Memory Transformers: Neural Memory Banks for Textual Memory

Praneeth Vadlapati
Independent researcher
praneethv@arizona.edu
ORCID: 0009-0006-2592-2564

Abstract: This paper introduces Memory Transformers, a class of compact neural systems that store and retrieve textual memories directly in trainable parameter banks. Memory Transformers combine lightweight sequence encoders with parametric memory slots to realize an end-to-end, locally runnable memory service that does not depend on external embedding providers or vector databases. The architecture supports write operations that sculpt slot vectors via short gradient loops, retrieval via cosine similarity augmented by lexical overlap and synaptic-strength weighting, and dynamic capacity growth through neurogenesis-like slot allocation. We describe the design and implementation of two representative instantiations: a transformer-inspired encoder paired with trainable memory slots, and a convolutional encoder with an explicit memory bank. The paper articulates evaluation protocols suitable for qualitative and quantitative assessment, presents illustrative demonstration experiments, and discusses the trade-offs between interpretability, privacy, and retrieval fidelity. We conclude by situating Memory Transformers within broader research on learned memory systems and proposing avenues for future work.

The source code is available at github.com/Pro-GenAI/Memory-Transformer.

Keywords: Artificial Intelligence, AI, Large Language Models, LLM, LLMs, neural memory, transformers, information retrieval, Generative AI

I. Introduction

Human cognition relies on the interplay between transient and consolidated memory representations. Computational systems that mimic elements of this interplay can enable agents and applications to maintain context and recall previously observed information without repeatedly querying large, external resources. Traditional approaches to persistent semantic memory typically separate encoding from storage: encoders produce dense embeddings, and a vector database stores those embeddings for nearest-neighbor retrieval. While effective at scale, that separation introduces dependencies on external services, complex system architecture, and challenges around interpretability and privacy.

A. Disadvantages with current approaches

Contemporary memory systems for language applications typically rely on pre-trained embedding models and dedicated vector search infrastructure. This design yields excellent retrieval quality in many settings but comes with several limitations.

First, system complexity increases. Maintaining separate embedding services and vector databases introduces operational overhead, increases the attack surface, and complicates maintenance. Second, dependency on external or cloud-hosted embedding providers raises privacy and compliance concerns when sensitive data must be encoded and transmitted. Third, the decoupling between embedding and storage loses an opportunity for compactness and co-adaptation: embeddings are static outputs of a pre-trained model, while storing them as-is in a

remote database prevents the storage mechanism from learning to amplify or compress signals most relevant to downstream retrieval tasks.

Another practical limitation is interpretability. When vectors are stored in a third-party database, correlating neural vectors with their originating text and metadata often requires additional bookkeeping. While such metadata can be stored alongside vectors, the stored representations themselves are not amenable to inspection without reloading the exact encoder and reproducing embeddings. Finally, scaling trade-offs and cost can be prohibitive for small teams or users requiring local, low-latency memory handling.

B. Proposed system and its benefits

This paper expands on the design space where encoder and storage are colocated within a single model artifact, with a focus on pragmatic mechanisms for writing, verifying, and retrieving textual memories in parameterized memory banks. We position the proposed system as a tool for scenarios where low-latency, local operation and interpretability outweigh the need for extreme scale.

In this work we propose Memory Transformers, a compact, locally executable family of models that integrate learned encoders with parametric memory banks. Memories are stored by adjusting trainable slot vectors to match encoded textual representations, and retrieval is performed by measuring similarity between a query encoding and the stored slots. Because the stored signal is represented in neural parameters, the system can run offline, provides a straightforward serialization mechanism using native model-weight checkpoints, and keeps human-readable metadata alongside neural representations for inspection.

Memory Transformers offer an alternative architecture in which memory vectors are parameters of a model and therefore live alongside the encoder. Two instantiations illustrate the design space. The first uses a char-level transformer encoder producing normalized representations, and a set of trainable memory slots initialized randomly and optimized when new memories are written. The second uses a convolutional encoder with a distinct memory bank module; it achieves similar goals with a different computational profile and inductive bias.

Writing a memory involves allocating or selecting a free slot and applying a short gradient-based loop that optimizes the slot's parameters to align with the encoded target vector. To improve robustness and usability, the write procedure optionally augments the target with token-level encodings to emphasize salient words. After iterative optimization, a slot is numerically normalized and verified by measuring cosine similarity with the intended target; if the similarity fails to meet a threshold, the system can free the partially written slot and expand capacity (neurogenesis), allocating a new slot and writing the target in a fast direct-path manner. This pragmatic combination of iterative optimization and guarded fallback provides reliable writes while avoiding silent failures.

Retrieval is performed by encoding a query and computing cosine similarities against normalized slots, masking unused slots, and optionally restricting candidates to a particular user or namespace. The raw similarity score is adjusted by a synaptic-strength multiplier associated with each memory item and by a small lexical-overlap boost when words in the query appear in the stored text. Bookkeeping maps between human-readable keys and slot indices, preserving text and metadata for inspection and serialization.

The primary benefits of this approach are operational simplicity, privacy, and interpretability. Because the memory vectors are native model parameters, deployment can be as simple as

shipping a single model checkpoint. There is no need for an external vector DB or cloud embedding service. The kept human-readable text and metadata accompanying each slot facilitate debugging and audit. Furthermore, the system supports local-only deployments, which is valuable when data must remain on-device.

C. New use cases of the system

Memory Transformers are well suited to scenarios where low-latency, local, and interpretable memory is required. For example, personal assistant applications that store short-term and long-term user preferences can operate entirely on-device, eliminating the need to transmit sensitive user data. Research prototypes exploring mechanisms of learned storage, replay, and capacity growth can reproduce and modify the memory dynamics directly by inspecting the slot parameters and their training traces. Edge and embedded systems with limited or no network connectivity can utilize Memory Transformers to retain contextual state across sessions. Finally, rapidly prototyping novel memory management strategies is facilitated by the tight integration between encoder and memory bank; experiments that jointly adapt encoder and storage strategies do not require separate systems to be coordinated.

D. Related work

Research on memory for machine learning has progressed along multiple axes: models that augment sequence processing with external differentiable memory, retrieval-augmented generation that consults large vector stores, and continual learning systems that attempt to preserve past knowledge while integrating new information. Each of these approaches contributes important capabilities, yet practical deployment choices often trade off engineering complexity, cost, and data governance. For practitioners building interactive agents or on-device assistants, the operational burden of maintaining separate embedding endpoints and vector databases can be substantial. At the same time, maintaining explainability and auditability of stored content is increasingly important for compliance and user trust.

The concept of using learned parameters to encode retrieval targets traces to multiple traditions in machine learning, including associative memory models and modern neural memory-augmented architectures. Prior approaches have used attention mechanisms, differentiable key-value stores, and external memory modules for sequence modeling. Memory Transformers align with these traditions but emphasize an operationally simple, locally runnable design that prioritizes human-readable metadata and robust write semantics. Where existing production systems rely on independent embedding pipelines and vector databases, Memory Transformers consolidate responsibilities into a single model artifact, trading some of the scalability advantages of external vector stores for simplicity and privacy.

II. METHODS

A. Architecture

The system consists of an encoder, a memory parameter module, and lightweight utilities for allocation, write optimization, retrieval, and persistence. Two encoder families are presented as reference implementations. The transformer-inspired variant uses a char-level tokenizer, positional embeddings, and a small TransformerEncoder that produces fixed-dimension vectors via mean and max pooling followed by normalization. The convolutional variant uses a stack of one-dimensional convolutions with global pooling followed by an optional projection head when

encoder and memory dimensions differ. Memory banks are implemented as parameters shaped (max_slots, dim) with an associated boolean mask tracking used slots.

B. Write procedure

Writing a new memory first reserves a free slot using a first-free allocation strategy. The text is encoded and optionally augmented by averaging token-level encodings to emphasize keywords. An optimizer with a modest learning rate and momentum runs for a bounded number of iterations to move the chosen slot toward the target vector. A modest multiplicative weight decay is applied to the memory parameters during the write loop to maintain numerical stability. After the iterative loop, the slot is normalized and a cosine-similarity check is performed. If the similarity falls below a configurable threshold, the slot is freed and the memory bank expands by a predetermined growth increment; the new slot is then directly written with the normalized target vector. This strategy provides a robust fallback that avoids silent write failures while enabling compact representation for the common case.

C. Retrieval

Queries are encoded using the same encoder and compared to normalized slots using cosine similarity. Unused slots are masked to prevent spurious matches. Candidate selection can be restricted to a user or namespace via simple bookkeeping. Scores are adjusted by per-memory synaptic-strength parameters and by a lexical-overlap term that modestly boosts results when query tokens appear in memory text. Returned results include both the original human-readable text and the metadata stored with the memory item.

D. Persistence and service wrapper

The system supports saving the encoder, memory parameters, and item metadata using PyTorch serialization, enabling the model to be loaded and resumed in another environment. A minimal FastAPI wrapper exposes endpoints for adding memories, searching, and deleting all memories for a user, illustrating how the model can be deployed as a lightweight service.

E. Evaluation protocols

To assess retrieval quality and operational properties we propose a multi-part evaluation protocol. First, synthetic retrieval tasks evaluate whether the system can store and retrieve short facts and paraphrases under constrained vocabulary and sequence length. Second, robustness tests measure write success rate under varying write-iteration budgets and learning rates. Third, privacy and deployment analyses compare the system's operational footprint and data locality against standard embedder+vectorDB pipelines. Qualitative case studies on representative conversational traces demonstrate interpretability and debugging advantages.

III. RESULTS

This paper focuses primarily on describing the architecture, implementation, and evaluation protocols for Memory Transformers. To illustrate typical behavior, we executed demonstration experiments using small in-memory configurations that exercise write and query operations. The following subsections summarize the observed behaviors and methodological insights from those demonstrations.

A. Demonstration experiments

In demonstration experiments the system readily stored short textual facts and retrieved them using semantically related queries. When multiple memories about a single user contained

overlapping topical vocabulary, queries that included core topical words returned those memories with the recommended lexical-overlap boost, and synaptic-strength values influenced ranking as expected. These experiments were intentionally small-scale and intended to exercise the end-to-end write and query lifecycle rather than to provide exhaustive benchmarks.

B. Write verification and robustness

Write verification mechanisms effectively prevented silent failures in these demonstrations. Situations that would otherwise produce low similarity after a bounded number of optimization iterations were resolved by the grow-and-rewrite fallback. In that fallback the partially written slot is freed and the bank grows by a configured increment; a new slot is then assigned and the normalized target vector is stored directly. This approach produces a pragmatic balance: most writes succeed through modest local optimization while adversarial or difficult examples are preserved by controlled capacity growth.

C. Qualitative encoder comparison

Qualitatively, the transformer-based encoder produced robust representations for relatively short sequences and phrases, benefiting from positional context and subtoken aggregation. The convolutional encoder provided a computationally inexpensive alternative with competitive retrieval behavior on short sequences. Both variants exhibited the interpretability advantage of maintaining original text and metadata alongside parameterized representations, which simplified result inspection and debugging.

D. Reporting and reproducibility guidance

We intentionally do not report aggregate numeric benchmarks in this paper because reliable quantitative comparisons depend on standardized datasets and controlled baselines. Instead we offer detailed evaluation protocols and guidance so that empirical studies can be reproduced and extended by independent researchers. The proposed protocols cover recall@k experiments, write success rate measurement, ablation of token-emphasis strategies, and deployment-oriented metrics such as model size and inference latency on representative hardware.

IV. DISCUSSION

Memory Transformers occupy a pragmatic point in the design space between fully externalized embedding pipelines and monolithic, end-to-end learned retrieval systems. Their principal advantages are operational simplicity and privacy: a single serialized model file contains both encoder and memory state, and human-readable text is preserved within the checkpoint for debugging and audits. These properties make the architecture attractive for on-device assistants and privacy-conscious deployment scenarios.

However, the design also has limits. Because the memory bank resides in model parameters, scaling to extremely large memory sizes will increase checkpoint sizes and memory consumption during inference. External vector databases remain more appropriate for very large-scale retrieval needs where distribution, sharding, and vector compression are central concerns. Additionally, because slot parameters are learned through gradient updates during write operations, concurrent or high-throughput write workloads require careful engineering to avoid contention or performance degradation; batching and background write threads are viable mitigations but are beyond the minimal implementations discussed here.

Another area for future work concerns the co-adaptation of the encoder and the memory bank. In the present design writes are localized and do not retrain the encoder in the common case;

research that jointly adapts both encoder and memory through continual learning paradigms may yield improved long-term storage fidelity but will need mechanisms to avoid catastrophic forgetting. Similarly, exploring more sophisticated allocation and compaction strategies, including semantic compaction and slot merging, could improve capacity efficiency.

The architecture also opens interesting avenues for interpretability research. Because each memory is associated with explicit text and metadata and the slot parameters are accessible, researchers can study how textual content maps to geometric structure in parameter space, trace the evolution of a slot during write iterations, and experiment with explainability techniques that project slot vectors back into token space.

V. CONCLUSION

Memory Transformers present a compact, transparent approach to learned textual memory that is appropriate for local, low-latency, and privacy-sensitive applications. By embedding storage within model parameters and preserving human-readable metadata, the approach simplifies deployment and enables inspection and reproducibility. The combination of iterative writes with guarded capacity growth yields a robust operational profile that works well for many practical use cases. We provide practical implementations and evaluation protocols to facilitate adoption and further research. Future work will investigate scaling strategies, encoder-memory co-adaptation, and richer memory management policies to broaden the applicability of learned parametric memories.

REFERENCES

[1] <To be added>