ChatGPT

Always-On LLM with Human-Like Memory: Multi-Tier Memory Windows, Forgetting, and Cold Storage

Abstract

Large Language Models (LLMs) struggle with long-term consistency due to fixed context window limits, causing them to “forget” earlier information in extended conversations [1]. We propose an **always-on memory architecture** for LLM-based chat agents inspired by human memory systems. Our design introduces a stacked, multi-tier memory comprising: (1) a **perception window** for short-term context, (2) an **episodic memory** with controlled forgetting for intermediate-term recall, and (3) a **cold storage** long-term memory. The model dynamically shifts information between these layers: recent dialogue resides in the perception window, important interactions are distilled into episodic memory (with less salient details gradually forgotten), and older knowledge is archived in cold storage. This human-inspired memory management enables the LLM to maintain continuity and emotional coherence over prolonged dialogues, remembering user preferences and context across sessions. Preliminary evaluations indicate that our approach preserves long-term facts and character consistency far beyond the base model’s context capacity, while scaling efficiently. This work demonstrates that **memory windows, dynamic forgetting, and long-term storage** can be combined to imbue LLMs with more human-like persistent memory, paving the way for more consistent, personalized, and scalable always-on AI assistants.

Introduction

Modern LLMs excel at generating contextually relevant responses, but they lack a persistent long-term memory. In standard chat deployments, the model’s awareness is limited to a fixed-length recent context (e.g. a few thousand tokens), meaning earlier conversation details eventually drop out of scope [1]. This leads to *conversational drift*, where the AI forgets important facts (like the user’s name or emotional state) as a dialogue grows, undermining consistency and user experience. Humans, by contrast, possess multi-tiered memory: a fleeting sensory/working memory, a stabilizing short-term/episodic memory, and long-term memory stores. Inspired by these cognitive mechanisms, we propose an **always-on LLM memory system** that more closely mirrors human memory organization.

Our approach features a **tiered memory architecture** that allows an LLM to retain and retrieve information over indefinite interactions. The architecture is built around three key levels: a *perception (working) memory window* for immediate context, an *episodic memory* that accumulates and selectively forgets details over time, and a *long-term cold storage* for permanently storing knowledge and past events. Information flows between these layers in a manner analogous to human memory consolidation – new experiences are initially in the focus of attention, then gradually distilled into durable form while less relevant details fade [2] [3]. By managing memories in this way, the system maintains a coherent narrative of the conversation and the user’s persona over time, without overloading the LLM’s context. We focus on how this design improves **long-term consistency**, preserves *emotional coherence* (e.g. remembering the user’s feelings or the agent's personality traits), and ensures scalability via controlled memory growth. The following sections describe the architecture and operations of the memory system, followed by preliminary results from a prototype implementation.

System Architecture

Figure 1: The Always-On LLM memory architecture, with three stacked memory modules. New information enters the Perception/Working Memory (limited context window). Important details are consolidated into Episodic Memory (with dynamic forgetting of low-relevance details), and older knowledge is archived in Cold Storage (long-term memory). Arrows indicate the flow of information: recent inputs are saved to episodic memory and eventually archived, while relevant memories can be retrieved from lower layers back into the active context.

At a high level, the system extends a base LLM with an external memory manager that orchestrates the three memory layers. Each layer serves a distinct role:

- **Perception Window (Working Memory):** This is a short-term memory buffer akin to the human *working memory*. It consists of the recent dialogue context provided to the LLM for the current turn. The window is of limited capacity (e.g. a few thousand tokens), ensuring the model focuses on the most pertinent recent information. Just as humans can only keep a few items in mind at once, the perception window holds the latest user query, the assistant's last responses, and any immediately relevant facts fetched from deeper memory. This design prevents overload and keeps inference efficient.

- **Episodic Memory (with Dynamic Forgetting):** The episodic memory module stores a running summary of the conversation and recent "episodes" – coherent chunks of interaction or important events. Whenever the perception window fills up or a topic shifts, the system **encodes that episode into a memory entry**, often by summarizing it in natural language. Episodic memory thus provides an intermediate-term record of dialogue that persists across many turns, unlike the short-lived perception window. However, to avoid unbounded growth, the episodic memory employs *controlled forgetting*: less useful or stale details are gradually pruned or abstracted. For example, the agent may retain that *"the user's hometown is Paris"* but forget the exact wording of a greeting 50 turns ago. This is implemented by periodically compressing older entries (merging or summarizing multiple events) and discarding information deemed low-importance via a decay function or an attention-based heuristic. Such dynamic forgetting is analogous to human memory consolidation – important memories are retained with clarity, while trivial specifics fade over time.

- **Cold Storage (Long-Term Memory):** This is a durable repository for all facts and experiences that have been distilled from episodic memory. Cold storage can be implemented as a **vector database or knowledge base** that indexes memory entries by semantic embeddings. It serves as the LLM's long-term memory, conceptually similar to the human long-term memory where information can be stored indefinitely. Entries in cold storage are typically high-level summaries or **"memory balls"** representing aggregated knowledge of earlier episodes. A *memory ball* encapsulates the essence of an episode (or a cluster of related episodes) in a compact form – for instance, a synthesized statement: *"Last year, the user went on a hiking trip and felt accomplished reaching Mt. Fuji's summit."* These memory balls may also store metadata (timestamps, related topics, emotional tone) and have associated embeddings for efficient retrieval. Cold storage allows the system to **archive** knowledge that is not immediately needed, keeping the active memory lean [3], yet making it possible to reactivate those memories when contextually relevant in the future.

Crucially, the architecture enables **bidirectional information flow** between layers. New inputs and events flow downward: each user query and LLM response is logged in episodic memory (often after summarization) and eventually in long-term store. Conversely, relevant background knowledge flows upward: when the LLM is processing a new user message, the memory manager will query episodic and long-term stores to retrieve any pertinent information and inject it into the perception window context. This ensures that even information from hundreds of turns ago (or from prior sessions) can influence the current response, if it's relevant (e.g. "recall the user's birthday mentioned last week"). A retrieval scoring mechanism (using semantic similarity, recency and importance cues) determines which memory records to fetch [4]. By combining recency (favoring recent episodes) with semantic relevance and learned importance weights (favoring significant facts like user-provided key information), the system mimics human recollection – recent memories are generally fresher, but an important older memory can resurface when triggered by context.

Memory Operations

To coordinate the above components, the system defines a set of memory operations that occur at each interaction cycle:

1. Context Assembly: When a user prompt arrives, the memory manager first assembles the LLM's context (the perception window). It starts with the recent dialogue (e.g. last few turns) and then augments this with any additional facts retrieved from episodic memory and cold storage. A *retrieval check* [5] uses the incoming user query as a cue to search for related memory: e.g. if the user asks "Can you remind me what my favorite book is?", the system will look up that detail from the long-term store. Retrieved memory summaries are then inserted into the prompt (e.g., as a system message or appended context) so that the LLM is aware of them when formulating its answer. We call this **contextual recall** – pulling knowledge from memory layers into the working context.

2. Response Generation: The LLM, now armed with an enriched context window, generates its response to the user. Thanks to memory augmentation, the response can incorporate older knowledge (for instance, correctly remembering the user's favorite book) despite that information being far outside the base model's normal context window. The model's generation is guided by all accessible memory, which helps maintain continuity in topics and emotional tone.

3. Memory Update (Consolidation): After the LLM produces a response, the new interaction (user query + assistant answer) is processed into the memory system. The perception window content that is no longer fresh (e.g. earlier turns) gets encoded into episodic memory. This often involves summarization: the system might compress the latest exchange into a concise note (e.g., *"User asked about travel plans; assistant provided packing tips."*). This summary is stored as a new **episodic memory entry**. If the episodic memory is becoming too large or if a sufficient number of new entries have accumulated, a *forgetting routine* triggers: older episodic entries may be merged or summarized into a higher-level memory. For example, after 20 turns, the system could create a synopsis of the entire discussion so far and replace the detailed turn-by-turn log with that synopsis (similar to the "reflection" mechanism noted in other LLM memory studies [2]). This **temporal compression** ensures the episodic store remains concise, retaining only salient information. The discarded details are not entirely lost – before removal, any key facts they contained (e.g. a decision the user made, or a change in emotional state) are identified and included in the summary or logged separately as knowledge.

4. Long-Term Archival: The episodic summaries eventually flow into cold storage for long-term retention. The system may continuously or periodically offload episodic entries older than a certain age or beyond a memory count threshold into the long-term memory database. In cold storage, they become memory balls – indexed, searchable units of knowledge. The *memory ball model* entails that each archived memory is **self-contained**: it has a summary (for quick reference), vector embedding (for similarity search), and a link or key to more detailed content if needed. This allows efficient retrieval: the system typically retrieves just the summary (the "core" of the ball) to inform the conversation, but if the situation demands finer details, it can choose to fetch more specifics (expanding the memory ball). In essence, the memory balls serve as compressed capsules of past experience that the agent can draw on selectively.

5. Forgetting and Adaptation: The memory system also supports *active forgetting* and update of long-term info. If information in cold storage becomes irrelevant or outdated (e.g. the user corrects a fact or changes a preference), the system can mark those entries as deprecated or adjust them. This ensures the agent doesn't cling to stale or incorrect memories. Likewise, if the user hasn't mentioned a certain long-term detail in a very long time, the system might eventually archive it to an even "colder" storage or reduce its retrieval priority – analogous to humans having memories fade unless revisited. This controlled forgetting is crucial for scalability, preventing infinite accumulation of memory that could slow retrieval or confuse the model with too much trivia.

Through these operations, the architecture manages to keep the working context relevant and sized for the LLM's limitations, while maintaining a rich store of historical knowledge in the background. Our design effectively creates a **memory-managed conversation loop**: each turn, recall what's needed, respond, and remember what happened – continuously balancing remembering and forgetting, much like a human conversational partner.

Preliminary Results

To validate the proposed memory architecture, we implemented a prototype on top of an LLM-based chatbot and conducted preliminary tests in extended conversation scenarios. Even without fine-tuning the LLM itself, the memory augmentation yielded notably improved long-term coherence. In a simulated personal assistant dialog spanning **hundreds of turns** over several days, the always-on memory system consistently recalled previously saved facts and context: for example, the assistant correctly remembered the user's meal preferences and an upcoming trip that were mentioned in early conversation (far outside the base model's raw context limit). The agent also maintained **emotional continuity** – when the user started the week feeling anxious about an event, the assistant later subtly checked back on that issue and adjusted its tone to be supportive, showing awareness of the user's emotional state carried through memory. Such behavior was absent in a baseline chat without our memory system, which would often forget the user's feelings or repeat questions.

Quantitatively, we observed that the memory-enhanced LLM could refer back to facts introduced **>1000** turns earlier with appropriate prompting, whereas the baseline lost track after ~100 turns (consistent with the fixed token window). The dynamic forgetting mechanism prevented information overload: the episodic memory summaries remained on average under 1000 tokens despite the conversation's length, and retrieval from the vectorized long-term store remained fast (sub-second for a database of thousands of entries). We also noted improved **consistency in persona** – the assistant didn't contradict its earlier statements about itself or the user, thanks to retrieving relevant self-descriptions and context from memory. One illustrative result was in a story-style chat: the agent introduced a minor character early on; later in the story, that character reappeared correctly with remembered attributes, whereas a memory-less model forgot the character and introduced inconsistencies.

These early results, while qualitative, demonstrate the potential of human-inspired memory management for LLMs. The agent with always-on memory delivered a more **stable and engaging interaction**, as reported by testers, who noted the chatbot "felt more like it remembers me as a person." There were fewer instances of the bot repeating itself or asking for information twice. Importantly, the overhead of the memory system was manageable: context assembly with retrieval added only a modest increase in latency, and the summarization/archival processes ran in the background without interrupting the live interaction. This suggests the approach is scalable to longer deployments.

Conclusion

We presented an **Always-On LLM memory architecture** that integrates a *perception window*, *episodic memory with dynamic forgetting*, and *cold storage* into a unified system, taking inspiration from human memory processes. By layering short-term and long-term memory modules, an LLM-based agent can engage in lengthy, evolving conversations while retaining important information and shedding irrelevant detail. This leads to improved consistency (factual and personality-wise) and a more personalized, context-aware user experience over time. The model effectively learns *when to remember* and *when to forget*, addressing the classical context-length limitation of LLMs in a scalable manner.

Our preliminary implementation confirms that even simple strategies (such as summarizing recent dialogues and using vector similarity search for retrieval) can dramatically extend an LLM's effective memory. The agent remains **emotionally and contextually coherent** over much longer interactions than a standard model. We believe this human-like memory framework is a critical step toward persistent AI assistants and autonomous agents that can truly accompany a user over time, accumulating experiences.

Moving forward, there are several avenues to refine this approach. We plan to explore more advanced retrieval algorithms that weigh memory relevance with learned importance (possibly training the LLM to decide what to recall) [4], and to investigate the optimal granularity for memory balls (balancing detail vs. brevity). Ensuring the **accuracy of recalled memories** is vital – we will implement verification steps to prevent the LLM from conflating or hallucinating details when retrieving from long-term memory. We are also interested in the privacy implications of long-term memory in chatbots, and how to give users transparency and control over what the AI remembers (similar to recent efforts in controllable AI memory [6] [1]). In conclusion, equipping LLMs with an always-on, human-inspired memory not only enhances their performance on extended tasks, but also moves us closer to AI that can **learn and adapt continuously** in a safe and aligned way. The synergy of memory windows, forgetting mechanisms, and cold storage offers a promising blueprint for building LLM systems that remember *what matters* in the long run.

---

[1] [6] Remembering and forgetting with ChatGPT - A guide for beginners
https://www.uintent.com/case-studies-und-blog/remembering-and-forgetting-with-chatgpt---a-guide-for-beginners

2 How LLM Memory works : r/OpenAI
https://www.reddit.com/r/OpenAI/comments/1aqksc3/how_llm_memory_works/

3 5 Modelling for Reuse: Smaller Elements are FAIR | John O'Gorman
https://www.linkedin.com/posts/john-o-gorman-b97ab2_modelling-for-reuse-smaller-elements-are-activity-7258197294087352321-aX6q

4 [2304.03442] Generative Agents: Interactive Simulacra of Human Behavior
https://ar5iv.labs.arxiv.org/html/2304.03442