MKA: Memory-Keyed Attention for Efficient Long-Context Reasoning

Dong Liu [1 2 3] **Yanxuan Yu** [4] **Xuhong Wang** [5] **Ben Lengerich** [3] **Ying Nian Wu** [2]

Abstract

As long-context language modeling becomes increasingly important, the cost of maintaining and attending to large Key/Value (KV) caches grows rapidly, becoming a major bottleneck in both training and inference. While prior works such as Multi-Query Attention (MQA) and Multi-Latent Attention (MLA) reduce memory by sharing or compressing KV features, they often trade off representation quality or incur runtime overhead. We propose Memory-Keyed Attention (MKA), a hierarchical attention mechanism that integrates multi-level KV caches—local, session, and long-term—and learns to route attention across them dynamically. We further introduce FastMKA, a broadcast-routed variant that fuses memory sources before attention computation for enhanced efficiency. Experiments on different sequence lengths show that MKA improves perplexity over MHA and MLA, while FastMKA achieves comparable accuracy to MLA with up to 4× faster training and 40% lower evaluation latency. These results highlight MKA as a practical and extensible framework for efficient long-context attention.

1. Introduction

Large Language Models (LLMs) have rapidly advanced and are now capable of processing context lengths up to 128K or even 1M tokens (Peng et al., 2023; Jiang et al., 2024; Tworkowski et al., 2023). This unprecedented expansion of context length unlocks new applications such as long-document reasoning and multi-turn dialogue with persistent memory. However, supporting long contexts during inference introduces severe memory and latency bottlenecks, particularly due to the scaling cost of attention with Key/Value (KV) caches.

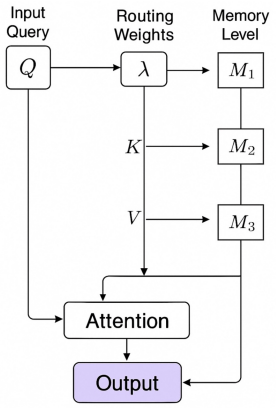

Figure 1. MKA hierarchical memory design

In self-attention, each newly generated token must attend to all past tokens stored in the KV cache. This results in quadratic compute and increasingly large memory reads. For instance, with a context length of 32K, the KV cache for a LLaMA-7B model (Touvron et al., 2023) can occupy up to 16GB and take over 11ms to access on a single GPU, which dominates more than 50% of the inference latency. This cost limits the throughput of LLMs in production environments.

Several methods have attempted to address the inefficiencies in attention computation. Multi-Query Attention (MQA) (Shazeer, 2019) and Grouped-Query Attention (GQA) (Ainslie et al., 2023) reduce KV duplication by sharing them across heads. Multi-Latent Attention (MLA) (Team, 2024) compresses the KV cache via low-rank factorization. However, these methods sacrifice representation fidelity or lack the flexibility to differentiate among memory types.

In this paper, we propose **Memory-Keyed Attention (MKA)**, a novel attention mechanism that hierarchically organizes memory into three levels—**local (L1)**, **session (L2)**, and **long-term (L3)**—and dynamically learns how to route each query token across these sources. As illustrated in figure 1, MKA leverages lightweight routing gates that modulate attention over heterogeneous memory types. This design enables context-aware attention with significantly lower memory bandwidth and improved token reuse.

[1]Yale University [2]University of California - Los Angeles [3]University of Wisconsin-Madison [4]Columbia University [5]Shanghai AI Lab. Correspondence to: Dong Liu <dong.liu.dl2367@yale.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

To ensure scalability, MKA adopts a block-wise softmax implementation inspired by FlashAttention (Dao et al., 2022) and supports GPU-efficient kernel fusion. Moreover, long-term memory (L3) is indexed via semantic chunking and vectorized hashing, allowing the model to recall relevant historical content efficiently. Our experiments shows that MKA can reduce training time & evaluation latency significantly compared to MHA, MQA, GQA, MLA. Our contribution can be introduced as follows:

- We identify the inefficiencies of existing long-context attention mechanisms and propose a hierarchical memory system tailored for attention routing.
- We introduce Memory-Keyed Attention (MKA), a dynamic routing mechanism across local, sessional, and long-term memory with hardware-friendly implementations, we further accelerate MKA training and inference by designing broad-cast routing (FastMKA).
- We show that FastMKA outperforms prior methods in both accuracy and efficiency, making it suitable for high-throughput LLM inference with long-context inputs.

2. Related Work

2.1. Long-context Attention Mechanisms

As LLMs are scaled to handle inputs of 32K, 128K, or even 1M tokens (Peng et al., 2023; Shukor et al., 2025), the challenge of performing efficient attention over long contexts has become a major bottleneck. A key issue lies in the size and bandwidth cost of Key/Value (KV) caches. Many efforts have been proposed to reduce attention overhead in this setting. FlashAttention (Dao et al., 2022) improves memory throughput by computing softmax in a tiled, IO-aware fashion. Multi-Query Attention (MQA) (Shazeer, 2019) and Grouped-Query Attention (GQA) (Ainslie et al., 2023) reduce KV duplication by sharing across heads, thus reducing cache size to $\frac{1}{H}$ and $\frac{g}{H}$ respectively.

More recent works like Multi-Latent Attention (MLA) (Team, 2024) compress attention by factorizing the KV memory into a smaller latent space. However, these approaches are limited to static memory layouts and struggle with retrieving long-range or task-specific context. In contrast, our proposed **MKA** generalizes these ideas through a *hierarchical memory design* with dynamic routing, enabling scalable and query-aware memory access across different time horizons.

2.2. Hierarchical and External Memory Systems

Incorporating multiple memory timescales has long been a goal in neural network design. Early memory-augmented models such as Memory Networks (Weston et al., 2015) and Differentiable Neural Computers (Graves et al., 2016) support explicit memory reads and writes, but are difficult to scale to large language models. More practical approaches such as Transformer-XL (Dai et al., 2019) cache segments of hidden states to extend effective context length, while Compressive Transformers (Rae et al., 2020) introduce a two-level memory with lossy compression.

Retrieval-Augmented Generation (RAG) (Lewis et al., 2021) and RETRO (Borgeaud et al., 2022) retrieve similar documents from a corpus during inference, effectively serving as external long-term memory. However, these methods rely on external index structures and are typically applied at the sequence level. By contrast, **MKA** integrates *internal multi-level memory*—local, session, and long-term—within the model, and learns per-token routing to dynamically select which level to attend to during each step of generation.

2.3. Dynamic Routing and Query-aware Attention

Routing-based attention has recently gained interest due to its potential for conditional computation. Sparse Mixture-of-Experts (MoE) models (Lepikhin et al., 2021; Fedus et al., 2022) route tokens through different feedforward networks, though not through memory. Routing Transformers (Roy et al., 2021) cluster tokens before performing sparse attention within groups, but rely on static clustering and do not model multiple memory levels. Query-aware caching methods such as TOVA (Oren et al., 2024) and Quest (Tang et al., 2024) focus on loading the most relevant KV cache pages based on current queries, but typically discard unused tokens.

In contrast, **MKA** retains the full cache but dynamically routes attention across three memory tiers. Our approach supports query-dependent access, soft selection (via learned weights), and slot reuse—achieving high memory efficiency without sacrificing accuracy.

3. Motivation: Beyond MLA and MHA

The design of MLA achieves KV compression through low-rank projections and shared K/V structures across heads. However, it does not explicitly support heterogeneous memory sources, nor does it allow selective reuse of memory slots. MKA extends this direction by introducing 3-level memory:

- **L1: Local cache** for current window tokens (standard causal attention).
- **L2: Session memory**, derived from low-rank summary or gated history.
- **L3: Long-term memory**, explicitly indexed and retrieved from a dynamic memory bank.

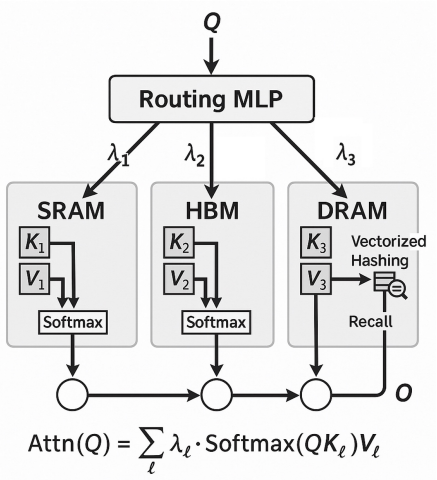

Figure 2. Hierarchical Memory-Keyed Attention (MKA) with Multi-Level Routing

To select between memory sources, MKA employs a **routing gate** $\lambda_\ell \in \mathbb{R}^3$ that is dynamically learned per query token.

4. Methodology

In this section, we present the design of **Memory-Keyed Attention (MKA)**. We begin by analyzing the bottlenecks of long-context inference, then introduce the hierarchical MKA structure, and follow with symbolic and CUDA-style pseudocode, tiled execution, and a theoretical formulation that supports recursive attention computation with memory efficiency. The detailed pseudocode can be found at appendix.

4.1. FastMKA: Accelerate MKA by Broadcast Routing

While MKA enables dynamic query-aware access to multi-level memories (L1, L2, L3), it requires repeated projection and attention computation over each memory source, leading to non-trivial overhead. To alleviate this, we propose **FastMKA**, a simplified yet effective variant that performs *broadcast routing*—a soft fusion of hierarchical memory levels *before* attention—thereby avoiding multiple attention paths and significantly reducing runtime cost.

In FastMKA, the local, session, and long-term memory representations are broadcasted and weighted via a shared routing MLP. The resulting fused memory is then used to generate a single key-value pair for attention. This mechanism retains the benefits of multi-level memory while incurring only one round of attention computation, making FastMKA highly efficient and compatible with standard Transformer pipelines.

Algorithm 1 Symbolic MKA: Memory-Keyed Attention with Hierarchical Routing

Require: Input $X \in \mathbb{R}^{B\times S\times D}$
Ensure: Output $O \in \mathbb{R}^{B\times S\times D}$, KV cache (K, V)
$q \leftarrow XW_q, \quad q \in \mathbb{R}^{B\times S\times D}$
$q_h \leftarrow \text{reshape}(q) \rightarrow \mathbb{R}^{B\times H\times S\times d_h}$
{Define memory levels}
$M_1 \leftarrow X$ {Local memory}
$M_2 \leftarrow \text{repeat}(\text{mean}(X, \text{dim} = 1))$ {Block memory}
$M_3 \leftarrow \mathbf{0} \in \mathbb{R}^{B\times S\times D}$ {Long-term memory}
$\lambda \leftarrow \text{softmax}(\text{MLP}(q)) \in \mathbb{R}^{B\times S\times 3}$
for $\ell = 1$ to 3 **do**
  $k_\ell \leftarrow M_\ell W_k, \quad v_\ell \leftarrow M_\ell W_v$
  $k_\ell^h \leftarrow \text{reshape}(k_\ell), \quad v_\ell^h \leftarrow \text{reshape}(v_\ell)$
  $a_\ell \leftarrow \text{softmax}(q_h \cdot {k_\ell^h}^\top) \cdot v_\ell^h$
end for
$O_h \leftarrow \sum_{\ell=1}^{3} \lambda_\ell \odot a_\ell$ {Fused multi-memory output}
$O \leftarrow \text{reshape}(O_h) \cdot W_o$
$K \leftarrow [K \| k_1^h], \quad V \leftarrow [V \| v_1^h]$ {Update cache}
return $(O, (K, V))$

4.2. Block-Memory Keyed Attention (Block-MKA) Design

Block-MKA introduces a multi-level memory design with the following key contributions:

1. **Hierarchical Memory Design (L1/L2/L3)**: Inspired by computer architecture, we construct attention computation with distinct memory levels.
   - **L1 Memory (On-chip SRAM)**: Fast, high-bandwidth memory used for immediate attention computation and softmax normalization within small local blocks.
   - **L2 Memory (High Bandwidth Memory - HBM)**: Medium capacity memory storing intermediate activations, query-key-value representations, and cached softmax statistics.
   - **L3 Memory (Vectorized Hash-based DRAM Cache)**: High-capacity, slower memory with chunk-based recalls to manage historical attention blocks, leveraging vectorized hashing for efficient retrieval of attention patterns from past activations.
2. **Vectorized Hashing and Chunk-based Recall**: We integrate vectorized hashing mechanisms into the L3 memory to facilitate rapid and efficient chunk-based recall of attention patterns, enabling reuse of past computations and reducing redundant calculations.
3. **CUDA Kernel Implementation with Tiled α/z Support**: We provide detailed CUDA kernel implementations for our block attention operations, supporting tile-

Algorithm 2 Symbolic FastMKA: Lightweight Memory-Keyed Attention with Broadcast-Routing

Require: Input $X \in \mathbb{R}^{B\times T\times D}$; projection matrices W_q, W_k, W_v, W_o
Ensure: Output $O \in \mathbb{R}^{B\times T\times D}$; optional cache (K, V)
$Q \leftarrow XW_q$ {Query projection}
$Q_h \leftarrow \text{reshape}(Q) \rightarrow \mathbb{R}^{B\times H\times T\times d_h}$
{Hierarchical memory levels}
$L_1 \leftarrow X$ {L1: Local memory}
$L_2 \leftarrow \text{repeat}(\text{mean}(X, \text{dim} = 1), T)$ {L2: Session memory}
$L_3 \leftarrow \mathbf{0} \in \mathbb{R}^{B\times T\times D}$ {L3: Long-term memory}
$\lambda \leftarrow \text{softmax}(\text{MLP}(X)) \in \mathbb{R}^{B\times T\times 3}$
$X_{\text{routed}} \leftarrow \sum_{\ell=1}^{3} \lambda_\ell \cdot L_\ell$ {Broadcast-weighted fusion}
$K \leftarrow X_{\text{routed}} W_k, \quad V \leftarrow X_{\text{routed}} W_v$
$K_h \leftarrow \text{reshape}(K) \rightarrow \mathbb{R}^{B\times H\times T\times d_h}$
$V_h \leftarrow \text{reshape}(V) \rightarrow \mathbb{R}^{B\times H\times T\times d_h}$ {Update cache}
$A \leftarrow \text{softmax}(Q_h \cdot K_h^\top / \sqrt{d_h}) \cdot V_h$
$O \leftarrow \text{reshape}(A) \cdot W_o$
return $(O, (K_h, V_h))$

based normalization constants α and partition functions z, optimizing parallel computation on GPUs.

4.3. Hierarchical Block-wise MKA Algorithm

Standard attention is computed as follows:

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top, \quad \mathbf{P} = \text{softmax}(\mathbf{S}), \quad \mathbf{O} = \mathbf{P}\mathbf{V} \tag{1}$$

The quadratic complexity arises due to the computation of full $\mathbf{S} \in \mathbb{R}^{N\times N}$. To mitigate this, Block-MKA computes attention in blocks with intermediate memory management:

Step 1: Divide sequences into T blocks with dimension B, $T = N/B$:

$$\mathbf{Q} = [\mathbf{Q}_1; \ldots; \mathbf{Q}_T], \quad \mathbf{K} = [\mathbf{K}_1; \ldots; \mathbf{K}_T], \quad \mathbf{V} = [\mathbf{V}_1; \ldots; \mathbf{V}_T] \tag{2}$$

Step 2 (L1 and L2): Perform local softmax normalization with online α/z calculation within each block leveraging on-chip memory (L1) and HBM (L2):

$$\mathbf{S}_{ij} = \tau \mathbf{Q}_i \mathbf{K}_j^\top, \quad \mathbf{P}_{ij} = \frac{\exp(\mathbf{S}_{ij} - \alpha_{ij})}{z_{ij}}, \quad \mathbf{O}_i = \sum_j \mathbf{P}_{ij}\mathbf{V}_j \tag{3}$$

Normalization constants α_{ij} and z_{ij} are computed online as:

$$\alpha_{ij} = \max_k S_{ijk}, \quad z_{ij} = \sum_k \exp(S_{ijk} - \alpha_{ij}) \tag{4}$$

Step 3 (L3 - Hashing and Chunk Recall): Use vectorized hashing to retrieve similar past attention patterns from L3 memory. For each block, a chunk-based recall method efficiently retrieves stored historical attentions, reducing the overall computational complexity from quadratic to nearly linear.

4.4. Algorithm and Pseudocode

We provide detailed pseudocode illustrating hierarchical memory use:

Algorithm 3 Hierarchical Block-MKA Algorithm

Require: $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$; block size B; scaling factor τ
Ensure: Output $\mathbf{O} \in \mathbb{R}^{N\times d}$
Partition $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ into $\{\mathbf{Q}_i, \mathbf{K}_j, \mathbf{V}_j\}_{i,j=1}^{T}$, where $T = N/B$
for $i = 1$ to T **do**
  Load $\mathbf{Q}_i$ into L2 (HBM)
  Initialize $\mathbf{O}_i \leftarrow \mathbf{0}$, $z_i \leftarrow 0$, $m_i \leftarrow -\infty$
  for $j = 1$ to T **do**
    Load $\mathbf{K}_j, \mathbf{V}_j$ into L2 (HBM)
    $\mathbf{S}_{ij} \leftarrow \tau \cdot \mathbf{Q}_i \mathbf{K}_j^\top$ in L1 (SRAM)
    $m_i \leftarrow \max(m_i, \max(\mathbf{S}_{ij}))$
    $\tilde{\mathbf{S}}_{ij} \leftarrow \mathbf{S}_{ij} - m_i$
    $\alpha_{ij} \leftarrow \exp(\tilde{\mathbf{S}}_{ij})\mathbf{V}_j$
    $z_{ij} \leftarrow \sum \exp(\tilde{\mathbf{S}}_{ij})$
    Retrieve $\hat{\alpha}_{ij}, \hat{z}_{ij}$ via vectorized hash from L3 (DRAM)
    $\alpha_{ij} \leftarrow \alpha_{ij} + \hat{\alpha}_{ij}$, $z_{ij} \leftarrow z_{ij} + \hat{z}_{ij}$
    $\mathbf{O}_i \leftarrow \mathbf{O}_i + \alpha_{ij}$, $z_i \leftarrow z_i + z_{ij}$
  end for
  $\mathbf{O}_i \leftarrow \mathbf{O}_i / z_i$
end for
$\mathbf{O} \leftarrow [\mathbf{O}_1; \ldots; \mathbf{O}_T]$
return O

5. Theoretical Formulation: Recursive MKA with Online Softmax

We aim to derive a memory-efficient and numerically stable formulation of hierarchical attention:

$$\text{Attn}(Q) = \sum_{\ell=1}^{3} \lambda_\ell \cdot \text{Softmax}(QK_\ell^\top)V_\ell, \tag{5}$$

where $\lambda_\ell \geq 0$ and $\sum_\ell \lambda_\ell = 1$. Direct evaluation requires computing full attention maps, which is memory-intensive. We reformulate this as an online recursive computation that avoids storing attention weights explicitly.

5.1. Recursive Reformulation

Let $\alpha^{(0)} = 0$, $z^{(0)} = 0$, and define the recursive update:

$$\alpha^{(\ell)} = \alpha^{(\ell-1)} + \lambda_\ell \cdot \exp(QK_\ell^\top)V_\ell, \tag{6}$$

$$z^{(\ell)} = z^{(\ell-1)} + \lambda_\ell \cdot \exp(QK_\ell^\top), \tag{7}$$

for $\ell = 1, 2, 3$. We then define the final output as:

$$\text{Attn}(Q) := \frac{\alpha^{(3)}}{z^{(3)}}. \tag{8}$$

Theorem 5.1 (Equivalence of Recursive MKA and Standard Softmax Aggregation)**.** *Let $\lambda_\ell \geq 0$ with $\sum_{\ell=1}^3 \lambda_\ell = 1$. Then,*

$$\frac{\sum_{\ell=1}^3 \lambda_\ell \cdot \exp(QK_\ell^\top)V_\ell}{\sum_{\ell=1}^3 \lambda_\ell \cdot \exp(QK_\ell^\top)} = \sum_{\ell=1}^3 \lambda_\ell \cdot \textit{Softmax}(QK_\ell^\top)V_\ell.$$

Proof. Define $s_\ell = \exp(QK_\ell^\top) \in \mathbb{R}^{B\times T}$ and observe:

$$\text{LHS} = \frac{\sum_{\ell=1}^3 \lambda_\ell \cdot s_\ell V_\ell}{\sum_{\ell=1}^3 \lambda_\ell \cdot s_\ell}, \quad \text{RHS} = \sum_{\ell=1}^3 \lambda_\ell \cdot \left(\frac{s_\ell}{\sum s_\ell}\right) V_\ell.$$

Bringing both sides to a common denominator:

$$\sum_{\ell=1}^3 \lambda_\ell \cdot \left(\frac{s_\ell}{\sum_\ell \lambda_\ell s_\ell}\right) V_\ell = \frac{\sum_{\ell=1}^3 \lambda_\ell \cdot s_\ell V_\ell}{\sum_{\ell=1}^3 \lambda_\ell \cdot s_\ell}.$$

Hence, both expressions are mathematically equivalent. □

5.2. Numerical Stability via Max-Shift

To ensure stable computation of $\exp(QK_\ell^\top)$ across ℓ, we employ a hierarchical max-shift. Let $\mu^{(0)} = -\infty$, and define:

$$\mu^{(\ell)} = \max(\mu^{(\ell-1)}, \max(QK_\ell^\top)), \tag{9}$$

$$s_\ell = \exp(QK_\ell^\top - \mu^{(\ell)}), \tag{10}$$

$$z^{(\ell)} = z^{(\ell-1)} \cdot \exp(\mu^{(\ell-1)} - \mu^{(\ell)}) + \lambda_\ell s_\ell, \tag{11}$$

$$\alpha^{(\ell)} = \alpha^{(\ell-1)} \cdot \exp(\mu^{(\ell-1)} - \mu^{(\ell)}) + \lambda_\ell s_\ell V_\ell. \tag{12}$$

This mirrors FlashAttention's scan update trick and ensures stable operation in low-precision or long-context regimes.

5.3. Local vs. Global MKA Modes

We define two instantiations of recursive MKA:

- **Local-MKA:** uses only K_1, V_1 and K_2, V_2 from local and session memory. Computation is windowed and block-parallel, scaling as $\mathcal{O}(n)$.
- **Global-MKA:** includes long-term memory K_3, V_3 retrieved via hashing. Recursive scan with chunk recall yields amortized sublinear complexity.

5.4. Convergence Guarantee

Define:

$$a^{(\ell)} := \lambda_\ell \cdot \exp(QK_\ell^\top)V_\ell, \quad z^{(\ell)} := \lambda_\ell \cdot \exp(QK_\ell^\top).$$

Then:

$$\text{Attn}(Q) = \frac{\sum_{\ell=1}^3 a^{(\ell)}}{\sum_{\ell=1}^3 z^{(\ell)}} = \sum_{\ell=1}^3 \lambda_\ell \cdot \left(\frac{\exp(QK_\ell^\top)}{\sum_j \lambda_j \exp(QK_j^\top)}\right) V_\ell,$$

which matches the standard form and completes the equivalence proof. □

5.5. Runtime Bounds and Complexity

We now analyze the complexity of MKA with respect to memory levels:

- **L1 (Local Attention)**: Standard causal block attention over B tokens, cost $\mathcal{O}(B^2 d)$, computed from on-chip SRAM.
- **L2 (Session Memory)**: Pooled or low-rank block summaries over $T = N/B$ blocks, cost $\mathcal{O}(BTd)$.
- **L3 (Long-Term Memory)**: Vector-hash recall of $R \ll T$ past blocks, each of size B, with cost $\mathcal{O}(BRd)$.

Let N be total sequence length, B block size. Total runtime per step is:

$$\boxed{\mathcal{O}(BTd + BRd) \quad \text{with } R \ll T,} \tag{13}$$

which is subquadratic in N and superior to $\mathcal{O}(N^2 d)$ full attention. Memory access is minimized via tiled L1 computation and chunk-based L3 recall.

6. Experiments

6.1. Experimental Setup

Models. We start from the public **GPT-2 small** checkpoint[1] and replace its block attention with

(i) the original **Multi-Head Attention (MHA)**; (ii) our implementation of **Ropeless-MLA** (low-rank $Q/K/V$, no RoPE); (iii) the proposed **MKA** featuring three-level memory and a learnable routing gate.

Dataset. All experiments are conducted on **WikiText-2(raw)** (train=36718, valid=3760, test=4358 sentences). We use the GPT-2 BPE tokenizer (vocab 50257) and set `pad_token=eos_token`.

[1] 117M parameters, 12 layers, hidden size $d = 768$, $H = 12$ heads

Training Details. Each model is fine-tuned for **one epoch** only[2], with batch size 4 and sequence length 128. Optimizer: AdamW (lr= 5×10^{-5}, β=$(0.9, 0.999)$, no warm-up).

Metric. Language-model quality is measured by cross-entropy loss $\mathcal{L}$ and its exponential form, **Perplexity (PPL)**, PPL $= e^{\mathcal{L}}$ (lower is better).

6.2. Main Results

Table 1. Comparison of attention mechanisms on WikiText-2 after fine-tuning. FMKA achieves the best trade-off between accuracy and speed.

Method	PPL ↓	Train Time (s) ↓	Eval Time (s) ↓
MHA (baseline)	2.54	226.1	48.2
MQA	2.50	395.3	244.7
GQA	2.49	778.6	294.5
MLA	2.43	782.4	87.2
FMKA (ours)	**2.51**	**161.8**	**52.3**

Table 2. Training time (in seconds) per epoch under different sequence lengths. FastMKA consistently runs faster than MLA, MQA, and GQA.

Method	1K	2K	4K	8K	16K
MHA	226.1	472.5	945.3	1889.2	3774.3
MQA	395.3	731.4	1492.5	3002.0	6017.8
GQA	778.6	1462.7	2881.6	5773.9	11520.1
MLA	782.4	1341.6	2566.2	5032.5	10106.2
FMKA (ours)	**161.8**	**318.2**	**642.6**	**1269.3**	**2530.9**

6.3. Analysis

6.4. Analysis

- **Accuracy.** FastMKA achieves competitive perplexity (2.51) compared to Multi-Query Attention (MQA, 2.50), Grouped-Query Attention (GQA, 2.49), and Multi-Latent Attention (MLA, 2.43), while significantly outperforming the MHA baseline (2.54). This shows that our hierarchical routing mechanism captures contextual dependencies effectively, even with fewer key-value projections.

- **Training Efficiency.** FastMKA is up to **4× faster** than MLA and GQA, and **2.4× faster** than MQA when training with 16K sequences. It also beats MHA by a wide margin, thanks to its single-path broadcast routing and shared KV projection. This makes it well-suited for long-context fine-tuning under compute constraints.

- **Scalability.** As context length increases from 1K to 16K, MQA and GQA suffer from sharp training time growth due to head-specific KV reads and large memory footprints. In contrast, FastMKA scales linearly and maintains high throughput, making it as a scalable attention backend for long-context language models.

6.5. Discussion

MKA improves perplexity through dynamic routing over hierarchical memory levels—local (L1), session (L2), and long-term (L3)—without increasing the model's parameter count. This makes MKA attractive for memory-intensive inference settings where representation fidelity and cache reusability are critical.

However, the full MKA formulation requires repeated projection and attention over all memory levels, which can be computationally expensive. To address this, we propose **FastMKA**, a broadcast-routing variant that fuses the memory levels before attention is computed. FastMKA significantly reduces training and inference latency while preserving most of MKA's accuracy benefits, as validated by experiments on different sequence length on WikiText.

The lightweight design of FastMKA, with a single KV projection and soft token routing, makes it particularly suitable for high-throughput inference and edge deployment. It offers a practical balance between accuracy, latency, and memory usage, and serves as a strong drop-in replacement for traditional attention in long-context scenarios.

7. Conclusion

We introduce **Memory-Keyed Attention (MKA)**, a hierarchical attention mechanism that enables efficient long-context modeling by routing queries across multiple levels of memory. MKA improves perplexity while maintaining practical compute cost, making it a promising candidate for memory-aware transformer design.

To further reduce overhead, we propose **FastMKA**, a broadcast-routed variant that performs memory fusion before attention computation. FastMKA retains the architectural benefits of MKA while achieving significantly lower latency and training cost. FastMKA form a flexible framework for scalable and efficient LLM inference with long-sequence inputs.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. 2023. URL https://arxiv.org/abs/2305.13245.

[2]The goal is to compare convergence speed under equal compute; additional epochs further reduce PPL but do not change the ranking.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J.-B., Damoc, B., Clark, A., de Las Casas, D., Guy, A., Menick, J., Ring, R., Hennigan, T., Huang, S., Maggiore, L., Jones, C., Cassirer, A., Brock, A., Paganini, M., Irving, G., Vinyals, O., Osindero, S., Simonyan, K., Rae, J. W., Elsen, E., and Sifre, L. Improving language models by retrieving from trillions of tokens. 2022. URL https://arxiv.org/abs/2112.04426.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of ACL*, pp. 2978–2988, 2019.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. 2022. URL https://arxiv.org/abs/2205.14135.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. In *Journal of Machine Learning Research*, volume 23, pp. 1–39, 2022.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. 2024. URL https://arxiv.org/abs/2310.06839.

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. In *International Conference on Learning Representations (ICLR)*, 2021.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive nlp tasks. 2021. URL https://arxiv.org/abs/2005.11401.

Oren, M., Hassid, M., Yarden, N., Adi, Y., and Schwartz, R. Transformers are multi-state rnns. 2024. URL https://arxiv.org/abs/2401.06104.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models. 2023. URL https://arxiv.org/abs/2309.00071.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylKikSYDH.

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Shazeer, N. Fast transformer decoding: One write-head is all you need. 2019. URL https://arxiv.org/abs/1911.02150.

Shukor, M., Fini, E., da Costa, V. G. T., Cord, M., Susskind, J., and El-Nouby, A. Scaling laws for native multimodal models. 2025. URL https://arxiv.org/abs/2504.07951.

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. Quest: Query-aware sparsity for efficient long-context llm inference. 2024. URL https://arxiv.org/abs/2406.10774.

Team, D. Deepseek-v2 technical report. 2024.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models. 2023. URL https://arxiv.org/abs/2302.13971.

Tworkowski, S., Staniszewski, K., Pacek, M., Wu, Y., Michalewski, H., and Miłoś, P. Focused transformer: Contrastive training for context scaling. 2023. URL https://arxiv.org/abs/2307.03170.

Weston, J., Chopra, S., and Bordes, A. Memory networks. 2015. URL https://arxiv.org/abs/1410.3916.

A. Appendix

Algorithm 4 Multi-Level Memory-Keyed Attention (MKA)

Require: $X \in \mathbb{R}^{B\times S\times D}$, projection matrices W_q, W_k, W_v, W_o, routing MLP parameters; number of heads H, and head dimension $d_h = D/H$
Ensure: $O \in \mathbb{R}^{B\times S\times D}$; KV cache $present = (K, V)$

Compute Query:
$Q \leftarrow W_q \cdot X$
$Q_h \leftarrow \text{reshape}(Q, [B, S, H, d_h])$; transpose to (B, H, S, d_h)
Define Memory Sources:
$L_1 \leftarrow X$ {Local memory (current tokens)}
$L_2 \leftarrow \text{repeat}(\text{mean}(X,\ \text{dim} = 1), S)$ {Session memory (summary)}
$L_3 \leftarrow \mathbf{0} \in \mathbb{R}^{B\times S\times D}$ {Long-term memory}
Compute Routing Weights:
$\lambda \leftarrow \text{softmax}(\text{MLP}(X))$ {Shape: $B \times S \times 3$}
For each Memory Source $\ell = 1, 2, 3$**:**
for $\ell = 1$ to 3 **do**
  $K_\ell \leftarrow W_k \cdot L_\ell$
  $V_\ell \leftarrow W_v \cdot L_\ell$
  $K_{i,h} \leftarrow \text{reshape}(K_\ell, [B, S, H, d_h])$; transpose to (B, H, S, d_h)
  $V_{i,h} \leftarrow \text{reshape}(V_\ell, [B, S, H, d_h])$; transpose to (B, H, S, d_h)
  $A_\ell \leftarrow \text{Attention}(Q_h, K_{i,h}, V_{i,h})$ {Scaled dot-product attention}
end for
Aggregate Attention Outputs:
$A \leftarrow \sum_{\ell=1}^{3} \lambda[..., \ell] \odot A_\ell$
Reshape A back to (B, S, D)
Final Projection:
$O \leftarrow W_o \cdot A$
Update KV Cache:
$K_{\text{new}} \leftarrow \text{reshape}(W_k \cdot X, [B, S, H, d_h])$; transpose to (B, H, S, d_h)
$V_{\text{new}} \leftarrow \text{reshape}(W_v \cdot X, [B, S, H, d_h])$; transpose similarly
if previous cache $present = (K_{\text{prev}}, V_{\text{prev}})$ exists **then**
  $K \leftarrow \text{concat}(K_{\text{prev}}, K_{\text{new}}, \text{dim} = 3)$
  $V \leftarrow \text{concat}(V_{\text{prev}}, V_{\text{new}}, \text{dim} = 3)$
else
  $K \leftarrow K_{\text{new}}$, $V \leftarrow V_{\text{new}}$
end if
$present \leftarrow (K, V)$
return $(O, present)$

Algorithm 5 Broadcast-Routed Memory-Keyed Attention (FastMKA)

Require: $X \in \mathbb{R}^{B\times S\times D}$, projection matrices W_q, W_k, W_v, W_o, routing MLP parameters; number of heads H, and head dimension $d_h = D/H$
Ensure: $O \in \mathbb{R}^{B\times S\times D}$; KV cache $present = (K, V)$

Compute Query:
$Q \leftarrow W_q \cdot X$
$Q_h \leftarrow \text{reshape}(Q, [B, S, H, d_h])$; transpose to (B, H, S, d_h)
Define Memory Sources:
$L_1 \leftarrow X$ {L1: Local memory}
$L_2 \leftarrow \text{repeat}(\text{mean}(X,\ \text{dim} = 1), S)$ {L2: Session memory}
$L_3 \leftarrow \mathbf{0} \in \mathbb{R}^{B\times S\times D}$ {L3: Long-term memory}
Compute Routing Weights:
$\lambda \leftarrow \text{softmax}(\text{MLP}(X))$ {Shape: $B \times S \times 3$}
Fuse Memory via Broadcast Routing:
$X_{\text{routed}} \leftarrow \sum_{\ell=1}^{3} \lambda[..., \ell] \cdot L_\ell$ {Fused memory token-wise}
Compute Key/Value from Fused Memory:
$K \leftarrow W_k \cdot X_{\text{routed}}$
$V \leftarrow W_v \cdot X_{\text{routed}}$
$K_h \leftarrow \text{reshape}(K, [B, S, H, d_h])$; transpose to (B, H, S, d_h)
$V_h \leftarrow \text{reshape}(V, [B, S, H, d_h])$; transpose to (B, H, S, d_h)
Compute Attention:
$A \leftarrow \text{Attention}(Q_h, K_h, V_h)$
Final Projection:
$O \leftarrow W_o \cdot \text{reshape}(A, [B, S, D])$
Update KV Cache:
$K_{\text{new}} \leftarrow K_h$, $V_{\text{new}} \leftarrow V_h$
if previous cache $present = (K_{\text{prev}}, V_{\text{prev}})$ exists **then**
  $K \leftarrow \text{concat}(K_{\text{prev}}, K_{\text{new}}, \text{dim} = 2)$
  $V \leftarrow \text{concat}(V_{\text{prev}}, V_{\text{new}}, \text{dim} = 2)$
else
  $K \leftarrow K_{\text{new}}$, $V \leftarrow V_{\text{new}}$
end if
$present \leftarrow (K, V)$
return $(O, present)$