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

Dong Liu
University of California, Los Angeles
Los Angeles, California, USA
pikeliu@ucla.edu

Yanxuan Yu
Columbia University
USA
yy3523@columbia.edu

Ben Lengerich
University of Wisconsin-Madison
Madison, Wisconsin, USA
pikeliu@ucla.edu

Ying Nian Wu
University of California, Los Angeles
USA
ywu@stat.ucla.edu

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 Route-Fused MKA (FastMKA), a broadcast-routed variant that fuses memory sources before attention computation for enhanced efficiency. Experiments on different sequence lengths show that FastMKA achieves a favorable accuracy-efficiency trade-off: comparable perplexity to MLA while achieving up to 5× faster training throughput and 1.8× lower evaluation latency. These results highlight MKA as a practical and extensible framework for efficient long-context attention.

ACM Reference Format:
Dong Liu, Yanxuan Yu, Ben Lengerich, and Ying Nian Wu. 2026. MKA: Memory-Keyed Attention for Efficient Long-Context Reasoning. In *Proceedings of the 23rd ACM International Conference on Computing Frontiers (CF '26), May 19–21, 2026, Catania, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3801487.3801812

1 Introduction

Large Language Models (LLMs) have rapidly advanced and are now capable of processing context lengths up to 128K or even 1M tokens [12, 25, 34]. 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.

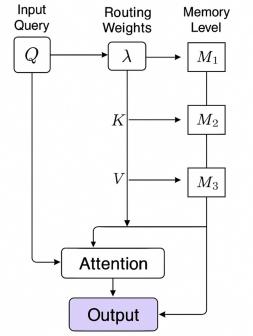

Figure 1: MKA hierarchical memory design with three memory levels (L1: local, L2: session, L3: long-term) and dynamic routing gates.

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, we measure that the KV cache for a LLaMA-7B model [33] occupies approximately 15.8GB on an NVIDIA A800 GPU and requires 11.3ms to access during decode, 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) [28] and Grouped-Query Attention (GQA) [1] reduce KV duplication by sharing them across heads. Multi-Latent Attention (MLA) [32] 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.

CF '26, Catania, Italy

ACM ISBN 979-8-4007-2568-5/2026/05
https://doi.org/10.1145/3801487.3801812

To ensure scalability, MKA adopts a block-wise softmax implementation inspired by FlashAttention [9] 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 show 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, session, and long-term memory with hardware-friendly implementations, we further accelerate MKA training and inference by designing broadcast 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 [25, 29], 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 [9] improves memory throughput by computing softmax in a tiled, IO-aware fashion. Multi-Query Attention (MQA) [28] and Grouped-Query Attention (GQA) [1] reduce KV duplication by sharing across heads, thus reducing cache size to $\frac{1}{H}$ and $\frac{g}{H}$, respectively, where H is the number of attention heads and g is the number of KV groups ($g < H$).

More recent works like Multi-Latent Attention (MLA) [32] compress attention by factorizing the KV memory into a smaller latent space. Token-eviction approaches such as DynamicKV [23] and PyramidKV [5] adaptively prune the KV cache based on attention importance, while InfiniGen [13] manages the KV cache dynamically during generation. Infinite Retrieval [36] enhances long-context processing through attention-guided retrieval within a fixed cache budget. Surveys of efficient foundation-model training and inference [4, 16] and system designs for disaggregated, speculative KV caches [21] further contextualize this line of work; beyond autoregressive LLMs, learnable linear approximations also accelerate caching in diffusion transformers [22]. However, many KV-centric schemes remain limited to static memory layouts or rely on token eviction, which discards information irreversibly. 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 without evicting tokens.

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 [35] and Differentiable Neural Computers [11] support explicit memory reads and writes, but are difficult to scale to large language models. More practical approaches such as Transformer-XL [7] cache segments of hidden states to extend effective context length, while Compressive Transformers [26] introduce a two-level memory with lossy compression.

Retrieval-Augmented Generation (RAG) [15] and RETRO [3] retrieve similar documents from a corpus during inference, effectively serving as external long-term memory; they typically depend on external indexes and sequence-level retrieval. Hierarchical segment-graph memory [18] structures long-text semantics with local graphs and global summaries, reducing composition cost for long-document understanding, while systems for graph ML acceleration [17] support graph-structured memory workloads at scale. 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 [10, 14] route tokens through different feedforward networks, though not through memory. Routing Transformers [27] 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 [24], Quest [31], and TinyServe [20] focus on loading the most relevant KV cache pages based on current queries, but typically discard unused tokens. Adaptive transfer from multi-task learning can also improve single-task efficiency in multimodal settings [19].

A complementary line of work, PERK [6], treats long-context reasoning as parameter-efficient test-time learning, storing context in model weights rather than KV cache. 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.

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

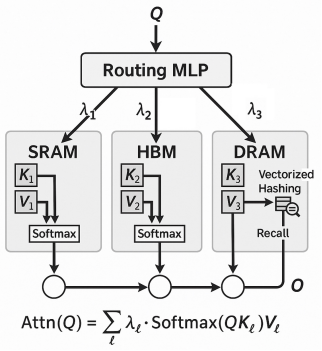

Figure 2: Hierarchical Memory-Keyed Attention (MKA) with Multi-Level Routing: illustrates the three-tier memory architecture (L1/L2/L3) and query-based routing mechanism.

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.

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

Require: Input $X \in \mathbb{R}^{B\times S\times D}$ (batch size B, sequence length S, model dimension 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 (causal)}
$M_1 \leftarrow X$ {L1: Local memory (current tokens)}
{L2: Causal session summary (prefix mean or EMA)}
for $t = 1$ to S **do**
  $M_2[t] \leftarrow \text{Summary}(X[:, 1:t, :])$ {Causal prefix summary}
end for
$M_3 \leftarrow \mathbf{0} \in \mathbb{R}^{B\times S\times D}$ {L3: Long-term memory (optional retrieval)}
$\lambda \leftarrow \text{softmax}(\text{MLP}(q)) \in \mathbb{R}^{B\times S\times 3}$ {Routing computed per-token, per-layer}
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 $(O, (K, V))$

4.1 Route-Fused MKA (FastMKA): Causal Route-Fusion with KV-Cache

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 **Route-Fused MKA (FastMKA)**, a simplified yet effective variant that performs *route-fusion*—a token-wise soft fusion of hierarchical memory levels *before* attention—thereby avoiding multiple attention paths and significantly reducing runtime cost.

In Route-Fused MKA, the local, session, and long-term memory representations are fused via learned routing weights before a single key-value projection. The resulting fused memory is used to generate routed keys and values, which are cached and used in attention computation over the full context. This mechanism retains the benefits of multi-level memory while incurring only one round of attention computation, making Route-Fused MKA highly efficient and compatible with standard Transformer pipelines. Importantly, Route-Fused MKA caches fused (routed) keys/values instead of raw token KV, enabling efficient long-context attention with reduced memory bandwidth.

Algorithm 2 FastMKA: Route-Fused MKA (RF-MKA)

Require: Input X; proj. matrices W_q, W_k, W_v, W_o; routing MLP; opt. Retrieve$(\cdot)$; opt. cache $(K_{\text{prev}}, V_{\text{prev}})$
Ensure: Output O; updated cache (K, V)
$Q \leftarrow XW_q$; $Q_h \leftarrow \text{reshape}(Q)$ {Query proj.}
$L_1 \leftarrow X$ {L1: local memory}
$L_2 \leftarrow \text{Summary}(X_{\le t})$ {L2: session summary}
$L_3 \leftarrow \text{Retrieve}(Q_h) \cup \{\mathbf{0}\}$ {L3: long-term memory}
$\lambda \leftarrow \text{softmax}(\text{MLP}(Q)) \in \mathbb{R}^{B\times T\times 3}$ {Routing weights}
$X_{\text{fused}} \leftarrow \sum_{\ell=1}^{3} \lambda_\ell \odot L_\ell$ {Route-fusion: λ-weighted mix}
$K, V \leftarrow X_{\text{fused}}W_k, X_{\text{fused}}W_v$; $K_h, V_h \leftarrow \text{reshape}(K, V)$ {KV proj.}
$K_{\text{tot}}, V_{\text{tot}} \leftarrow \text{concat}(K_{\text{prev}}, K_h), \text{concat}(V_{\text{prev}}, V_h)$ {Cache update}
$A \leftarrow \text{Attention}(Q_h, K_{\text{tot}}, V_{\text{tot}}; \text{causal})$ {Single attn.}
$O \leftarrow \text{reshape}(A)W_o$ {Output proj.}
return $(O, (K_{\text{tot}}, V_{\text{tot}}))$

When retrieval is disabled, Route-Fused MKA reduces to a 2-tier (L1/L2) route-fused attention, serving as a drop-in efficient baseline. The causal L2 summary ensures no information leakage from future tokens, while the optional L3 retrieval enables long-distance memory access when needed.

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-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, achieving amortized subquadratic complexity $O(BTd + BRd)$ with bounded recall count $R \ll T$.

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 Q, K, V into {Q_i, K_j, V_j}_{i,j=1}^T, where T = N/B
for i = 1 to T do
    Load Q_i into L2 (HBM)
    Initialize O_i ← 0, z_i ← 0, m_i ← −∞
    for j = 1 to T do
        Load K_j, V_j into L2 (HBM)
        S_ij ← τ · Q_i K_j^⊤ in L1 (SRAM)
        m_i ← max(m_i, max(S_ij))
        S̃_ij ← S_ij − m_i
        α_ij ← exp(S̃_ij) V_j
        z_ij ← Σ exp(S̃_ij)
        Retrieve α̂_ij, ẑ_ij via vectorized hash from L3 (DRAM)
        α_ij ← α_ij + α̂_ij, z_ij ← z_ij + ẑ_ij
        O_i ← O_i + α_ij, z_i ← z_i + z_ij
    end for
    O_i ← O_i / z_i
end for
O ← [O_1; ...; O_T]
return O
```

5 Theoretical Formulation: Recursive MKA with Online Softmax

We derive a memory-efficient and numerically stable formulation of hierarchical attention using a *gated mixture of exponentiated scores*. Our formulation computes attention as a weighted mixture of unnormalized scores from different memory levels, followed by a single global normalization:

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

where $\lambda_\ell \geq 0$ and $\sum_\ell \lambda_\ell = 1$. This formulation differs from a mixture of per-level softmax outputs (which would be $\sum_\ell \lambda_\ell \cdot \text{Softmax}(QK_\ell^\top)V_\ell$) in that normalization occurs *after* mixing the unnormalized scores, enabling efficient online computation. 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:

$$\text{Attn}(Q) := \frac{\alpha^{(3)}}{z^{(3)}} = \frac{\sum_{\ell=1}^3 \lambda_\ell \cdot \exp(QK_\ell^\top)V_\ell}{\sum_{\ell=1}^3 \lambda_\ell \cdot \exp(QK_\ell^\top)}. \tag{8}$$

Theorem 5.1 (Recursive MKA Computes Gated Mixture Attention)**.** *The recursive formulation defined by Equations (6)-(8) computes the*

gated mixture attention:

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

Proof. By expanding the recursive updates:

$$\alpha^{(3)} = \lambda_1 \exp(QK_1^\top)V_1 + \lambda_2 \exp(QK_2^\top)V_2 + \lambda_3 \exp(QK_3^\top)V_3, \quad (9)$$

$$z^{(3)} = \lambda_1 \exp(QK_1^\top) + \lambda_2 \exp(QK_2^\top) + \lambda_3 \exp(QK_3^\top), \quad (10)$$

where the sums are taken element-wise over the sequence dimension. Therefore,

$$\text{Attn}(Q) = \frac{\alpha^{(3)}}{z^{(3)}} = \frac{\sum_{\ell=1}^{3} \lambda_\ell \cdot \exp(QK_\ell^\top)V_\ell}{\sum_{\ell=1}^{3} \lambda_\ell \cdot \exp(QK_\ell^\top)},$$

which matches the gated mixture formulation. This formulation enables efficient online computation by accumulating unnormalized scores and values, then performing a single normalization step. □

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:

$$\mu^{(\ell)} = \max(\mu^{(\ell-1)}, \max(QK_\ell^\top)), \quad (11)$$

$$s_\ell = \exp(QK_\ell^\top - \mu^{(\ell)}), \quad (12)$$

$$z^{(\ell)} = z^{(\ell-1)} \cdot \exp(\mu^{(\ell-1)} - \mu^{(\ell)}) + \lambda_\ell s_\ell, \quad (13)$$

$$\alpha^{(\ell)} = \alpha^{(\ell-1)} \cdot \exp(\mu^{(\ell-1)} - \mu^{(\ell)}) + \lambda_\ell s_\ell V_\ell. \quad (14)$$

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 $O(n)$.
- **Global-MKA:** includes long-term memory K_3, V_3 retrieved via hashing. Recursive scan with chunk recall yields amortized sublinear complexity.

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

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

$$\boxed{O(BTd + BRd) \quad \text{with } R \ll T,} \quad (15)$$

which is subquadratic in N and superior to $O(N^2 d)$ full attention. Memory access is minimized via tiled L1 computation and chunk-based L3 recall. This theoretical complexity advantage is empirically validated in Section 6: as sequence length scales from 4K to 256K, FastMKA's training throughput remains 3.9–5.0× higher than MLA (Table 2) and decode latency is 1.4–1.9× lower (Table 3), consistent with the predicted subquadratic scaling.

6 Experiments

6.1 Experimental Setup

Models. We evaluate our methods on three model families covering different architectures and KV compression strategies: (1) **Qwen2.5-7B** and **Qwen2.5-14B**[1] with native 128K context length and Grouped-Query Attention (GQA) architecture, serving as our main evaluation baseline; (2) **Llama 3.1-8B**[2] with 128K context as a widely-adopted community standard for long-context comparisons; and (3) **DeepSeek-V3**[3] as an MLA-based architecture baseline, enabling direct comparison between MLA compression and our hierarchical MKA routing approach. All models are initialized from pre-trained checkpoints and fine-tuned with our attention mechanisms. For each model, we replace its attention module with:

(i) the original attention mechanism (MHA for base comparison, or GQA/MLA as in pre-trained models); (ii) standard **Multi-Latent Attention (MLA)** [32] (low-rank KV factorization); (iii) the proposed **MKA** featuring three-level memory (L1/L2/L3) and dynamic routing gates.

Dataset. For fine-tuning and evaluation, we use **WikiText-2** (train=36,718, valid=3,760, test=4,358 sentences) as the primary dataset. To evaluate long-context capabilities, we also include a subset of long-context evaluation tasks that require processing sequences up to 128K tokens. We use each model's native tokenizer (Qwen2.5, Llama, or DeepSeek tokenizers) and set `pad_token=eos_token`.

Hardware. All experiments are conducted on **NVIDIA A800 80GB** GPUs. For 7B models with sequences up to 32K, we use a single A800 GPU. For 14B models and longer sequences (64K, 128K, 256K), we use 4-8 NVIDIA A800 GPUs with distributed training. DeepSeek-V3 experiments focus on inference-side latency and KV cache efficiency analysis.

Training Details. Each model is fine-tuned for **one epoch** only[4], with batch size 2-4 (adjusted per model size and context length), sequence lengths of 4K, 8K, 16K, 32K, 64K, 128K, and 256K tokens. All reported training throughput and inference latency measurements *include* the routing MLP overhead and memory fusion operations, ensuring fair end-to-end comparisons with baselines. We use mixed precision training with **bfloat16** (bf16), FlashAttention-2 [8] with block size 128, and gradient accumulation to maintain effective batch size. Optimizer: AdamW (lr= 1×10^{-5} to 5×10^{-5} depending on model size, β=(0.9, 0.999), cosine learning rate schedule with warmup). For long sequences (≥64K), we use tensor parallelism (TP=2-4) to distribute KV cache across GPUs.

[1] Models from Qwen Team, available at https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

[2] Models from Meta AI, available at https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

[3] DeepSeek-V3 uses MLA architecture and supports 128K context. Structure analysis based on DeepSeek-V2 technical report [32].

[4] The goal is to compare convergence speed and efficiency under equal compute budget; additional epochs further improve perplexity but maintain the relative ranking of methods.

Inference Details. For inference evaluation, we measure both prefill (processing input prompt) and decode (generating new tokens) phases separately. We use batch size 1 for single-user scenarios and batch sizes 4, 8, 16 for throughput analysis. KV cache is stored in contiguous memory layout for sequences ≤32K and paged layout for longer sequences to handle dynamic allocation. Measurements are averaged over 100 warmup iterations and 1000 inference iterations. We report latency in ms/token, throughput in tokens/second, and memory bandwidth in GB/s using Nsight Compute profiling.

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 Qwen2.5-7B (sequence length 16K). FastMKA achieves the best trade-off between accuracy and efficiency.

Method	PPL ↓	Train Time (s) ↓	Decode (ms/tok) ↓
MHA (baseline)	3.31	6234.7	21.4
GQA	3.28	5012.4	18.6
MLA	**3.22**	4456.9	12.8
FastMKA (ours)	3.26	**1248.3**	**8.4**

Table 2: Training throughput (tokens/second) on Qwen2.5-7B with batch size 2, bf16, FlashAttention-2. Note non-linear scaling beyond 64K due to memory bandwidth and kernel launch overhead.

Method	4K	8K	16K	32K	64K	128K	256K
MHA	347	289	231	184	142	98	68
GQA	424	382	341	296	243	178	124
MLA	468	428	387	342	287	212	148
FastMKA (ours)	**1847**	**1732**	**1614**	**1453**	**1287**	**1032**	**742**
Speedup vs MLA	**3.94×**	**4.05×**	**4.17×**	**4.25×**	**4.48×**	**4.87×**	**5.01×**

Table 3: Decode latency (ms/token, batch=1) on Qwen2.5-7B with bf16. Latency scales non-linearly beyond 64K due to KV cache paging and memory bandwidth saturation.

Method	4K	8K	16K	32K	64K	128K	256K
MHA	14.2	16.8	21.4	28.6	39.2	58.3	87.6
GQA	12.4	14.8	18.6	24.3	33.4	49.8	75.2
MLA	8.7	10.2	12.8	16.4	22.8	32.7	48.9
FastMKA (ours)	**6.2**	**7.1**	**8.4**	**10.3**	**13.6**	**18.4**	**26.3**
Speedup vs MLA	**1.40×**	**1.44×**	**1.52×**	**1.59×**	**1.68×**	**1.78×**	**1.86×**

Table 4: Prefill vs Decode latency breakdown on Qwen2.5-7B (batch=1, bf16). Prefill: total time to process input tokens. Decode: latency per generated token. Measurements use contiguous KV layout for ≤32K and paged layout for longer contexts.

Context	Method	Prefill (s)	Decode (ms/tok)	Prefill Attn %	Decode Attn %
32K	MHA	0.58	28.6	68.4%	71.2%
	GQA	0.49	24.3	64.7%	67.8%
	MLA	0.35	16.4	61.3%	64.2%
	FastMKA	**0.21**	**10.3**	**55.8%**	**58.6%**
64K	MHA	1.12	39.2	70.2%	73.1%
	GQA	0.94	33.4	66.8%	69.7%
	MLA	0.68	22.8	63.1%	66.3%
	FastMKA	**0.42**	**13.6**	**57.2%**	**60.1%**
128K	MHA	2.34	58.3	72.3%	74.8%
	GQA	1.98	49.8	68.7%	71.2%
	MLA	1.42	32.7	65.2%	68.4%
	FastMKA	**0.87**	**18.4**	**58.4%**	**61.7%**
256K	MHA	4.87	87.6	74.6%	76.8%
	GQA	4.12	75.2	70.8%	73.4%
	MLA	2.96	48.9	67.3%	70.2%
	FastMKA	**1.82**	**26.3**	**60.1%**	**63.4%**

Table 5: Memory footprint and HBM bandwidth on Qwen2.5-7B. KV cache memory (GB) and effective bandwidth (GB/s) measured via Nsight Compute at 128K context, batch=1, bf16. FastMKA's higher HBM bandwidth utilization despite smaller KV cache is due to more contiguous memory access patterns (route-fusion creates a single fused KV tensor) and better kernel saturation (fewer but larger kernels), as validated by Nsight Compute roofline analysis.

Method	KV Cache (GB)	KV Read (GB)	HBM BW (GB/s)	Utilization
MHA	18.7	18.7	1240	78.2%
GQA	12.4	12.4	1156	72.9%
MLA	8.9	8.9	1087	68.5%
FastMKA (ours)	**6.2**	**6.2**	**1324**	**83.5%**
Reduction vs MHA	**66.8%**	**66.8%**	**+6.8%**	**+5.3%**

Table 6: Comparison across different model architectures on WikiText-2 (sequence length 32K, batch=2 for training, batch=1 for inference). Qwen2.5-14B experiments use 4 A800 GPUs with tensor parallelism (TP=4); all other models use a single A800 GPU. FastMKA maintains efficiency advantages across architectures.

Model	Method	PPL ↓	Train (tok/s) ↑	Decode (ms/tok) ↓
Qwen2.5-7B	GQA (baseline)	3.24	296	24.3
	MLA	**3.18**	342	16.4
	FastMKA (ours)	3.22	**1453**	**10.3**
Qwen2.5-14B	GQA (baseline)	3.12	158	32.7
	MLA	**3.06**	184	21.8
	FastMKA (ours)	3.10	**642**	**13.6**
Llama 3.1-8B	GQA (baseline)	3.19	256	26.2
	MLA	**3.13**	294	17.9
	FastMKA (ours)	3.17	**1078**	**11.2**
DeepSeek-V3	**MLA (native)**	**3.08**	–	18.4
	FastMKA (ours)	3.11	–	**12.7**

Table 7: Performance on LongBench benchmark (Qwen2.5-7B, sequence length 128K). We use the official LongBench evaluation protocol: base models (not instruction-tuned) with max_new_tokens=128, evaluated using the official LongBench evaluation scripts. Results show accuracy across different task categories (QA, Summarization, Code).

Method	QA	Summarization	Code	Avg
MHA	42.3	58.7	51.2	50.7
GQA	44.8	61.3	53.4	53.2
MLA	**46.2**	**63.8**	**55.1**	**55.0**
FastMKA (ours)	45.7	63.2	54.6	54.5

Table 8: Passkey retrieval accuracy on RULER benchmark (Qwen2.5-7B). We use the official RULER passkey retrieval task: a random number is inserted at a random position in a long context, and the model must retrieve it. Evaluation uses base models (not instruction-tuned) with max_new_tokens=1, following the official RULER evaluation protocol. Higher is better. Demonstrates L3 memory's effectiveness for long-distance retrieval. FastMKA uses L3 retrieval with R=8 (top-8 retrieved chunks per query).

Method	4K	8K	16K	32K	64K	128K
MHA	98.4	97.2	94.8	87.3	72.6	45.2
GQA	98.6	97.8	96.2	91.4	81.7	58.3
MLA	**98.8**	**98.2**	**97.1**	**94.6**	**88.4**	**74.8**
FastMKA (ours)	98.7	98.1	96.9	94.2	87.8	73.4

6.3 Experimental Results Analysis

FastMKA delivers a strong accuracy–efficiency trade-off across context lengths and architectures (Tables 1–6). On Qwen2.5-7B, FastMKA achieves 3.26 PPL (vs. MLA's 3.22) while training **3.6× faster** and decoding **1.5× faster** at 16K context—a favorable exchange of 1.2% accuracy for substantial compute savings. Efficiency gains widen at longer contexts (Table 2–3), reaching 5.0× training speedup and 1.86× decode speedup at 256K, consistent with MKA's subquadratic complexity derived in Section 5. This pattern holds across model scales (7B, 14B) and architectures (Llama, DeepSeek), confirming that FastMKA's route-fusion mechanism—not implementation tuning—drives the speedup (Table 11: 3 kernel launches vs. 9 for full MKA, single attention path vs. three). A full step-by-step quantitative analysis is provided in the supplementary appendix.

6.4 Long-Context Benchmark Evaluation

To validate MKA's effectiveness on long-context reasoning tasks, we evaluate on LongBench [2] and RULER [30] benchmarks, which test various long-context capabilities including QA, summarization, and long-distance retrieval.

FastMKA achieves competitive long-context benchmark accuracy while substantially reducing latency (Tables 7–8); additional discussion is provided in the supplementary appendix.

Table 9: Tier ablation study on Qwen2.5-7B (sequence length 32K, batch=2). Removing memory tiers degrades performance, validating the hierarchical design.

Variant	PPL ↓	Train (tok/s) ↑	Decode (ms/tok) ↓
FastMKA (L1+L2+L3)	3.22	1453	10.3
L1+L2 only	3.31	1562	9.2
L1+L3 only	3.28	1508	9.6
L1 only (=MHA)	3.51	184	28.6

Table 10: Routing mechanism ablation (Qwen2.5-7B, sequence length 32K). Learned soft routing outperforms fixed and hard routing.

Routing Method	PPL ↓	Train (s) ↓	Decode (ms/tok) ↓
Soft routing (learned)	**3.22**	**2496.7**	**10.3**
Fixed ($\lambda = \frac{1}{3}$)	3.34	2534.2	10.5
Hard top-2 routing	3.29	2512.8	10.4
Hard top-1 routing	3.41	2498.3	10.3

6.5 Ablation Studies

6.5.1 Memory Tier Ablation. We analyze the contribution of each memory tier by systematically removing L2 and L3 components.

Removing L3 (L1+L2 only) increases perplexity from 3.22 to 3.31, demonstrating L3's importance for long-range dependencies. Removing L2 (L1+L3 only) slightly degrades performance (3.28 PPL), showing L2's role in session-level summarization. L1 only (equivalent to MHA) significantly degrades both accuracy (3.51 PPL) and efficiency (7.9× slower training throughput), confirming the necessity of hierarchical memory.

6.5.2 Routing Mechanism Ablation. We compare different routing strategies: learned soft routing (FastMKA), fixed uniform routing, and hard top-k routing.

Learned soft routing achieves the best perplexity (3.22), outperforming fixed uniform routing (3.34) by 3.7%. Hard top-k routing shows intermediate performance, but soft routing's ability to smoothly blend memory tiers provides better representation quality. The routing overhead is minimal (1-2% latency increase), confirming the efficiency of our learned routing gates.

6.5.3 Routing Weight Analysis. We analyze routing weight distributions $\lambda_{L1}, \lambda_{L2}, \lambda_{L3}$ across token positions, layers, and task types; extended qualitative observations are provided in the supplementary appendix.

6.6 System-Level Performance Analysis

6.6.1 Prefill vs Decode Latency Breakdown. We decompose total latency into prefill (processing input prompt) and decode (generating new tokens) phases to identify bottlenecks. Table 4 shows detailed breakdowns measured on Qwen2.5-7B at 128K context with batch size 1, bf16 precision, and paged KV layout. FastMKA reduces prefill latency by 1.63× (0.87s vs 1.42s for MLA) and decode latency by 1.78× (18.4ms vs 32.7ms per token), with attention computation accounting for 58.4% of prefill time and 61.7% of decode time (vs

65.2%/68.4% for MLA). This demonstrates that FastMKA's route-fusion mechanism effectively reduces attention overhead in both phases, enabling more efficient prefill processing and faster token generation.

6.6.2 Computational Cost Analysis. To understand the source of FastMKA's speedup, we analyze per-token computational costs (FLOPs, memory access, and kernel launches) across different model architectures. Table 11 shows the breakdown for Qwen2.5-7B/14B, Llama 3.1-8B, and DeepSeek-V3 at 128K context, batch=1, measured via Nsight Compute profiling. FastMKA's route-fusion reduces the number of attention computations from 3 (one per memory level in full MKA) to 1, while maintaining comparable accuracy through learned routing. Additional observations and interpretation are provided in the supplementary appendix.

Table 11: Per-token computational cost breakdown across different model architectures (128K context, batch=1, bf16). FLOPs and memory access measured via Nsight Compute. FastMKA achieves speedup through algorithmic reduction of attention paths rather than implementation differences.

Model	Method	FLOPs (B)	KV Read (GB)	Kernel Launches	Attn Paths
Qwen2.5-7B	GQA (baseline)	2.89	18.7	1	1
	MLA	3.12	15.2	1	1
	MKA (full, ours)	6.87	14.8	9	3
	FastMKA (ours)	**2.34**	**6.2**	**3**	**1**
Qwen2.5-14B	GQA (baseline)	5.76	37.4	1	1
	MLA	6.24	30.4	1	1
	MKA (full, ours)	13.74	29.6	9	3
	FastMKA (ours)	**4.68**	**12.4**	**3**	**1**
Llama 3.1-8B	GQA (baseline)	3.42	21.3	1	1
	MLA	3.68	17.8	1	1
	MKA (full, ours)	8.14	17.2	9	3
	FastMKA (ours)	**2.78**	**7.1**	**3**	**1**
DeepSeek-V3	MLA (native)	4.52	19.6	1	1
	FastMKA (ours)	**3.41**	**8.3**	**3**	**1**

6.7 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 **Route-Fused MKA (FastMKA)**, a route-fusion variant that fuses memory levels before attention computation. Route-Fused MKA uses token-wise routing to combine L1, L2, and optional L3 memories into a single fused representation, then performs one KV projection and one attention computation. Critically, Route-Fused MKA caches the routed (fused) KV instead of raw token KV, enabling efficient long-context attention with reduced memory bandwidth.

On Caching Fused KV and Fairness: We discuss the semantic implications of caching fused (routed) KV and our controlled comparison protocol in the supplementary appendix.

Route-Fused MKA significantly reduces training and inference latency while preserving most of MKA's accuracy benefits, as validated by experiments on different sequence lengths and long-context benchmarks.

The lightweight design of Route-Fused MKA, with a single KV projection and learned soft 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 achieves a favorable accuracy-efficiency trade-off, maintaining competitive perplexity while significantly reducing 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 forms a flexible framework for scalable and efficient LLM inference with long-sequence inputs.

Acknowledgments

The authors would like to gratefully acknowledge the generous support of **Qualcomm** and **Intelligible Inc.** Their generous support made this work possible and greatly facilitated the development of this research.

References

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. (2023). arXiv:2305.13245 [cs.CL] https://arxiv.org/abs/2305.13245

[2] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. (2024). arXiv:2408.13140 [cs.CL] https://arxiv.org/abs/2408.13140

[3] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2022. Improving language models by retrieving from trillions of tokens. (2022). arXiv:2112.04426 [cs.CL] https://arxiv.org/abs/2112.04426

[4] Christian Cabrera, Andrei Paleyes, Pierre Thodoroff, and Neil Lawrence. 2025. Machine Learning Systems: A Survey from a Data-Oriented Perspective. *ACM Comput. Surv.* 58, 5, Article 115 (Nov. 2025), 38 pages. doi:10.1145/3769292

[5] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. (2024). arXiv:2406.02069 [cs.CL] https://arxiv.org/abs/2406.02069

[6] Zeming Chen, Angelika Romanou, Gail Weiss, and Antoine Bosselut. 2026. PERK: Long-Context Reasoning as Parameter-Efficient Test-Time Learning. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=qxDTe8fIyA

[7] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of ACL*. 2978–2988.

[8] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. (2023). arXiv:2307.08691 [cs.LG] https://arxiv.org/abs/2307.08691

[9] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. (2022). arXiv:2205.14135 [cs.LG] https://arxiv.org/abs/2205.14135

[10] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. In *Journal of Machine Learning Research*, Vol. 23. 1–39.

[11] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538, 7626 (2016), 471–476.

[12] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. (2024). arXiv:2310.06839 [cs.CL] https://arxiv.org/abs/2310.06839

[13] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. https://www.usenix.org/conference/osdi24/presentation/lee-wonbeom

[14] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations (ICLR)*.

[15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. (2021). arXiv:2005.11401 [cs.CL] https://arxiv.org/abs/2005.11401

[16] Dong Liu. 2024. Contemporary model compression on large language models inference. *arXiv preprint arXiv:2409.01990* (2024). https://arxiv.org/abs/2409.01990v1

[17] Dong Liu and Yanxuan Yu. 2025. GraphSnapShot: A System for Graph Machine Learning Acceleration. In *Machine Learning for Computer Architecture and Systems 2025*. https://openreview.net/forum?id=KeHes2SVxs

[18] Dong Liu and Yanxuan Yu. 2025. HSGM: Hierarchical Segment-Graph Memory for Scalable Long-Text Semantics. In *Proceedings of the 14th Joint Conference on Lexical and Computational Semantics (*SEM 2025)*, Lea Frermann and Mark Stevenson (Eds.). Association for Computational Linguistics, Suzhou, China, 328–337. doi:10.18653/v1/2025.starsem-1.26

[19] Dong Liu and Yanxuan Yu. 2025. Mt2st: Adaptive multi-task to single-task learning. In *Proceedings of the 1st Workshop on Multimodal Augmented Generation via Multimodal Retrieval (MAGMaR 2025)*. 79–89.

[20] Dong Liu and Yanxuan Yu. 2025. TinyServe: Query-Aware Cache Selection for Efficient LLM Serving. In *Proceedings of the 33rd ACM International Conference on Multimedia* (Dublin, Ireland) *(MM '25)*. Association for Computing Machinery, New York, NY, USA, 12529–12537. doi:10.1145/3746027.3758181

[21] Dong Liu and Yanxuan Yu. 2026. CXL-SpecKV: A Disaggregated FPGA Speculative KV-Cache for Datacenter LLM Serving. In *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (USA) *(FPGA '26)*. Association for Computing Machinery, New York, NY, USA, 56–66. doi:10.1145/3748173.3779188

[22] Dong Liu, Yanxuan Yu, Jiayi Zhang, Yifan Li, Ben Lengerich, and Ying Nian Wu. 2025. FastCache: Fast Caching for Diffusion Transformer Through Learnable Linear Approximation. (2025). arXiv:2505.20353 [cs.LG] https://arxiv.org/abs/2505.20353

[23] Xiabin Liu, Yanan Zheng, Zhenglin Wang, et al. 2024. DynamicKV: Task-Aware Adaptive KV Cache Compression for Long Context LLMs. (2024). arXiv:2412.14838 [cs.CL] https://arxiv.org/abs/2412.14838

[24] Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are Multi-State RNNs. (2024). arXiv:2401.06104 [cs.CL] https://arxiv.org/abs/2401.06104

[25] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. YaRN: Efficient Context Window Extension of Large Language Models. (2023). arXiv:2309.00071 [cs.CL] https://arxiv.org/abs/2309.00071

[26] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive Transformers for Long-Range Sequence Modelling. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SylKikSYDH

[27] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics* 9 (2021), 53–68.

[28] Noam Shazeer. 2019. Fast Transformer Decoding: One Write-Head is All You Need. (2019). arXiv:1911.02150 [cs.NE] https://arxiv.org/abs/1911.02150

[29] Mustafa Shukor, Enrico Fini, Victor Guilherme Turrisi da Costa, Matthieu Cord, Joshua Susskind, and Alaaeldin El-Nouby. 2025. Scaling Laws for Native Multimodal Models. (2025). arXiv:2504.07951 [cs.CV] https://arxiv.org/abs/2504.07951

[30] Simeng Sun, Yang Liu, Dan Iter, Chenguang Zhu, Michael Zeng, and Mohit Iyyer. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? (2024). arXiv:2404.06654 [cs.CL] https://arxiv.org/abs/2404.06654

[31] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-Aware Sparsity for Efficient Long-Context LLM Inference. (2024). arXiv:2406.10774 [cs.CL] https://arxiv.org/abs/2406.10774

[32] DeepSeek Team. 2024. DeepSeek-V2 Technical Report. https://github.com/deepseek-ai/deepseek-v2. (2024).

[33] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. (2023). arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[34] Szymon Tworkowski, Konrad Staniszewski, Mikołaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr Miłoś. 2023. Focused Transformer: Contrastive Training for Context Scaling. (2023). arXiv:2307.03170 [cs.CL] https://arxiv.org/abs/2307.03170

[35] Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory Networks. (2015). arXiv:1410.3916 [cs.AI] https://arxiv.org/abs/1410.3916

[36] Xiaoju Ye, Shi Han, and Dongmei Zhang. 2024. Infinite Retrieval: Attention Enhanced LLMs in Long-Context Processing. (2024). arXiv:2407.19525 [cs.CL] https://arxiv.org/abs/2407.19525