Memory-Augmented Transformers via Learnable Cross-Attention Memory Banks

Tasmay Pankaj Tibrewal[*]
tasmay.tibrewal@fractal.ai
Fractal AI Research

Pritish Saha[*]
pritish.saha@kgpian.iitkgp.ac.in
IIT Kharagpur

Ankit Meda[*]
ankitm18@kgpian.iitkgp.ac.in
IIT Kharagpur

January 2026

Abstract

Transformers, despite their remarkable success across natural language processing and beyond, lack an explicit architectural mechanism for persistent memory storage and retrieval. Current approaches to memory augmentation predominantly rely on engineering solutions such as KV-caching or retrieval-augmented generation, which operate at inference time rather than at the architectural level. We propose a novel architectural modification: the integration of learnable memory banks accessed via cross-attention mechanisms within transformer blocks. This memory bank consists of latent tokens, randomly initialized and learned during training, which the model can reference as contextual information. We present two complementary research directions: (1) training memory-augmented transformers from scratch, and (2) utilizing memory layers as parameter-efficient adapters for existing pretrained models. We further introduce chapter-based routing inspired by Mixture-of-Experts (MoE) architectures to enable efficient scaling of memory bank size. Additionally, we outline a future direction for dynamic memory updates during inference to enable persistent context retention across conversations. This proposal targets submission to the ICLR 2026 Workshop on New Frontiers in Associative Memories.

1 Introduction

Transformer architectures [Vaswani et al., 2017] have become the dominant paradigm for sequence modeling, achieving state-of-the-art results across natural language processing, computer vision, and multimodal applications. However, a fundamental limitation persists: transformers do not possess an explicit, architecturally integrated memory layer. Some internal components can be interpreted as storing and retrieving patterns implicitly, for example the feed-forward sublayers behave like key-value memories over training data [Geva et al., 2021]. Nonetheless, an explicit, addressable memory bank may provide a clearer mechanism for controlled capacity scaling and retrieval beyond what is captured implicitly in parameters. While the self-attention mechanism allows tokens to attend to all other tokens within the context window, there is no dedicated component for storing and retrieving knowledge that persists beyond the immediate input sequence. Notably, recent theoretical work has established connections between attention mechanisms and associative memory systems, showing that the softmax attention operation corresponds to the update rule of Modern Hopfield Networks [Ramsauer et al., 2020]. Related work on dense associative memories shows that energy-based retrieval can support very large pattern

[*]Equal contribution.

capacity, which motivates adding explicit memory components that scale beyond the immediate context window [Krotov and Hopfield, 2016, Demircigil et al., 2017]. This connection suggests that explicit memory mechanisms may be a natural extension of the transformer architecture.

Current solutions to this limitation fall into two broad categories. The first involves engineering approaches such as KV-caching, where key-value pairs from previous tokens are stored and reused during inference to avoid redundant computation. The second encompasses retrieval-augmented generation (RAG) systems [Lewis et al., 2020], where external documents are retrieved and concatenated to the input context. While these methods have proven effective in practice, they represent applied solutions rather than fundamental architectural innovations. The memory in these systems exists outside the model's learned representations. KV cache retention, eviction, and compression schemes provide strong inference-time baselines for long-context use, but do not add a persistent learned memory store [Zhang et al., 2023, Xiao et al., 2023, Karami et al., 2025]. For inference-time efficiency, general KV-cache eviction frameworks such as NACL provide important baselines for long-context inference [Chen et al., 2024]. We view these methods as complementary: cache policies manage attention history, while memory banks provide a learned persistent substrate. Separately, differentiable external-memory architectures such as Neural Turing Machines and Differentiable Neural Computers demonstrate end-to-end learned read/write memory, but were not developed around modern transformer blocks [Graves et al., 2014, 2016].

Several prior works have explored memory augmentation at the architectural level. Memformer [Wu et al., 2020] introduced cross-attention between layer activations and external memory banks. The Recurrent Memory Transformer (RMT) [Bulatov et al., 2022] augments transformers with global memory tokens passed between segments to enable recurrence. The Memorizing Transformer [Wu et al., 2022] uses approximate kNN lookup into a non-differentiable memory of past key-value pairs. Sandler et al. [Sandler et al., 2022] demonstrated learnable memory tokens for fine-tuning image transformers. Shah et al. [Shah et al., 2025] propose learned meta-tokens as trainable landmarks that improve recall and length generalization. Persistent learned memory vectors integrated directly into attention have also been explored [Sukhbaatar et al., 2019], which is closely related in spirit, although our design uses cross-attention to an explicit bank and focuses on scalable routing. Product Key Memory layers [Lample et al., 2019] showed how to efficiently access large memory banks using product key addressing. More recently, comprehensive surveys [Omidi et al., 2025] have systematically categorized these approaches according to functional objectives, memory representations, and integration mechanisms.

In this proposal, we present an architectural modification that introduces a learnable memory bank integrated via cross-attention. Unlike inference-time memory solutions, our memory bank is learned during training, allowing the model to develop compressed, efficient representations of knowledge. We propose two research tracks: (1) training memory-augmented models from scratch, and (2) using memory layers as parameter-efficient adapters. We further introduce MoE-inspired chapter-based routing to scale memory efficiently, and outline future work on dynamic memory updates during inference.

2 Idea Overview

Base proposal: Introduce an explicit, architectural memory component for transformers, rather than relying only on inference-time mechanisms such as extending the context window, KV-cache policies, or external retrieval pipelines [Lewis et al., 2020].

Objective: Standard transformers do not expose an explicit, addressable memory layer. Our goal is to increase effective knowledge retention and controllable capacity by adding a persistent memory substrate within the architecture.

Core change (learnable memory bank): Maintain a memory bank of latent tokens, randomly initialized and optimized end-to-end during training. For each forward pass, the token stream produces queries ($\mathbf{Q}$), while the memory bank provides keys and values ($\mathbf{K}, \mathbf{V}$). Cross-attention then retrieves relevant information from this persistent latent store, enabling memory access without requiring the information to be present in the input sequence.

Integration into a transformer block (variants):

- **Variant A:** Self-attention $\rightarrow$ memory cross-attention $\rightarrow$ MLP.
- **Variant B:** Self-attention $\rightarrow$ MLP $\rightarrow$ memory cross-attention $\rightarrow$ MLP.
- Memory layers need not appear in every block; we can place them in the first k layers, last k layers, every n-th layer, or any other sparse schedule.

Shared vs. per-layer memory:

- **Per-layer banks:** smaller bank per memory-enabled layer (supports layer-specific information specialization).
- **Shared bank:** one larger bank used by many layers (increases accessible information per access, with potential interference from attending to less relevant tokens).
- Hypothesis: even if layers operate in slightly different representational spaces, layer-specific memory projections can learn the necessary alignment jointly with the key/value mapping.

Scaling memory access (chapters + routing): Since cross-attention cost grows linearly with the number of memory tokens, we partition the bank into "chapters" and train a lightweight router (MoE-style [Shazeer et al., 2017, Fedus et al., 2022]) to select top-k chapters per input. Each memory-enabled layer then attends only to routed chapters (e.g., 100k tokens split into 100 chapters; attending to top 20 chapters results in 20k tokens per access).

Token-wise routing (challenge and opportunity): In MoE, routing is typically token-wise. Here, token-wise chapter routing during prefill/training is memory-prohibitive because it effectively requires per-token routed $\mathbf{K}, \mathbf{V}$ materialization, scaling with sequence length. A promising systems direction is a custom matmul/CUDA kernel that stores only *unique* routed chapters and uses an index map so many query tokens can reuse the same chapter rows without redundant copies. Token-wise routing becomes substantially more feasible during autoregressive generation (sequence length = 1), but may increase VRAM usage because routed memory must coexist with the KV cache.

Increasing bank size under VRAM constraints:

- **Quantized storage:** store memory tokens in low precision.
- **Low-rank structure** (LoRA-inspired [Hu et al., 2022]): factorize the bank, or store memory directly in a reduced dimension and perform memory attention in the reduced space before projecting back.
- The same techniques apply both to the training-time memory bank and to the inference-time context bank described below.

Mitigating stage-wise forgetting: When moving from pretraining $\rightarrow$ instruction fine-tuning, protect stored knowledge by freezing the memory bank (and optionally memory projections), or by applying lower learning rates to memory parameters than to other trainable components.

Experimental plan (workshop-feasible):

- **From-scratch track:** Train a smaller model with memory from scratch and compare against a same-size baseline transformer, testing whether explicit memory improves retention and capacity.
- **Adapter track (primary):** Use memory layers as PEFT modules on a pretrained model; compare full fine-tuning vs LoRA vs Memory-Adapters vs Memory-Adapters+LoRA. Evaluate on reasoning traces and/or domain datasets (medical/legal), where retention and controlled updates are central.

Future extension (dynamic inference-time context bank): In addition to the learned (static) memory bank, maintain a separate *context bank* that stores information encountered during inference, enabling long-horizon agents and cross-conversation memory. Outline:

- **Compression:** compress recent prompts into a small number of latent vectors (e.g., VAE/autoencoder over early+late layer embeddings).
- **Retrieval without a trained router:** cluster context vectors (k-means/hierarchical/online; IVF-PQ at scale) and attend only to top-k clusters selected by similarity to the current query.
- **Capacity management:** when the bank reaches capacity, merge nearby vectors to free space, optionally guided by usage-derived importance (how often/strongly attended) and recency/age, yielding a controlled forgetting rule.
- **Separation of concerns:** keep the learned memory bank and inference-time context bank separate so post-deployment updates do not interfere with training-time knowledge.

3 Proposed Architecture

3.1 Core Memory Bank Design

We propose augmenting the standard transformer block with a cross-attention layer that enables prompt tokens to attend to a set of learnable memory tokens. Let $\mathbf{M} \in \mathbb{R}^{N_m \times d}$ denote the memory bank, where N_m is the number of memory tokens and d is the embedding dimension. This memory bank is randomly initialized and updated through gradient descent during training.

Architecturally, this follows the general pattern of cross-attention from the token stream to a fixed latent array, as used in Perceiver-style models, although here the latent array is trained to act as a persistent knowledge store [Jaegle et al., 2021a,b]. Persistent learned memory vectors inserted into self-attention are related in spirit [Sukhbaatar et al., 2019], but our approach uses a separate bank accessed via cross-attention rather than appending persistent tokens to the self-attention context. This is also closely related in form to external-memory attention mechanisms used for long-range sequence modeling [Wu et al., 2020].

Given a sequence of prompt tokens with hidden representations $\mathbf{H} \in \mathbb{R}^{L \times d}$ (where L is the sequence length), the cross-attention operation proceeds as follows:

$$\mathbf{Q} = \mathbf{H}\mathbf{W}_Q \tag{1}$$

$$\mathbf{K} = \mathbf{M}\mathbf{W}_K \tag{2}$$

$$\mathbf{V} = \mathbf{M}\mathbf{W}_V \tag{3}$$

$$\text{MemAttn}(\mathbf{H}, \mathbf{M}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{4}$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_k}$ are learnable projection matrices and d_k is the head dimension.

3.2 Transformer Block Variants

We consider two variants for integrating the memory cross-attention layer within a transformer block:

Variant A (Post-Self-Attention):

1. Self-attention layer
2. Memory cross-attention layer
3. MLP layer

Variant B (Post-MLP):

1. Self-attention layer
2. MLP layer
3. Memory cross-attention layer
4. Additional MLP layer

Variant B provides an additional non-linear transformation after memory retrieval, potentially enabling more complex integration of retrieved information.

3.3 Memory Layer Placement Strategies

Rather than placing memory cross-attention in every transformer block, selective placement may prove more efficient:

- **First-k layers:** Memory reference in the first k layers, where syntactic and lower-level information is typically processed.
- **Last-k layers:** Memory reference in the final k layers, where higher-level semantic information is integrated.
- **Every-nth layer:** Periodic memory reference (e.g., every 3rd layer).
- **All layers:** Full memory integration throughout the network.

The optimal placement strategy likely depends on the type of information stored in the memory bank and the downstream task requirements.

3.4 Shared vs. Per-Layer Memory Banks

Two memory bank configurations are possible:

Per-Layer Memory Banks: Each memory-enabled layer maintains its own memory bank $\mathbf{M}^{(l)} \in \mathbb{R}^{N_m^{(l)} \times d}$. This allows different layers to store information at different levels of abstraction, analogous to how different transformer layers capture different linguistic phenomena.

Shared Memory Bank: A single memory bank $\mathbf{M} \in \mathbb{R}^{N_m \times d}$ is shared across all memory-enabled layers. This configuration provides each layer access to a larger pool of information but raises a concern: different layers operate in slightly different representational vector-spaces (linear subspaces).

We address this vector-space mismatch concern as follows. The projection matrices $\mathbf{W}_K^{(l)}$ and $\mathbf{W}_V^{(l)}$ at each layer l can learn not only the key/value transformations but also the vector-space transformation between the shared memory space and the layer-specific representation space. Since these projection matrices have dimension $d \times d_k$, and a linear transformation of this size is sufficient to map between any two d-dimensional vector-spaces, the layer-specific projections can absorb the vector-space alignment task without requiring additional parameters.

Formally, we can conceptualize this as:

$$\mathbf{W}_K^{(l)} = \mathbf{W}_K^{\text{core}} \cdot \mathbf{T}_K^{(l)} \tag{5}$$

$$\mathbf{W}_V^{(l)} = \mathbf{W}_V^{\text{core}} \cdot \mathbf{T}_V^{(l)} \tag{6}$$

where $\mathbf{W}_K^{\text{core}}$ performs the standard key projection and $\mathbf{T}_K^{(l)}$ handles vector-space transformation. However, since both are learnable and their product is also a $d \times d_k$ matrix, a single learned $\mathbf{W}_K^{(l)}$ can capture both functionalities.

The shared memory configuration has the advantage of providing each layer access to a larger information pool. A potential downside is that different layers may require different types of information (e.g., syntactic information in early layers, factual knowledge in middle layers), and shared memory may force layers to attend to irrelevant tokens. We propose to investigate this trade-off empirically.

4 Scaling Memory via Chapter-Based Routing

4.1 Motivation

A key challenge in memory-augmented transformers is scaling the memory bank size. With N_m memory tokens and sequence length L, the cross-attention computation scales as $O(L \cdot N_m)$. For large memory banks (e.g., $N_m = 100,000$ tokens), this becomes computationally prohibitive.

Related approaches scale memory by using sparse top-k retrieval rather than dense attention, for example product-key addressing for large memory layers and MIPS-style memory querying [Lample et al., 2019, Wu et al., 2022], and recent work shows that trainable memory layers can be scaled effectively with strong factual gains [Berges et al., 2025].

We address this challenge by introducing chapter-based routing, inspired by Mixture-of-Experts (MoE) architectures [Shazeer et al., 2017, Fedus et al., 2022]. Rather than attending to all memory tokens, we partition the memory bank into "chapters" and use a router to select the most relevant chapters for each input.

This is aligned with content-based sparse attention methods that route queries to a small subset of candidates, for example by clustering tokens and restricting attention to routed groups [Roy et al., 2021].

This goal also aligns with scalable long-term memory extensions that select a small subset of a large internal memory at each step [Wang et al., 2025].

Relatedly, MoM formulates sequence modeling with a mixture of memory components combined by learned routing or gating, offering a useful precedent for selectively accessing multiple memory slots rather than treating memory as monolithic [Du et al., 2025]. Neural Routing by Memory likewise studies routing decisions driven by a memory state, supporting the general design pattern of memory-conditioned module selection [Zhang et al., 2021]. In contrast, our emphasis is on augmenting Transformer blocks with learnable cross-attention memory banks and optional chapter-level routing, keeping the base self-attention backbone intact.

Notably, memory chapter routing can achieve much higher sparsity than typical MoE expert routing. While MoE systems distribute computational operations (MLP transformations) across experts and must maintain reasonable coverage for diverse token types, memory queries are inherently more specific: a given query typically requires access to a small, targeted subset of knowledge rather than diverse computational pathways. This allows for sparser routing (e.g., selecting 10-20 out of 1000 chapters) without degrading performance, as queries naturally align with specific memory regions rather than requiring broad computational diversity.

4.2 Chapter Organization

Let the memory bank be partitioned into C chapters, each containing $N_c = N_m/C$ tokens:

$$\mathbf{M} = [\mathbf{M}_1; \mathbf{M}_2; \ldots; \mathbf{M}_C] \tag{7}$$

where $\mathbf{M}_c \in \mathbb{R}^{N_c \times d}$ denotes the c-th chapter.

4.3 Router Architecture

The router computes chapter importance scores based on the input sequence. Let $\bar{\mathbf{h}} = \frac{1}{L}\sum_{i=1}^{L} \mathbf{h}_i$ be the mean-pooled representation of the input sequence. The router computes:

$$\mathbf{s} = \mathrm{softmax}(\mathbf{W}_r \bar{\mathbf{h}} + \mathbf{b}_r) \tag{8}$$

where $\mathbf{W}_r \in \mathbb{R}^{C \times d}$ and $\mathbf{b}_r \in \mathbb{R}^{C}$ are learnable parameters, and $\mathbf{s} \in \mathbb{R}^{C}$ contains the importance scores for each chapter.

We select the top-k chapters with highest scores and perform cross-attention only over the tokens in these selected chapters. If $k = 20$ chapters are selected from $C = 100$ chapters (each containing $N_c = 1,000$ tokens), we attend to only $20,000$ tokens instead of $100,000$, achieving a $5\times$ reduction in attention computation.

4.4 Token-Level Routing Challenges

Ideally, routing would be performed at the token level, allowing different query tokens to attend to different memory chapters. This mirrors the token-level expert assignment in standard MoE [Zhou et al., 2022]. However, token-level routing presents significant memory challenges.

Consider the tensor dimensions for standard cross-attention with sequence-level routing:

$$\mathbf{Q} \in \mathbb{R}^{B\times H\times L\times D} \tag{9}$$

$$\mathbf{K} \in \mathbb{R}^{B\times H\times N_{\text{routed}}\times D} \tag{10}$$

where B is batch size, H is number of heads, L is sequence length, D is head dimension ($D = d_k$), and N_{routed} is the number of routed memory tokens.

For token-level routing, where each token can route to different chapters, the key tensor must expand to accommodate per-token routing:

$$\mathbf{K}_{\text{token-level}} \in \mathbb{R}^{(B\times L)\times H\times N_{\text{routed}}\times D} \tag{11}$$

To enable this with standard matrix multiplication, we would need to reshape the query tensor to match. The reshaping sequence would be:

$$\begin{aligned} &\mathbf{Q} \in \mathbb{R}^{B\times H\times L\times D} \\ &\quad \xrightarrow{\text{transpose(1,2)}} \mathbb{R}^{B\times L\times H\times D} \\ &\quad \xrightarrow{\text{view(0,1)}} \mathbb{R}^{(B\times L)\times H\times D} \\ &\quad \xrightarrow{\text{unsqueeze(3)}} \mathbb{R}^{(B\times L)\times H\times D\times 1} \\ &\quad \xrightarrow{\text{transpose(2,3)}} \mathbb{R}^{(B\times L)\times H\times 1\times D} \end{aligned} \tag{12}$$

Now $\mathbf{Q}$ and $\mathbf{K}^\top$ would be compatible for matrix multiplication, producing attention scores of shape $(B \times L) \times H \times 1 \times N_{\text{routed}}$, which can be reshaped back to $B \times L \times H \times N_{\text{routed}}$ for the attention computation.

However, this approach is infeasible due to memory requirements. For practical values ($B = 250$, $L = 10,000$, $H = 32$, $D = 128$, $N_{\text{routed}} = 16,000$), the key tensor would require approximately 300 TB of memory, which is clearly prohibitive.

We identify a potential solution through custom CUDA kernels. Rather than materializing the full key tensor, a custom matrix multiplication kernel could:

1. Store only the unique chapters in memory: $\mathbf{K}_{\text{unique}} \in \mathbb{R}^{C_{\text{unique}}\times H\times N_c\times D}$ where C_{unique} is the number of unique chapters and N_c is tokens per chapter
2. Maintain a routing index mapping each query token to its assigned chapters (approximately 160 MB for 40 million references, approximately negligible)
3. Perform attention by referencing the same chapter data for multiple queries, avoiding redundant storage

This would reduce memory from $O(B \cdot L \cdot N_{\text{routed}})$ to $O(C_{\text{unique}} \cdot N_c)$ plus routing indices. For a memory bank of 1 million tokens divided into 1000 chapters of 1000 tokens each, this yields approximately 8 GB for the chapters, a reduction of approximately 40,000× compared to the naïve approach. However, implementing this custom kernel requires significant CUDA expertise and careful optimization to maintain parallelization efficiency.

Token-Level Routing During Generation: While token-level routing is infeasible during prefill (processing the entire input sequence) and training due to memory constraints, it becomes viable during autoregressive generation. During generation, we process one token at a time (sequence length $L = 1$), which naturally eliminates the multiplicative L factor in memory requirements. In this regime, token-level routing can be performed efficiently, though it requires storing the routed memory tokens in VRAM alongside the KV cache. For example, routing to 50,000 memory tokens per generation step would add approximately 200 MB per sample to VRAM (at full precision for a 4096-dimensional model). This overhead could be further reduced by the custom CUDA kernel approach described above, limiting total memory to the size of the memory bank itself rather than per-token copies.

Given these challenges, we leave token-level routing during prefill as future work and proceed with sequence-level routing for our workshop submission, while noting that token-level routing during generation remains a practical option.

5 Memory Adapters for Efficient Fine-Tuning

5.1 Motivation

Given the timeline constraints for the ICLR workshop (deadline: February 14, 2026), pretraining a memory-augmented transformer from scratch and conducting comprehensive experiments may not be feasible within approximately three weeks. We therefore propose an alternative approach: using memory layers as parameter-efficient adapters for existing pretrained models.

This approach offers several advantages:

- Leverages the knowledge already encoded in pretrained weights
- Requires training only the memory bank and associated projection matrices
- Enables direct comparison with established PEFT methods (e.g., adapters [Houlsby et al., 2019] and prefix-tuning [Li and Liang, 2021])
- Includes LoRA as a standard low-rank adaptation baseline [Hu et al., 2022]
- Provides comparison against strong prompt-based tuning baselines such as P-Tuning v2 [Liu et al., 2022]
- Reduces computational requirements for experiments

Recent work also treats memory systems through the lens of parameter-efficient adapters. MemLoRA distills expert adapters for on-device memory systems, providing a concrete reference point for memory as a PEFT module design choices [Bini et al., 2025]. Our approach is orthogonal: we introduce learnable cross-attention memory banks (and optional routing), which can coexist with adapter-based fine-tuning.

5.2 Memory Adapter Architecture

For a pretrained transformer, we insert memory cross-attention layers at selected positions while keeping the original weights frozen. The trainable parameters consist of:

- Memory bank $\mathbf{M} \in \mathbb{R}^{N_m \times d}$
- Projection matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_k}$ per memory layer
- (Optional) Router parameters for chapter-based routing

5.3 Memory Compression Techniques

To reduce the parameter count and memory footprint of memory adapters, we propose several compression strategies:

Quantized Memory Banks: Store memory tokens in lower precision (e.g., 4-bit or 8-bit) while maintaining full precision for computations.

Low-Rank Memory Factorization: We can decompose the memory bank into lower-rank factors:

$$\mathbf{M} = \mathbf{A}\mathbf{B}^\top \tag{13}$$

where $\mathbf{A} \in \mathbb{R}^{N_m \times r}$ and $\mathbf{B} \in \mathbb{R}^{d \times r}$ with $r \ll \min(N_m, d)$.

This reduces the parameter count from $N_m \cdot d$ to $(N_m + d) \cdot r$. However, this comes at the cost of reduced expressiveness, as each memory token can now store less information. This technique is particularly valuable for the memory adapter approach, as it dramatically reduces the number of trainable parameters that need to be added to a frozen pretrained model, making memory adapters more parameter-efficient and competitive with other PEFT approaches.

Low-Rank Projections: The projection matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ can also be factorized:

$$\mathbf{W}_Q = \mathbf{W}_Q^{\text{down}} \mathbf{W}_Q^{\text{up}} \tag{14}$$

where $\mathbf{W}_Q^{\text{down}} \in \mathbb{R}^{d \times r}$ and $\mathbf{W}_Q^{\text{up}} \in \mathbb{R}^{r \times d_k}$. This projection factorization further reduces the adapter parameter count and can be combined with memory bank factorization for maximum parameter efficiency.

Low-Rank Dimension Reduction: An alternative approach is to store the memory bank directly in reduced dimensions $\mathbf{M} \in \mathbb{R}^{N_m \times r}$ where $r \ll d$ (e.g., $r = 512$ instead of $d = 4096$). The projection matrices then map between the full model dimension and the reduced memory dimension:

$$\mathbf{W}_Q \in \mathbb{R}^{d \times r} \quad \text{(maps queries from } d \text{ to } r\text{)} \tag{15}$$

$$\mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{r \times r} \quad \text{(operate in reduced space)} \tag{16}$$

$$\mathbf{W}_O \in \mathbb{R}^{r \times d} \quad \text{(maps outputs back to } d\text{)} \tag{17}$$

The cross-attention computation becomes:

$$\mathbf{Q}_r = \mathbf{H}\mathbf{W}_Q \in \mathbb{R}^{L \times r} \tag{18}$$

$$\mathbf{K}_r = \mathbf{M}\mathbf{W}_K \in \mathbb{R}^{N_m \times r} \tag{19}$$

$$\mathbf{V}_r = \mathbf{M}\mathbf{W}_V \in \mathbb{R}^{N_m \times r} \tag{20}$$

$$\text{MemAttn}_r(\mathbf{H}, \mathbf{M}) = \text{softmax}\left(\frac{\mathbf{Q}_r \mathbf{K}_r^\top}{\sqrt{r}}\right) \mathbf{V}_r \mathbf{W}_O \tag{21}$$

This approach reduces memory bank storage by a factor of d/r (e.g., $8\times$ reduction for $r = 512$, $d = 4096$) while performing all attention computations in the lower-dimensional space. The output projection $\mathbf{W}_O$ maps the result back to the full model dimension for integration with subsequent layers. This trades expressiveness for efficiency: each memory token stores less information, but the dramatic reduction in storage enables much larger memory banks within the same memory budget. For memory adapters specifically, this dimension reduction approach is especially advantageous: it allows deploying memory adapters with substantially larger memory banks (potentially 100k+ tokens) on resource-constrained hardware, making the adapter approach practical for real-world fine-tuning scenarios where VRAM is limited.

5.4 Proposed Experiments

We propose the following experimental comparisons using a base model such as Qwen-2.5-1.5B [Qwen Team, 2024] or Qwen-2.5-Math-1.5B:

1. **Full Fine-Tuning:** Update all model parameters (baseline)
2. **Adapters:** Inserted adapter modules [Houlsby et al., 2019]

3. **Prefix-Tuning:** Learned prefix prompts [Li and Liang, 2021]
4. **P-Tuning v2:** Deep prompt tuning baseline [Liu et al., 2022]
5. **LoRA:** Low-rank adaptation baseline [Hu et al., 2022]
6. **Sparse Memory PEFT:** Hierarchical sparse memory baselines such as SPARTAN [Deshpande et al., 2022]
7. **Memory Adapters (Ours):** Memory bank with cross-attention
8. **Memory Adapters + LoRA:** Memory bank with cross-attention combined with LoRA

Datasets: We consider fine-tuning on:

- DeepSeek-R1 reasoning traces from Open-R1 math dataset (to evaluate whether memory can capture reasoning patterns)
- Domain-specific datasets (medical, legal) where factual knowledge retention is critical

Evaluation: Performance will be measured on held-out test sets, with particular attention to:

- Task accuracy
- Parameter efficiency (trainable parameters vs. performance)
- Inference speed overhead from memory attention

5.5 Preventing Knowledge Loss During Fine-Tuning

A known issue with fine-tuning is catastrophic forgetting, where the model loses information acquired during pretraining. Memory adapters may help mitigate this issue, as the pretrained weights remain frozen. Related discrete adaptor and bottleneck approaches aim to localize updates into a small set of key-value slots, reducing interference during continual edits [Hartvigsen et al., 2023, Träuble et al., 2023]. Additionally, we propose:

Memory Freezing: After pretraining (for from-scratch models) or an initial fine-tuning phase, freeze the memory bank and associated $\mathbf{W}_K, \mathbf{W}_V$ matrices while continuing to train other components.

Differential Learning Rates: Apply lower learning rates to memory bank parameters compared to other trainable parameters, reducing the rate at which memorized information is overwritten.

6 From-Scratch Training

While memory adapters are our primary focus for the workshop submission, we also outline plans for training memory-augmented transformers from scratch, if possible in time.

6.1 Proposed Setup

- **Model Size:** Approximately 100M parameters
- **Pretraining Data:** 5-10 billion tokens
- **Instruction Fine-Tuning:** 10-100M tokens
- **Memory Configuration:** 10,000-1M memory tokens (shared or per-layer variants)

6.2 Expected Outcomes

We hypothesize that the learned memory bank will contain more compressed and efficient representations compared to:

- Textual memory (as in retrieval-augmented systems)
- Non-differentiable memory stores (as in Memorizing Transformers [Wu et al., 2022])

- Inference-time memory (as in TransformerFAM [Hwang et al., 2024])

The model should be compared against a baseline transformer of equivalent parameter count (excluding memory bank) on standard language modeling benchmarks.

7 Dynamic Memory Update During Inference (Future Work)

7.1 Motivation

The memory mechanisms described thus far are static: the memory bank is fixed after training and does not update during inference. This limits the model's ability to incorporate new information encountered at inference time.

This goal is also related to online adaptation methods that maintain a memory of amortized contexts during deployment [Tack et al., 2024].

We propose extending the architecture with a dynamic "context bank" that is updated during inference, enabling:

- Long-context retention beyond the attention window (current models effectively handle up to approximately 400,000 tokens before losing coherence; existing techniques like context compression and selective memory documentation mitigate this but still face limitations for autonomous agents running for hours or days)
- Cross-conversation memory (remembering information across sessions)
- Personal agents that accumulate knowledge about users over time
- Autonomous agents operating continuously over extended periods (hours to days) with efficient, precise long-term memory representation
- Rapid knowledge updates on recent news or important events without retraining

Recent work on long-term and episodic memory for language models studies related goals such as retaining information across extended interactions and controlling what is stored and retrieved [Wang et al., 2023, Liu et al., 2024]. Test-time memory mechanisms that learn what to store during inference provide an additional point of comparison for dynamic updates [Behrouz et al., 2025]. Compressed context memory for online language model interaction addresses a closely related setting where a model maintains and retrieves a compact interaction memory over time [Kim et al., 2024].

Episodic test-time memory is an active direction. EM-LLM introduces an episodic memory mechanism for large language models, storing and retrieving past episodes to improve long-horizon consistency [Fountas et al., 2025]. This is closely adjacent to our Dynamic Context Bank discussion; however, our main contribution remains architectural memory via learnable cross-attention banks, and our dynamic memory sketches emphasize alternative compression and structuring choices rather than purely episodic retrieval.

This architectural approach offers several advantages over existing memory systems. Unlike current ChatGPT-like memory implementations that rely on tool-based text storage and selective text retrieval, our context bank operates at the architectural level with learned latent representations. This enables the model to remember fine-grained details more clearly and precisely, as the memory is embedded in the model's representational space rather than stored as unstructured text that must be re-encoded at each use.

7.2 Context Bank Architecture

We maintain a separate context bank $\mathbf{C} \in \mathbb{R}^{N_{ct} \times d}$ that stores compressed representations of inference-time inputs. The context bank is distinct from the memory bank to prevent training-time knowledge from being overwritten.

Compression: Input sequences are compressed before storage using a VAE-style encoder. This direction is complementary to recent context compression modules that produce compact representations for long-context inference [Activation Beacon, 2024, Wang et al., 2024, Ge et al., 2023]. Learned context auto-compressors for language models are a related direction [Chevalier et al., 2023]. Autoencoding-free context compression via contextual semantic anchors is another related approach that targets efficient long-context inference without a separate reconstruction model [Liu et al., 2025]. This line of work is relevant for any Dynamic Context Bank variant that must retain salient information under tight context or KV-cache budgets. Given input tokens with embeddings from the first layer $\mathbf{E}^{(1)} \in \mathbb{R}^{L \times d}$ and last layer $\mathbf{E}^{(L_{\max})} \in \mathbb{R}^{L \times d}$, we train an encoder:

$$\mathbf{z} = \text{Encoder}([\mathbf{E}^{(1)}; \mathbf{E}^{(L_{\max})}]) \tag{22}$$

where $\mathbf{z} \in \mathbb{R}^{L_c \times d}$ with $L_c \ll L$ (e.g., compressing 10,000 tokens to 100 tokens).

The encoder is trained to reconstruct the concatenated embeddings, ensuring that both raw semantic information (from early layers) and processed information (from late layers) are preserved.

Alternative: Time Series VAE: Instead of treating the token sequence as independent embeddings, we can train a time series VAE that explicitly models temporal dependencies in the sequence. This approach captures the sequential structure and dependencies between tokens, potentially enabling better compression of coherent spans of text (e.g., complete sentences or paragraphs) by leveraging their temporal structure. The time series VAE would use recurrent or convolutional encoders to process the sequence temporally before compressing it to the latent representation.

Storage: Compressed representations $\mathbf{z}$ are appended to the context bank until a maximum capacity $N_{ct}^{\max}$ is reached.

Retrieval: Since attending to a large context bank (e.g., 1M tokens) is computationally prohibitive, we use clustering-based retrieval (clusters are analogous to chapters here, but since we are adding tokens in test-time, we cannot have a trained router for retrieval and thus we cluster similar tokens together for retrieval), similar to approaches in efficient attention literature [RetrievalAttention, 2024]:

1. Cluster context bank tokens using k-means or hierarchical clustering
2. For a new query, compute dot products with cluster centroids
3. Select top-k clusters and attend only to tokens within those clusters

This approach requires no training at inference time, as cluster selection is based on embedding similarity rather than learned routing.

Scalable Clustering Methods: For very large context banks, standard k-means may become computationally expensive. We can employ more scalable techniques:

- **Hierarchical clustering:** Build a tree structure enabling logarithmic-time search for relevant clusters
- **Online k-means:** Update cluster centroids incrementally as new tokens are added, avoiding full re-clustering
- **IVF-PQ (Inverted File with Product Quantization):** Use inverted index structures with quantized representations for memory-efficient storage and fast approximate nearest neighbor search

These methods enable efficient addition of new tokens to the context bank and fast retrieval even as the bank scales to millions of tokens.

7.3 Memory Consolidation

When the context bank reaches capacity and new information must be stored, we propose a consolidation mechanism inspired by how humans forget information:

Vector Merging: Identify pairs of vectors with highest cosine similarity and merge them via addition or averaging:

$$\mathbf{c}_{\text{merged}} = \frac{\mathbf{c}_i + \mathbf{c}_j}{2} \tag{23}$$

This is inspired by RAG systems where chunk embeddings are averaged without catastrophic information loss for semantic matching.

Importance-Weighted Merging: Incorporate additional factors when selecting which vectors to merge:

- **Reference frequency:** How often a vector has been attended to
- **Attention importance:** Average softmax weight when attended
- **Recency (Age):** Newer information is retained longer (benefit of doubt for importance). Age is measured as the number of inference runs (context bank updates) since the vector was added, where one run corresponds to processing one prompt through the model. This provides a discrete timestep measure of how long information has been stored.

A scoring function could weight these factors:

$$\text{importance}(\mathbf{c}) = \alpha \cdot \text{freq}(\mathbf{c}) + \beta \cdot \text{avg\_attn}(\mathbf{c}) + \gamma \cdot \text{recency}(\mathbf{c}) \tag{24}$$

where recency is inversely related to age (recent tokens have low age values). Vectors with low importance scores are prioritized for merging.

Note: All compression techniques (low-rank, quantization) described for the memory bank can also be applied to the context bank.

8 Relation to Prior Work

Our proposal builds upon and differs from several lines of prior research. We provide a comprehensive comparison to position our contributions.

Associative Memory and Hopfield Networks: Classical Hopfield networks [Hopfield, 1982] provided the foundational framework for associative memory in neural networks. Subsequent work on Modern Hopfield and dense associative memory models established that these systems can achieve very large (in some settings, exponential) storage capacity [Krotov and Hopfield, 2016, Demircigil et al., 2017]. Ramsauer et al. [Ramsauer et al., 2020] then showed that the attention mechanism in transformers is mathematically equivalent to the update rule of Modern Hopfield Networks, providing theoretical grounding for viewing attention as associative retrieval. Our work extends this view by adding an explicit, learnable memory bank that expands the set of patterns available for retrieval.

Cross-Attention Memory with Dynamic Updates: Several works have explored cross-attention to memory banks. Memformer [Wu et al., 2020] introduced cross-attention between layer activations and external memory banks using the same Q/K/V formulation as our approach (input provides Q, memory provides K,V) but with gated updates that modify memory during the forward pass. Most recently, LM2 (Large Memory Models) [Kang et al., 2025] proposed an auxiliary memory module with cross-attention and LSTM-style gating (input, output, forget gates) for dynamic memory updates. Stateful Memory-Augmented Transformers [Wu et al., 2022] apply similar ideas to dialogue modeling with recurrent memory states. MEMORYLLM [Wang et al., 2024] is representative of self-updatable language models that maintain an internal latent memory pool during deployment, and M+ extends this direction toward scalable long-term memory [Wang et al., 2025]. In this workshop submission, we focus primarily on static learned memory banks with chapter routing and a PEFT memory-adapter instantiation, while treating dynamic updates as future work. Our key distinction is that our memory bank is **static after training**: it stores persistent learned knowledge rather than working memory that evolves during inference. This design choice targets knowledge retention rather than context tracking.

Conditional Memory and Hash-Based Lookup: DeepSeek's Engram [DeepSeek, 2026] introduces conditional memory as a complementary sparsity axis to MoE, using hash-based O(1) lookup for static token patterns (common phrases, entities). While Engram and our approach share the goal of separating memory from computation, the mechanisms differ fundamentally: Engram uses deterministic hash addressing for specific n-grams, while our attention-based approach enables soft retrieval with learned content-based addressing. Our chapter routing provides semantic organization, whereas Engram's hashing is pattern-based.

Product Key Memory and Memory Layers: Lample et al. [Lample et al., 2019] introduced Product Key Memory layers that enable efficient access to very large memory banks using product key addressing. More recently, Memory Layers at Scale [Berges et al., 2025] demonstrated that trainable key-value memory layers can be scaled effectively and show strong factual gains. EMAT [Wu et al., 2022] achieves efficient memory-augmented transformers using Maximum Inner Product Search (MIPS) for fast memory querying. MATTER [Lee et al., 2024] extends this to heterogeneous knowledge sources (paragraphs and QA pairs) with fixed-length neural memories. These techniques use sparse top-k retrieval rather than attention and could be integrated with our chapter-based routing for further scalability.

Memory Tokens and Global Memory: Memory Transformer [Burtsev et al., 2020] and GMAT [Gupta and Berant, 2020] add special memory tokens that participate in self-attention (not cross-attention). ETC [Ainslie et al., 2020] uses global tokens for encoding long and structured inputs. These systems demonstrate that memory-like token mechanisms help with long-context reasoning, but memory tokens are typically dynamic per sequence and use self-attention rather than cross-attention to a separate persistent store. Persistent learned memory vectors integrated directly into attention provide another closely related mechanism for fixed memory access [Sukhbaatar et al., 2019].

Recurrent Memory Approaches: The Recurrent Memory Transformer (RMT) [Bulatov et al., 2022, 2023] augments transformers with memory tokens passed between segments via self-attention. The Associative Recurrent Memory Transformer (ARMT) [Rodkin et al., 2024] extends RMT with linear attention-style associative memory, achieving impressive results on 50M+ token sequences. Block-Recurrent Transformers [Hutchins et al., 2022] use block-level recurrence. These approaches focus on segment-level recurrence for long sequences; our memory bank stores learned knowledge accessed via cross-attention without recurrence.

Non-Parametric Memory: Memorizing Transformer [Wu et al., 2022] stores actual KV pairs from past forward passes and retrieves via kNN lookup. kNN-LM [Khandelwal et al., 2019] retrieves from a datastore of context-target pairs. These non-parametric approaches cache actual activations rather than learning compressed representations. Our memory is learned end-to-end, potentially storing more efficient representations.

Entity and Knowledge-Specific Memory: Entities as Experts [Fevry et al., 2020] stores entity embeddings in dedicated memory slots with sparse access. Facts as Experts [Verga et al., 2020] similarly uses sparse memory over symbolic knowledge with interpretable fact representations. Mention Memory [de Armas et al., 2021] stores 150M entity mention embeddings accessed via attention. QAMAT [Chen et al., 2023] uses dedicated memory for question-answering. Knowledge-Infused Self Attention Transformers [Roy, 2023] systematically infuse external knowledge graphs into transformer components. Relational Memory-Augmented Language Models [Liu et al., 2022] store relation triples rather than token pairs, enabling causal interventions. These demonstrate the value of explicit memory structures but are often task/entity-specific, whereas our approach targets general learned knowledge.

Long-Context Transformers: Transformer-XL [Dai et al., 2019] introduced segment-level recurrence. Compressive Transformers [Rae et al., 2020] compress past activations. Infini-attention [Munkhdalai et al., 2024] proposed compressive memory within attention. Cached Transformers [Zhang et al., 2023] use gated recurrent cached attention with differentiable memory. HMT [He et al., 2024] employs hierarchical memory transformers for efficient long-context processing with memory recall mechanisms. Augmenting Language Models with Long-Term Memory [Wang et al., 2023] adds persistent memory for extended conversations. These focus on extending effective context length, while we focus on persistent learned knowledge.

KV Cache Compression and Retention: H_2O [Zhang et al., 2023] proposes an attention-score-driven eviction strategy that retains high-importance ("heavy hitter") tokens in the KV cache. StreamingLLM [Xiao et al., 2023] identifies "attention sinks" and shows that preserving a small set of sink tokens stabilizes long-context generation. Trellis [Karami et al., 2025] learns to compress KV memory dynamically at test time with a bounded-memory mechanism. NACL proposes a general KV-cache eviction framework for long-context inference [Chen et al., 2024]. These works motivate our focus on learning what to store in a persistent, parameterized memory bank rather than relying on heuristic retention or runtime cache compression.

Test-Time Memory and Adaptation: Titans [Behrouz et al., 2025] learns what to memorize at test time with dynamic updates. Larimar [Liu et al., 2024] provides episodic memory control for LLMs. EM-LLM [Fountas et al., 2025] proposes episodic memory for large language models via storing and retrieving past episodes. NAMMs [Cetin et al., 2025] use evolutionary optimization for memory decisions. MAC [Tack et al., 2024] enables online adaptation with a memory of amortized contexts using meta-learning. Extended Mind Transformers [Klett and Ahle, 2024] use kNN-based retrieval to attend to external pre-computed memories. These approaches update memory during inference; our memory is fixed after training, providing stable knowledge without runtime overhead.

Cross-Attention Architectures: Perceiver [Jaegle et al., 2021a] and Perceiver IO [Jaegle et al., 2021b] use cross-attention to a latent array as a computational bottleneck. Flamingo [Alayrac et al., 2022] uses cross-attention to inject visual information into frozen language models. Our memory cross-attention is conceptually similar but targets persistent knowledge storage rather than multimodal fusion or computational bottlenecks.

Learnable Memory Tokens: Sandler et al. [Sandler et al., 2022] demonstrated learnable memory tokens for vision transformers, concatenating them to input for self-attention. Shah et al. [Shah et al., 2025] introduce learned meta-tokens injected during pre-training as content-based landmarks for improved recall and length generalization. Heterogeneous Memory Augmented Neural Networks [Park et al., 2023] use learnable memory for OOD generalization. Our work extends learnable token ideas to language models with cross-attention to a separate persistent bank (not self-attention), chapter-based routing, and memory adapter formulations.

Explicit Memory Systems: Neural Turing Machines [Graves et al., 2014] and Differentiable Neural Computers [Graves et al., 2016] pioneered differentiable read/write access to an external memory for learning algorithmic behavior. More recently, Memory3 [Yang et al., 2024] models explicit memory as retrievable model parameters, and Token Turing Machines [Hawthorne et al., 2024] use memory tapes with read/write operations. These represent alternative memory paradigms; our approach uses standard attention mechanics for compatibility with existing transformer infrastructure.

Context Compression: Activation Beacon [Activation Beacon, 2024], IC-Former [Wang et al., 2024], and In-Context Autoencoder (ICAE) [Ge et al., 2023] compress long contexts into shorter representations, and learned context auto-compressors, compressed context memory, and semantic-anchor based compression study closely related settings [Chevalier et al., 2023, Kim et al., 2024, Liu et al., 2025]. Our context bank proposal incorporates similar compression ideas but maintains a persistent store across inference sessions with importance-weighted consolidation, combining VAE-based compression with clustering-based retrieval and memory consolidation mechanisms.

Retrieval-Augmented Methods: RAG [Lewis et al., 2020] and RETRO [Borgeaud et al., 2021] augment models with retrieved text from external corpora. These store memory as text requiring re-encoding; our approach stores learned latent representations that may be more compact and task-appropriate.

Routing and Sparse Attention: Routing Transformer [Roy et al., 2021] uses clustering for content-based sparse attention. MoE architectures [Shazeer et al., 2017, Fedus et al., 2022, Zhou et al., 2022] use token-wise routing to experts, with variants like expert choice routing providing alternative routing mechanisms. MoM and memory-conditioned routing mechanisms provide related precedents for selecting among multiple memory components [Du et al., 2025, Zhang et al., 2021]. Our chapter routing applies MoE-style routing to memory chapters, a novel application that enables scaling memory banks while maintaining attention-based access.

Parameter-Efficient Fine-Tuning: Adapters [Houlsby et al., 2019] provide a modular approach to parameter-efficient transfer by inserting small trainable blocks into frozen models. Prefix-tuning [Li and Liang, 2021] and P-Tuning v2 [Liu et al., 2022] add learned continuous prompts attended via self-attention, which is conceptually close to learnable tokens. LoRA [Hu et al., 2022] provides a strong low-rank adaptation baseline. Discrete key-value adaptor and bottleneck methods such as GRACE [Hartvigsen et al., 2023] and the Discrete Key-Value Bottleneck [Träuble et al., 2023] similarly localize updates into a small memory-like interface, which is conceptually aligned with memory-as-adaptor designs. MemLoRA distills expert adapters for on-device memory systems, providing a related perspective on memory-centric adapters under deployment constraints [Bini et al., 2025]. SPARTAN [Deshpande et al., 2022] uses sparse hierarchical memory for parameter-efficient adaptation. TRIME [Zhong et al., 2022] trains language models with memory augmentation using in-batch examples as accessible memory. Our memory adapters differ in using cross-attention to a separate memory store with explicit capacity scaling via chapters.

8.1 Summary of Distinctions

Several components of our proposal have close precedents, including persistent learned memory vectors in attention [Sukhbaatar et al., 2019] and internal memory pools that can be updated over time [Wang et al., 2024, 2025]. Our contribution is to articulate and evaluate a specific combination focused on cross-attention to an explicit learned bank, chapter-based routing for scalable access, and a PEFT instantiation of memory as adapters compared against standard baselines.

9 Novelty Summary

The novel contributions of this proposal include:

1. **Learnable cross-attention memory banks** for language transformers, with analysis of shared vs. per-layer configurations and the vector-space alignment property of projection matrices.

2. **Chapter-based routing** for scaling memory bank size, including analysis of token-level routing challenges and a proposed custom CUDA kernel solution.

3. **Memory adapters** as a parameter-efficient fine-tuning method, with low-rank and quantization variants.

4. **Comprehensive experimental framework** comparing memory adapters against established PEFT baselines (e.g., adapters and prefix-tuning) and full fine-tuning.

5. **Dynamic context bank** concept with VAE compression, clustering-based retrieval, and importance-weighted memory consolidation (future work).

10 Conclusion

We have presented a comprehensive proposal for memory-augmented transformers via learnable cross-attention memory banks. The architecture provides an explicit mechanism for knowledge storage and retrieval at the architectural level, addressing a fundamental limitation of standard transformers. Chapter-based routing enables efficient scaling, while memory adapters provide a practical path to experimentation within workshop timelines. The dynamic context bank extends these ideas to enable persistent memory across inference sessions. We believe this direction holds significant promise for improving the knowledge retention and contextual reasoning capabilities of transformer-based language models.

References

Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554-2558.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G.K., Greiff, V., Kreil, D., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. (2020). Hopfield networks is all you need. In *International Conference on Learning Representations.* `https://arxiv.org/abs/2008.02217`

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. `https://arxiv.org/abs/1706.03762`

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33. `https://arxiv.org/abs/2005.11401`

Wu, Q., Lan, Z., Gu, J., and Yu, Z. (2020). Memformer: The memory-augmented transformer. *arXiv preprint arXiv:2010.06891.* `https://arxiv.org/abs/2010.06891`

Bulatov, A., Kuratov, Y., and Burtsev, M.S. (2022). Recurrent memory transformer. In *Advances in Neural Information Processing Systems*, volume 35. `https://arxiv.org/abs/2207.06881`

Bulatov, A., Kuratov, Y., and Burtsev, M.S. (2023). Scaling transformer to 1M tokens and beyond with RMT. *arXiv preprint arXiv:2304.11062.* `https://arxiv.org/abs/2304.11062`

Wu, Y., Rabe, M.N., Hutchins, D., and Szegedy, C. (2022). Memorizing transformers. In *International Conference on Learning Representations.* `https://arxiv.org/abs/2203.08913`

Sandler, M., Zhmoginov, A., Vladymyrov, M., and Jackson, A. (2022). Fine-tuning image transformers using learnable memory. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition.* `https://arxiv.org/abs/2203.15243`

Lample, G., Sablayrolles, A., Ranzato, M., Denoyer, L., and Jégou, H. (2019). Large memory layers with product keys. In *Advances in Neural Information Processing Systems*, volume 32. `https://arxiv.org/abs/1907.05242`

Omidi, P., Huang, X., Laborieux, A., Nikpour, B., Shi, T., and Eshaghi, A. (2025). Memory-augmented transformers: A systematic review from neuroscience principles to technical solutions. *arXiv preprint arXiv:2508.10824.* `https://arxiv.org/abs/2508.10824`

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations.* `https://arxiv.org/abs/1701.06538`

Fedus, W., Zoph, B., and Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1-39. `https://arxiv.org/abs/2101.03961`

Demircigil, M., Heusel, J., Löwe, M., Upgang, S., and Vermet, F. (2017). On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168(2):288–299.

Krotov, D. and Hopfield, J.J. (2016). Dense associative memory for pattern recognition. In *Advances in Neural Information Processing Systems*, volume 29. `https://arxiv.org/abs/1606.01164`

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural Turing machines. *arXiv preprint arXiv:1410.5401.* `https://arxiv.org/abs/1410.5401`

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

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z., and Chen, B. (2023). H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36. `https://arxiv.org/abs/2306.14048`

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. (2023). Efficient streaming language models with attention sinks. In *International Conference on Learning Representations.* `https://arxiv.org/abs/2309.17453`

Shah, A.N., Gupta, K., Ramji, K., and Chaudhari, P. (2025). Language modeling with learned meta-tokens. *arXiv preprint arXiv:2509.16278.* `https://arxiv.org/abs/2509.16278`

Karami, M., Behrouz, A., Kacham, P., and Mirrokni, V. (2025). Trellis: Learning to compress key-value memory in attention models. *arXiv preprint arXiv:2512.23852.* `https://arxiv.org/abs/2512.23852`

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning.* `https://arxiv.org/abs/1902.00751`

Qwen Team (2024). Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115.* `https://arxiv.org/abs/2412.15115`

Zhang, P., Qian, H., Ye, Q., and Dou, Z. (2024). Long context compression with activation beacon. *arXiv preprint arXiv:2401.03462.* `https://arxiv.org/abs/2401.03462`

Wang, X., Chen, Z., Xu, T., Xie, Z., He, Y., and Chen, E. (2024). In-context former: Lightning-fast compressing context for large language model. In *Findings of EMNLP 2024.* `https://arxiv.org/abs/2406.13618`

Cetin, E., Palmieri, A., Poli, M., Coda, D., Fang, B., and Tang, L. (2025). An evolved universal transformer memory. Sakana AI Blog. `https://sakana.ai/namm/` (Accessed: January 2026).

Hwang, D., Wang, W., Huo, Z., Sim, K.C., and Mengibar, P. (2024). TransformerFAM: Feedback attention is working memory. *arXiv preprint arXiv:2404.09173.* `https://arxiv.org/abs/2404.09173`

Liu, D., et al. (2024). RetrievalAttention: Accelerating long-context LLM inference via vector retrieval. *arXiv preprint arXiv:2409.10516.* `https://arxiv.org/abs/2409.10516`

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., and Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.* `https://arxiv.org/abs/1901.02860`

Ge, T., Hu, J., Li, L., Qiu, X., Huang, X., and Si, L. (2023). In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945.* `https://arxiv.org/abs/2307.06945`

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A., Chen, Z., Le, Q., and Laudon, J. (2022). Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems*, volume 35. `https://arxiv.org/abs/2202.09368`

Rae, J.W., Potapenko, A., Jayakumar, S.M., Hillier, C., and Lillicrap, T.P. (2020). Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations.* `https://arxiv.org/abs/1911.05507`

Munkhdalai, T., Faruqui, M., and Gopal, S. (2024). Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143.* `https://arxiv.org/abs/2404.07143`

Behrouz, A., Zhong, P., and Mirrokni, V. (2025). Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663.* `https://arxiv.org/abs/2501.00663`

Khandelwal, U., Levy, O., Jurafsky, D., Zettlemoyer, L., and Lewis, M. (2019). Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations.* `https://arxiv.org/abs/1911.00172`

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., Paez, P., Sheridan, G., Landon, J., and Sifre, L. (2021). Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning.* `https://arxiv.org/abs/2112.04426`

Berges, V.-P., Oguz, B., Haziza, D., Yih, W.-t., Zettlemoyer, L., and Ghosh, G. (2025). Memory layers at scale. In *International Conference on Machine Learning.* `https://arxiv.org/abs/2412.09764`

Burtsev, M.S., Kuratov, Y., Peganov, A., and Sapunov, G.V. (2020). Memory transformer. *arXiv preprint arXiv:2006.11527.* `https://arxiv.org/abs/2006.11527`

Gupta, A. and Berant, J. (2020). GMAT: Global memory augmentation for transformers. *arXiv preprint arXiv:2006.03274.* `https://arxiv.org/abs/2006.03274`

Ainslie, J., Ontanon, S., Alberti, C., Cvicek, V., Fisher, Z., Pham, P., Ravula, A., Sanghai, S., Wang, Q., and Yang, L. (2020). ETC: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing.* https://arxiv.org/abs/2004.08483

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. (2021). Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53-68. https://arxiv.org/abs/2003.05997

Li, X.L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics.* https://arxiv.org/abs/2101.00190

Liu, X., Ji, K., Fu, Y., Tam, W., Du, Z., Yang, Z., and Tang, J. (2022). P-Tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics.* https://arxiv.org/abs/2110.07602

Jaegle, A., Gimeno, F., Brock, A., Zisserman, A., Vinyals, O., and Carreira, J. (2021). Perceiver: General perception with iterative attention. In *International Conference on Machine Learning.* https://arxiv.org/abs/2103.03206

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ber, D., Lakshminarayanan, B., Zisserman, A., Vinyals, O., and Carreira, J. (2021). Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations.* https://arxiv.org/abs/2107.14795

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., Ring, R., Rutherford, E., Cabi, S., Han, T., Gong, Z., Samangooei, S., Monteiro, M., Menick, J., Borgeaud, S., Brock, A., Nematzadeh, A., Sharifzadeh, S., Bińkowski, M., Barreira, R., Vinyals, O., Zisserman, A., and Simonyan, K. (2022). Flamingo: A visual language model for few-shot learning. In *Advances in Neural Information Processing Systems*, volume 35. https://arxiv.org/abs/2204.14198

Kang, J., Wu, W., Christianos, F., Chan, A.J., Greenlee, F., Thomas, G., Purtorab, M., and Toulis, A. (2025). LM2: Large memory models. *arXiv preprint arXiv:2502.06049.* https://arxiv.org/abs/2502.06049

DeepSeek-AI (2026). Engram: Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372.* https://arxiv.org/abs/2601.07372

Rodkin, I., Kuratov, Y., Bulatov, A., and Burtsev, M. (2024). Associative recurrent memory transformer. In *ICML 2024 Workshop on Next Generation of Sequence Modeling Architectures.* https://arxiv.org/abs/2407.04841

Hutchins, D., Schlag, I., Wu, Y., Dyer, E., and Neyshabur, B. (2022). Block-recurrent transformers. In *Advances in Neural Information Processing Systems*, volume 35. https://arxiv.org/abs/2203.07852

Fevry, T., Soares, L.B., FitzGerald, N., Choi, E., and Kwiatkowski, T. (2020). Entities as experts: Sparse memory access with entity supervision. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing.* https://arxiv.org/abs/2004.07202

de Armas, R.M., Martins, B., and Calado, P. (2021). Mention memory: Incorporating textual knowledge into transformers through entity mention attention. *arXiv preprint arXiv:2110.06176.* https://arxiv.org/abs/2110.06176

Chen, W., Pat, A., and Roth, D. (2023). Augmenting pre-trained language models with QA-memory for open-domain question answering. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics.* `https://aclanthology.org/2023.eacl-main.117/`

Wang, W., Dong, L., Cheng, H., Liu, X., Yan, X., Gao, J., and Wei, F. (2023). Augmenting language models with long-term memory. In *Advances in Neural Information Processing Systems*, volume 36. `https://arxiv.org/abs/2306.07174`

Liu, P., Zhang, Y., and Weld, D.S. (2024). Larimar: Large language models with episodic memory control. *arXiv preprint arXiv:2403.11901.* `https://arxiv.org/abs/2403.11901`

Park, S., Kim, J., and Lee, J. (2023). Heterogeneous memory augmented neural networks. *arXiv preprint arXiv:2310.10909.* `https://arxiv.org/abs/2310.10909`

Yang, H., et al. (2024). Memory3: Language modeling with explicit memory. *arXiv preprint.* `https://arxiv.org/abs/2407.01178`

Hawthorne, C., et al. (2024). Token turing machines are efficient vision models. *arXiv preprint arXiv:2409.07613.* `https://arxiv.org/abs/2409.07613`

He, Z., Cao, Y., Qin, Z., Prakriya, N., Sun, Y., and Cong, J. (2024). HMT: Hierarchical memory transformer for efficient long context language processing. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics.* `https://arxiv.org/abs/2405.06067`

Klett, P. and Ahle, T. (2024). Extended mind transformers. *arXiv preprint arXiv:2406.02332.* `https://arxiv.org/abs/2406.02332`

Lee, D., Prakash, C.S., FitzGerald, J., and Lehmann, J. (2024). MATTER: Memory-augmented transformer using heterogeneous knowledge sources. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 16110–16121. `https://arxiv.org/abs/2406.04670`

Liu, Q., Yogatama, D., and Blunsom, P. (2022). Relational memory-augmented language models. *Transactions of the Association for Computational Linguistics*, 10:555–572. `https://arxiv.org/abs/2201.09680`

Roy, K. (2023). Knowledge-infused self attention transformers. *arXiv preprint arXiv:2306.13501.* `https://arxiv.org/abs/2306.13501`

Deshpande, A., Sultan, M.A., Ferritto, A., Kalyan, A., Narasimhan, K., and Sil, A. (2022). SPARTAN: Sparse hierarchical memory for parameter-efficient transformers. *arXiv preprint arXiv:2211.16634.* `https://arxiv.org/abs/2211.16634`

Tack, J., et al. (2024). Online adaptation of language models with a memory of amortized contexts. In *Advances in Neural Information Processing Systems*, volume 37. `https://arxiv.org/abs/2403.04317`

Verga, P., Sun, H., Soares, L.B., and Cohen, W. (2020). Facts as experts: Adaptable and interpretable neural memory over symbolic knowledge. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* `https://arxiv.org/abs/2007.00849`

Wu, Q. and Yu, Z. (2022). Stateful memory-augmented transformers for efficient dialogue modeling. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 853–867. `https://arxiv.org/abs/2209.07634`

Wu, Y., Zhao, Y., Hu, B., Minervini, P., Stenetorp, P., and Riedel, S. (2022). An efficient memory-augmented transformer for knowledge-intensive NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5184–5196. `https://arxiv.org/abs/2210.16773`

Zhang, Z., Shao, W., Ge, Y., Wang, X., Gu, J., and Luo, P. (2023). Cached transformers: Improving transformers with differentiable memory cache. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(15):16935–16943. `https://arxiv.org/abs/2312.12742`

Zhong, Z., Lei, T., and Chen, D. (2022). Training language models with memory augmentation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5657–5673. `https://arxiv.org/abs/2205.12674`

Sukhbaatar, S., Grave, E., Bojanowski, P., and Joulin, A. (2019). Augmenting self-attention with persistent memory. *arXiv preprint arXiv:1907.01470*. `https://arxiv.org/abs/1907.01470`

Geva, M., Schuster, R., Berant, J., and Levy, O. (2021). Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. `https://arxiv.org/abs/2012.14913`

Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*. `https://arxiv.org/abs/2106.09685`

Hartvigsen, T., Yin, F., Hartman, M., Mahoney, M., and Van Durme, B. (2023). Aging with GRACE: Lifelong model editing with discrete key-value adaptors. In *Advances in Neural Information Processing Systems*, volume 36. `https://arxiv.org/abs/2305.15031`

Träuble, F., Fuchs, F.B., Gelly, S., and Luisier, F. (2023). Discrete key-value bottleneck. In *Proceedings of the 40th International Conference on Machine Learning*. `https://arxiv.org/abs/2305.18353`

Chevalier, A., Sulem, E., and Choshen, L. (2023). Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. `https://aclanthology.org/2023.emnlp-main.632/`

Kim, J.H., Ma, J., Lausen, L., Tan, C., Zhang, J., and Cui, Y. (2024). Compressed context memory for online language model interaction. In *International Conference on Learning Representations*. `https://arxiv.org/abs/2312.03414`

Wang, Y., et al. (2024). MEMORYLLM: Towards self-updatable large language models. In *Proceedings of the 41st International Conference on Machine Learning*. `https://arxiv.org/abs/2402.04624`

Wang, Y., et al. (2025). M+: Extending MemoryLLM with scalable long-term memory. *arXiv preprint arXiv:2502.00592*. `https://arxiv.org/abs/2502.00592`

Du, J., Sun, W., Lan, D., Hu, J., and Cheng, Y. (2025). MoM: Linear sequence modeling with mixture-of-memories. *arXiv preprint arXiv:2502.13685*. `https://arxiv.org/abs/2502.13685`

Zhang, K., Li, Z., Li, Z., Liu, W., and Sato, Y. (2021). Neural routing by memory. In *Advances in Neural Information Processing Systems*, volume 34.

Bini, L., Bohdal, S., Michieli, U., Akata, Z., and Ceritli, E.C. (2025). MemLoRA: Distilling expert adapters for on-device memory systems. *arXiv preprint arXiv:2512.04763*. `https://arxiv.org/abs/2512.04763`

Fountas, N., De Cao, N., Wolff, T., and Peyrard, M. (2025). EM-LLM: Episodic memory for large language models. In *International Conference on Learning Representations.* https://arxiv.org/abs/2407.09450

Liu, X., Zhao, R., Huang, P., Liu, X., Xiao, J., Xiao, C., Xiao, T., Gao, S., Yu, Z., and Zhu, J. (2025). Autoencoding-free context compression for LLMs via contextual semantic anchors. *arXiv preprint arXiv:2510.08907.* https://arxiv.org/abs/2510.08907

Chen, Y., Wang, G., Shang, J., Cui, S., Zhang, Z., Liu, T., Wang, S., Sun, Y., Yu, D., and Wu, H. (2024). NACL: A general and effective KV cache eviction framework for LLMs at inference time. In *Annual Meeting of the Association for Computational Linguistics (ACL).* https://arxiv.org/abs/2408.03675