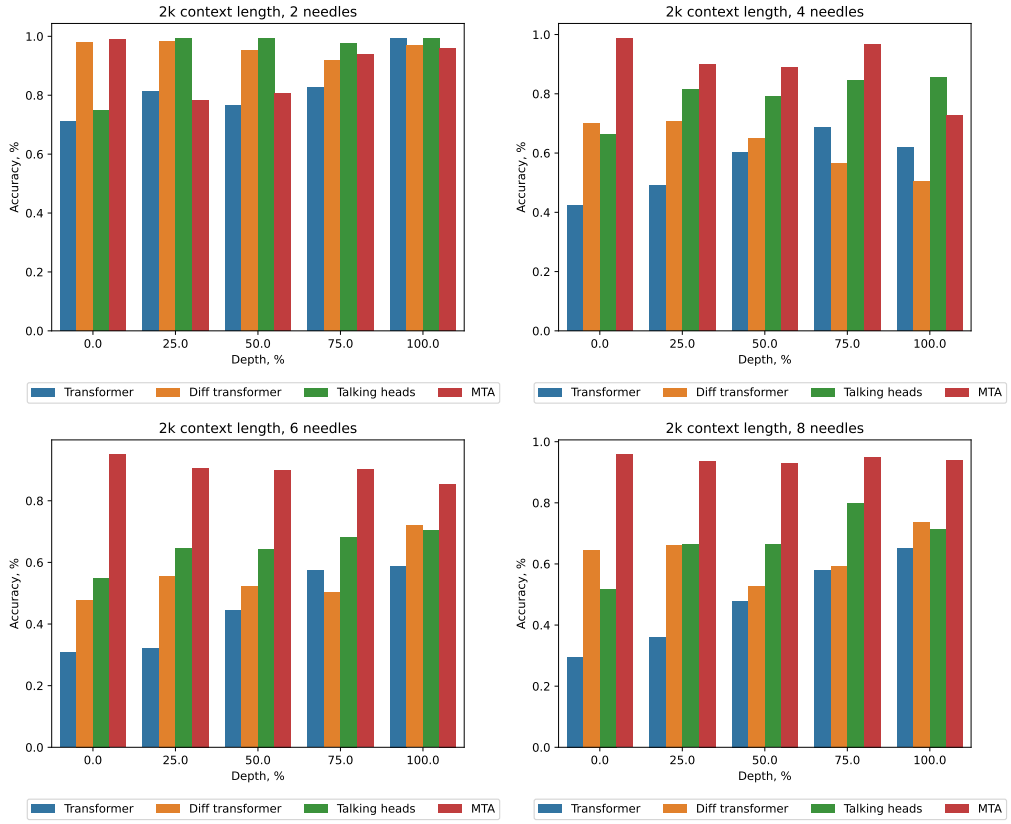

Figure 8: Needle-in-a-Haystack accuracy across different models. Models trained with MTA are better at predicting needles hidden deep in the context.

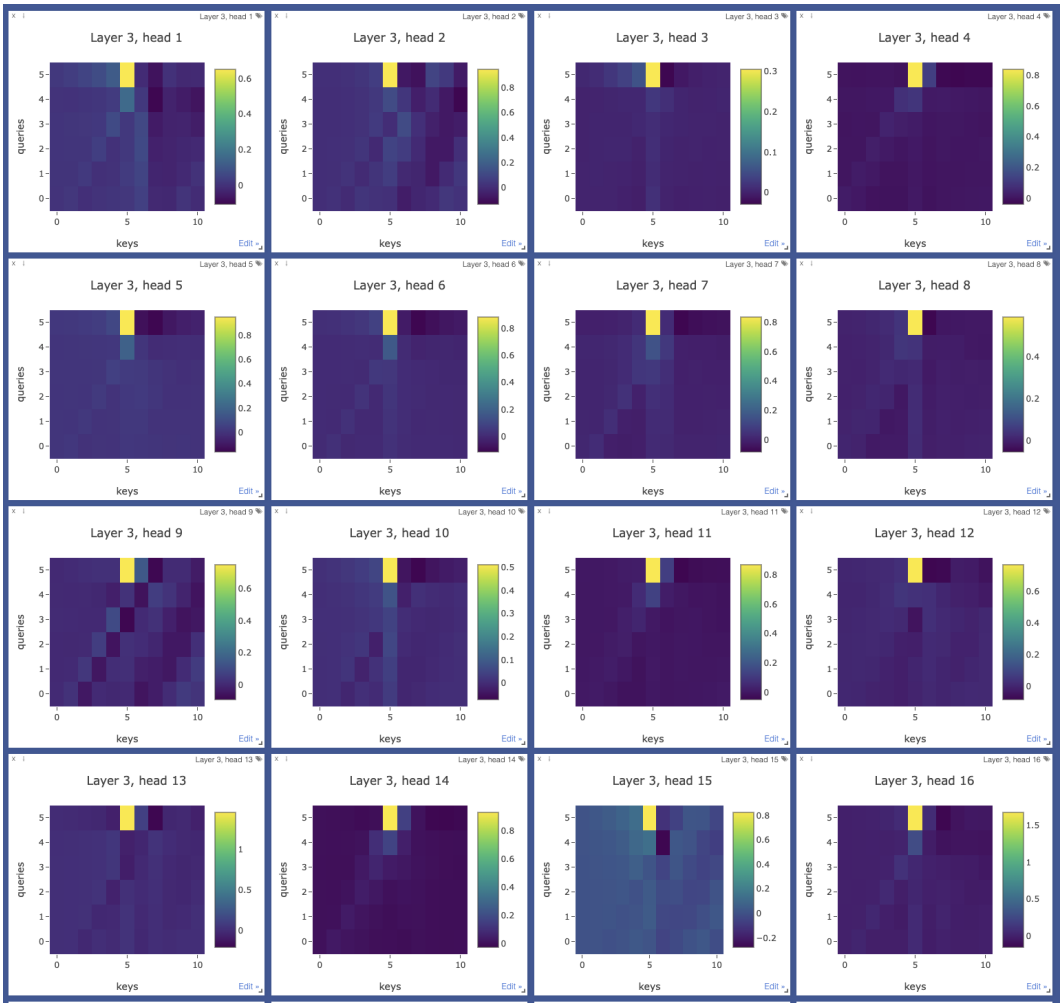

Figure 9: Key-query kernel patterns across heads at 3rd layer of the Transformer model with MTA.

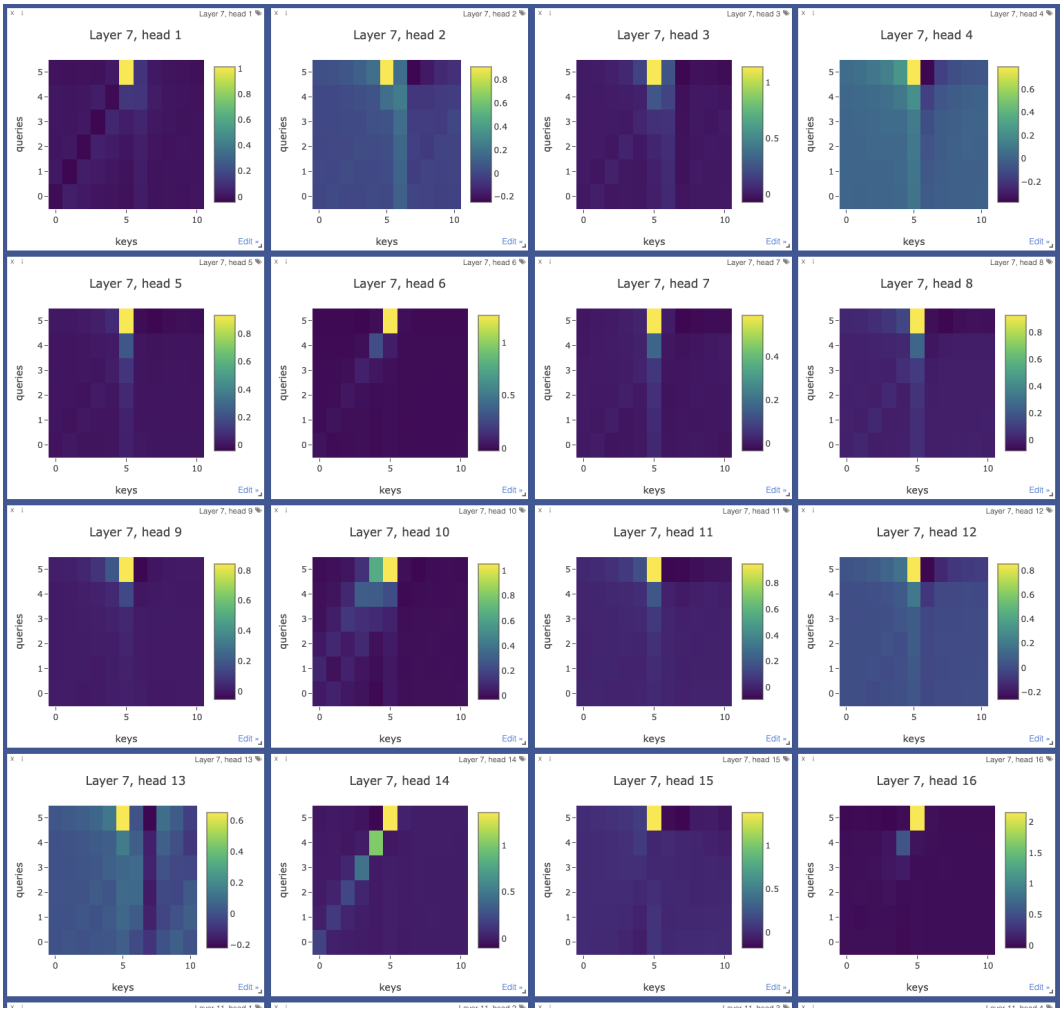

Figure 10: Key-query kernel patterns across heads at 7th layer of the Transformer model with MTA.

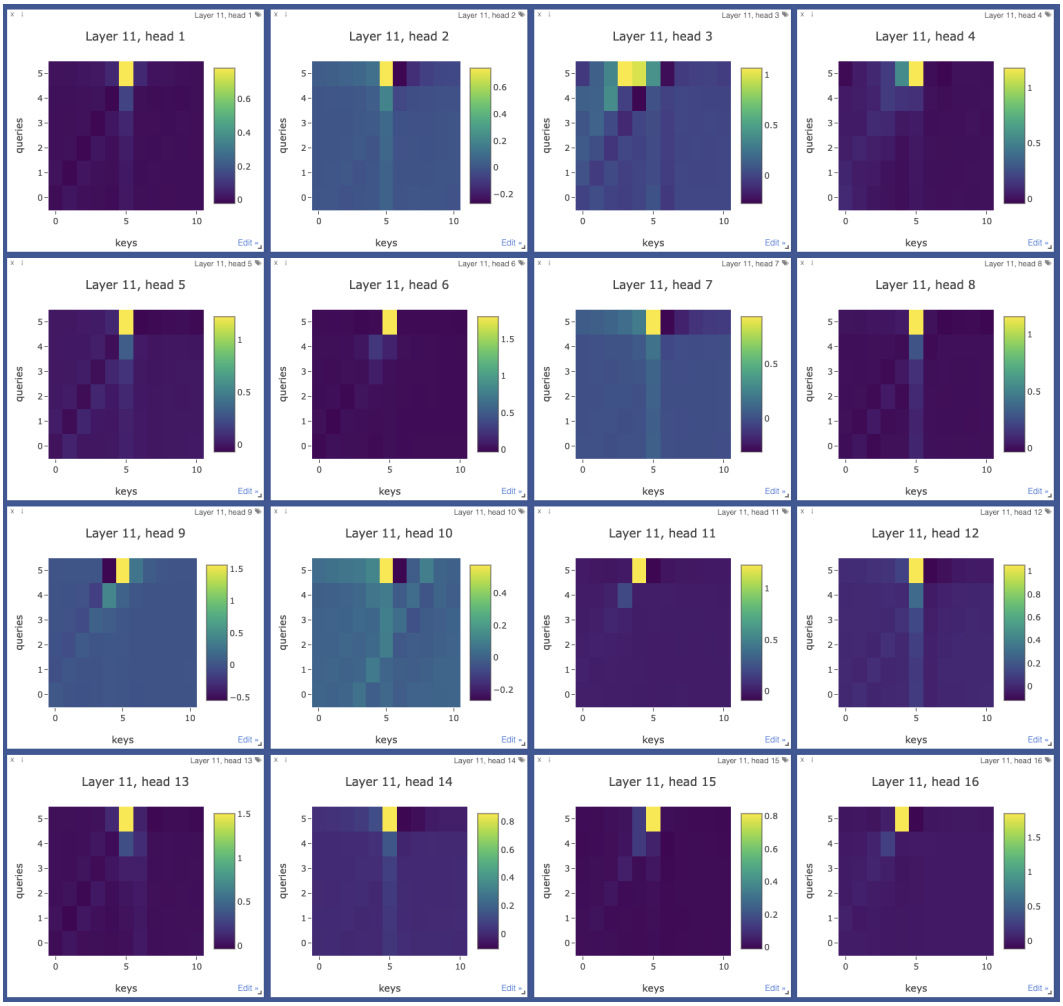

Figure 11: Key-query kernel patterns across heads at 11th layer of the Transformer model with MTA.

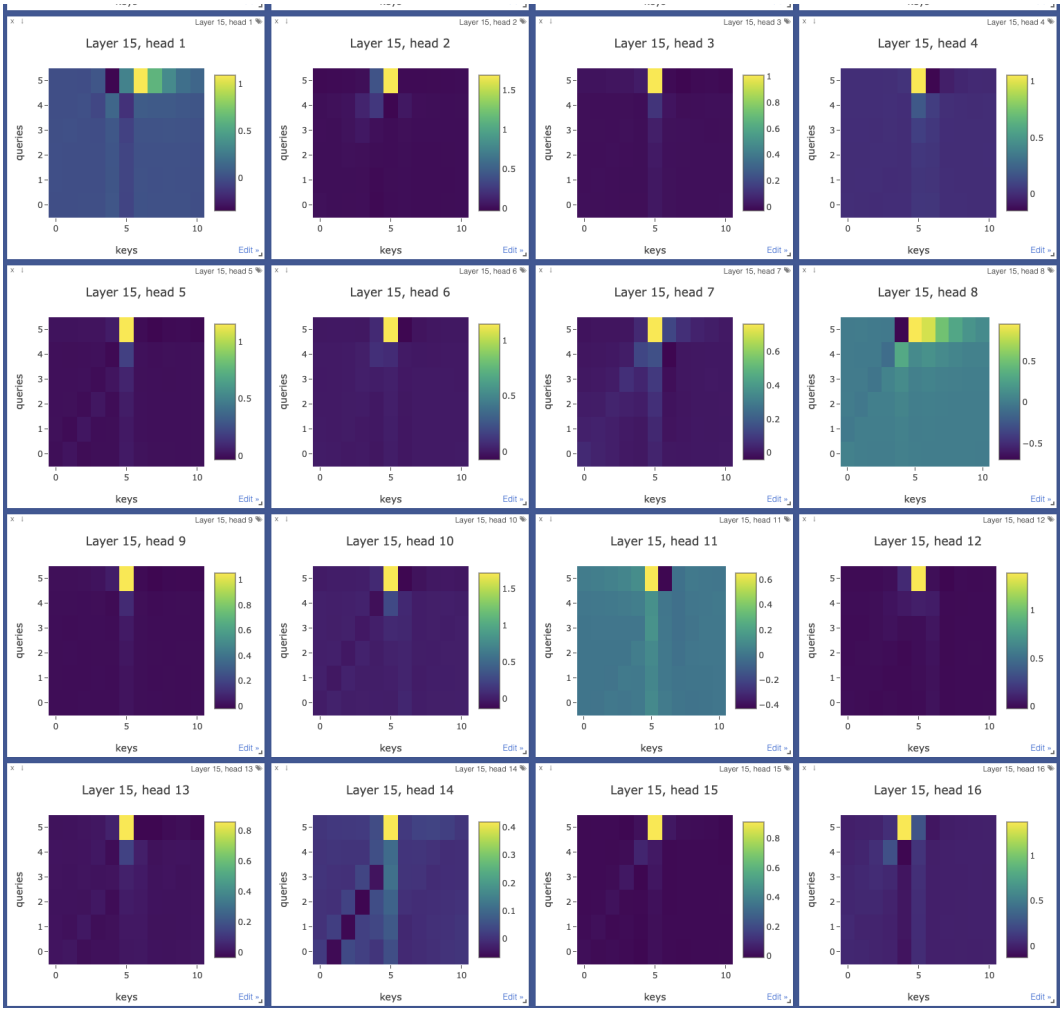

Figure 12: Key-query kernel patterns across heads at 15th layer of the Transformer model with MTA.

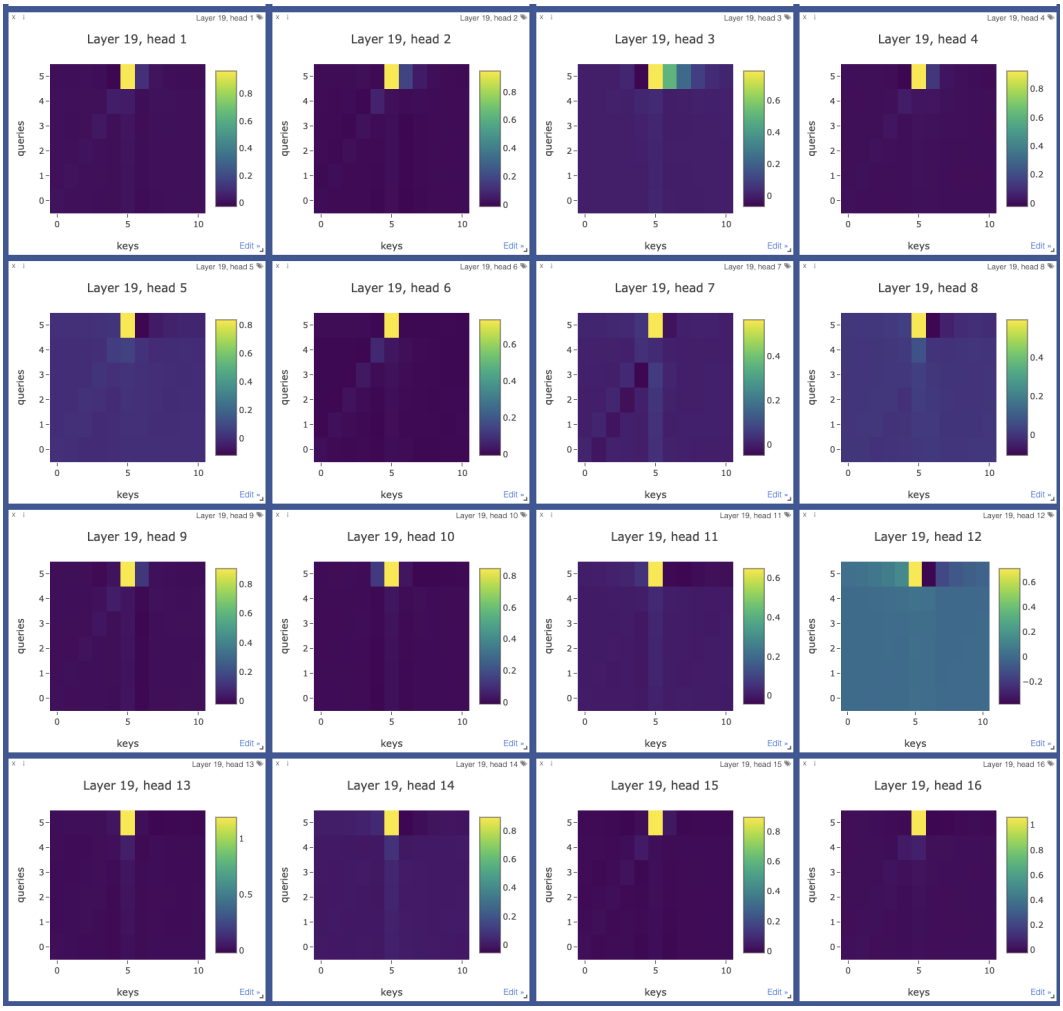

Figure 13: Key-query kernel patterns across heads at 19th layer of the Transformer model with MTA.

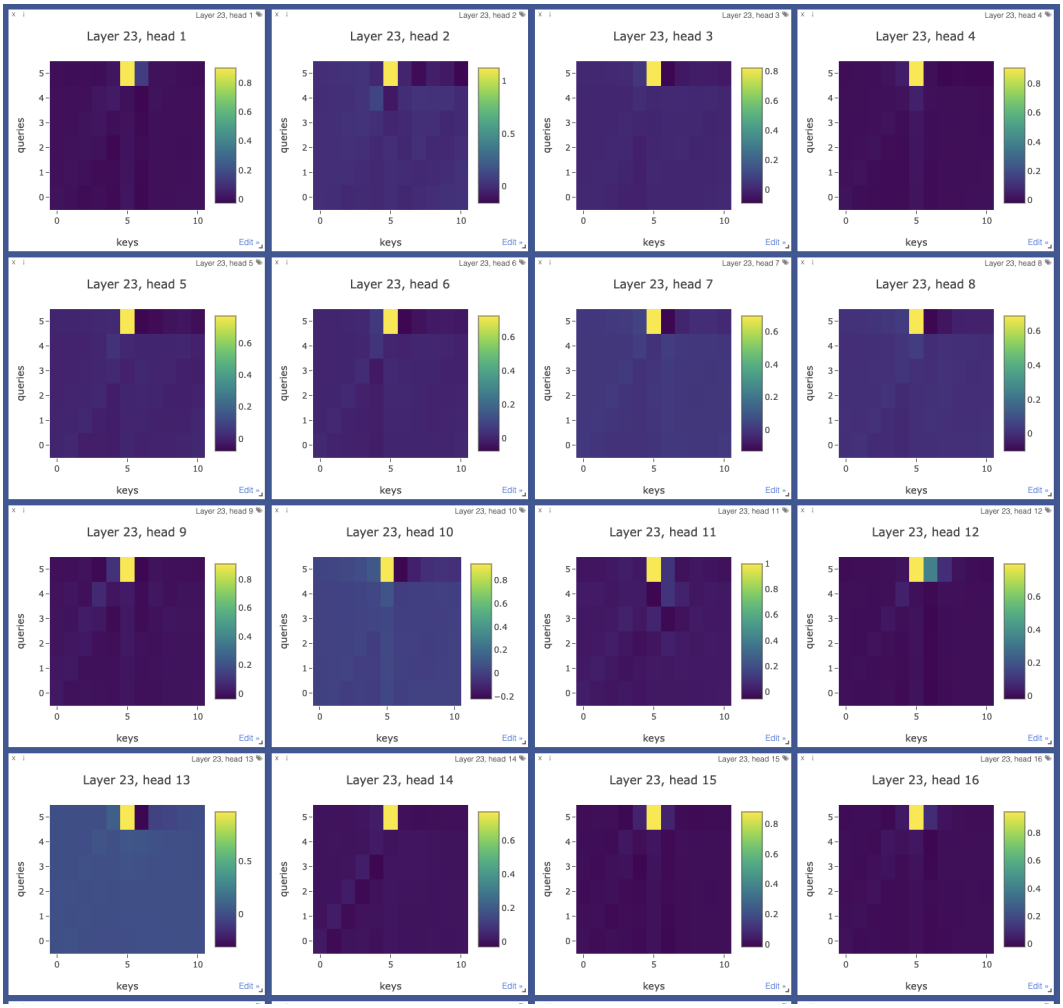

Figure 14: Key-query kernel patterns across heads at 23rd layer of the Transformer model with MTA.

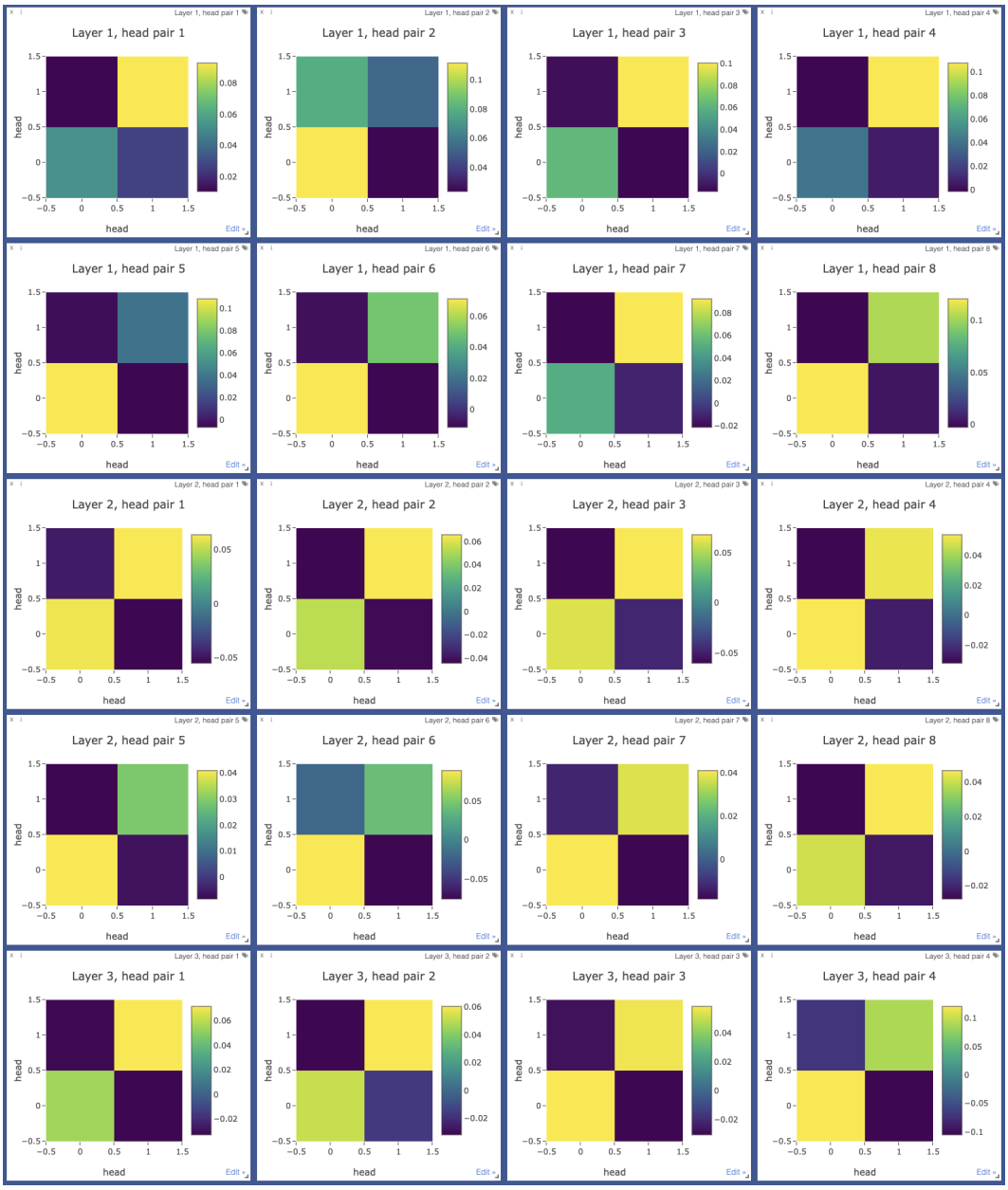

Figure 15: Head kernel patterns across first three layers of the Transformer model with MTA.

increase with layers when there is no group normalization (see Figure 17). This is probably to compete with the residual stream, which gets larger with the model's depth. However, this pattern is not present with group normalization because it will undo the effect of such scaling.

I Finetuning with MTA

One natural question readers might ask is if MTA could be integrated into models that were already trained with standard attention? Would this require complete retraining, or could it be added through some form of adaptation? Architectually, MTA can be added as additional layers to already trained models, and weights can be updated with continual

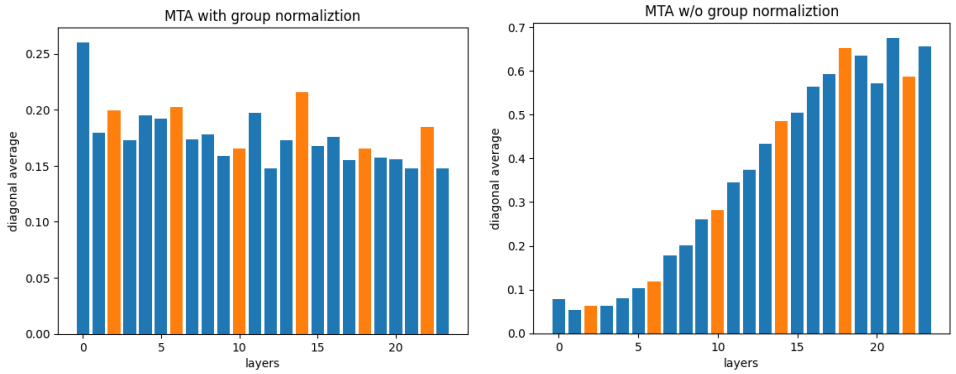

Figure 16: Average of the diagonal kernel values across head pairs for Multi-Token Attention models with (left) and without (right) group normalization. Values for layers with key-query convolution are colored in orange.

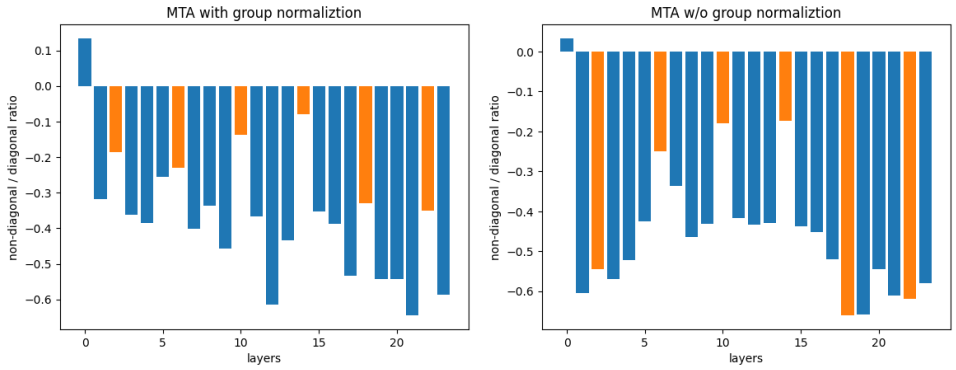

Figure 17: Average of the ratio between non-diagonal and diagonal kernel values across head pairs for Multi-Token Attention models with (left) and without (right) group normalization. Values for layers with key-query convolution are colored in orange.

Pretraining	Cont training	arxiv	book	c4	cc	github	se	wiki	Avg PPL ↓
1.4B models									
Transformer	Transformer	4.54	12.87	19.21	13.77	3.99	9.49	10.96	10.69
Transformer	MTA	4.51	12.78	19.09	13.67	3.95	9.41	10.87	10.61
MTA	MTA	**4.47**	**12.57**	**18.82**	**13.47**	**3.89**	**9.25**	**10.65**	**10.44**
Llama 3 herd									
Llama 3.2 1B	Llama 3.2 1B	4.54	14.60	18.14	13.40	3.92	8.84	10.25	10.53
Llama 3.2 1B	MTA	4.52	14.49	18.04	13.32	3.89	8.79	10.19	10.46
Llama 3.2 3B	Llama 3.2 3B	4.12	12.08	15.58	11.50	3.37	7.52	8.35	8.93
Llama 3.2 3B	MTA	4.11	12.03	15.51	11.46	3.35	7.48	8.31	8.89
Llama 3.1 8B	Llama 3.1 B	4.00	11.04	15.04	10.99	3.29	7.33	8.00	8.53
Llama 3.1 8B	MTA	**3.98**	**10.97**	**14.97**	**10.94**	**3.27**	**7.29**	**7.97**	**8.48**

Table 10: Validation perplexity on SlimPajama dataset after continuous training for 10.5B tokens on our 1.4B models, and continuous training for 5.3B tokens for Llama 3 models.

training. Since the main component of MTA is a convolution, we can initialize it to identity and insert it in an existing a Transformer layer. Such a modification will not change the output of the layer, so when we start training the model, it should maintain its performance. However, as the added convolution starts deviating from the identity state, it will allow the transformer to condition its attention on multiple keys and queries.

To understand how well pre-trained models can learn we perform preliminary experiments by finetuning our 1.4B model, as well as opensourced Llama models (Grattafiori et al., 2024). In all these experiments we used the same finetuning setup similar to Section 4.3, but kept the context length at 2048 tokens.

Validation perplexity results are reported in Table 10. We observe that all models finetuned with MTA were not only able to incorporate new kernels, but outperform baselines in terms of perplexity.