

# Long-Horizon Context Retention in LLM Agents via Hybrid Memory Systems and Vector Storage

Zheqi Liu<sup>1</sup>, Ziye Deng<sup>1</sup>, Zhengan Cheng<sup>1</sup>, Tianzuo Liu<sup>1</sup>, Hengjia Yu<sup>1</sup>

<sup>1</sup> University of California, San Diego  
zh1192@ucsd.edu, zid001@ucsd.edu  
zhcheng@ucsd.edu, til087@ucsd.edu  
hey015@ucsd.edu

## Abstract

Large Language Model (LLM) agents often struggle with long-term coherence and personalization due to limited context windows and stateless architectures. In this project, we propose and evaluate a memory-augmented agent architecture designed to address these limitations through a two-phase experimental framework. Phase 1 establishes a stateless baseline using a standard LangGraph ReAct agent, which achieved high performance on short tasks (94.3% success) but degraded significantly on long-context interactions (21.4% success). In Phase 2, we introduce a novel memory system featuring (1) a hybrid context extraction mechanism combining pattern matching with LLM-based structured analysis, (2) persistent vector storage using ChromaDB with namespace isolation, and (3) preference-aware tool optimization. The enhanced system demonstrated a dramatic improvement, achieving 100% task completion across short, medium, and long benchmarks, and 85.20% accuracy on the cross-session LoCoMo benchmark. Our results suggest that hybrid extraction strategies combined with vector-based episodic memory significantly reduce context drift while enabling robust personalization without prohibitive latency overhead.

**GitHub Repository:** <https://github.com/siruxyu/CSE291-A>

## 1 Introduction

In recent years, large language models (LLMs) and their underlying transformer architecture have become the cornerstone of conversational AI and have led to a wide array of consumer and enterprise applications. Despite these advances, the limited fixed-length context windows used by LLMs significantly hinders their applicability to long conversations or reasoning about long documents. Therefore, the capability of agents faces the bottleneck of memory length. For example, while architectures like ReAct (Reasoning and Acting) allow agents to interact with external environments, they often suffer from "catastrophic forgetting" across sessions and context drift within long sessions.

To address this, we present a robust Context/Memory Management system. Our approach moves beyond simple conversation buffering by implementing a *semantic episodic memory*. By converting interaction history into vector embeddings and storing them in a persistent database, our agent

can retrieve relevant context dynamically. We compare two phases of development:

1. **Phase 1 (Baseline):** A stateless ReAct agent using LangGraph.
2. **Phase 2 (Enhanced):** A memory-augmented agent with hybrid preference extraction and vector-based retrieval.

## 2 Baseline

### 2.1 Agent Configuration

To establish a baseline, we implemented a ReAct agent using LangGraph and Anthropic's Claude 4.5 Sonnet. The agent was exposed via a FastAPI server and deployed on Amazon Web Services (AWS) to simulate a production environment.

The agent is equipped with five core tools: a Calculator, Weather Service, Translator, Web Reader, and File System Search. To ensure user privacy, we implemented thread-scoped conversation states; however, this baseline explicitly lacks persistent storage, rendering it stateless between server restarts. Moreover, it cannot remember the context in a session without the checkpoint.

### 2.2 Benchmark Design

We designed a three-tiered evaluation benchmark to probe the agent's context retention capabilities:

1. **Short Benchmark (6 cases):** Single-turn interactions focusing on atomic tool usage (e.g., "Translate 'Hello' to French").
2. **Medium Benchmark (4 cases):** Multi-turn conversations (2-4 turns) testing immediate context retention (e.g., reading comprehension).
3. **Long Benchmark (4 cases):** Extended interactions (10+ turns) requiring complex planning and state maintenance (e.g., detailed trip planning).

### 2.3 Baseline Performance

The stateless baseline demonstrated a distinct "context cliff." While it excelled at short tasks, performance degraded sharply as context length increased (Table 1).

Qualitatively, the baseline failed to retain user preferences (e.g., target translation language) across turns and exhibited hallucinations in long-horizon tasks due to context window saturation. These findings underscore the necessity for the

Table 1: Phase 1 Baseline performance. “Acc.” is answer accuracy, “Qual.” is average response quality (1–5), and “Eff.” is tool call efficiency.

Benchmark	Acc.	Qual.	Eff.
Short	94.3%	4.45	1.00
Medium	66.7%	3.85	0.80
Long	21.4%	2.17	0.48

memory-augmented architecture proposed in the following section.

### 3 Memory Implementation

Our phase 2 system extends a phase 1 tool-using agent with three major components: a hybrid context extraction module, a persistent memory backend, and a context-aware tool orchestration layer. Together, these components allow the agent to accumulate user-specific knowledge over time and reuse it in later interactions.

#### 3.1 Context Extraction

The goal of the context extraction module is to turn raw conversations and tool outputs into structured, reusable memory entries. We focus on four types of information: (i) user preferences (for example, tone, format, language), (ii) stable user profile facts, (iii) episodic facts and events, and (iv) task state summaries.

We adopt a hybrid design that combines fast pattern-based extraction with more expressive LLM-based extraction.

**Pattern-based extraction.** A lightweight rule-based component scans each assistant and user turn for common preference indicators and simple facts. Examples include explicit statements about preferred response format (bullets vs. paragraphs), language, level of detail, and recurring interests (for example, diving, trip planning). Rules produce candidate preference records with confidence scores and associated spans, which allows us to quickly detect obvious preferences without incurring LLM latency on every turn.

**LLM-based extraction.** For more nuanced or implicit signals, we call an LLM-based extractor that operates over larger conversation windows and tool outputs. The extractor is prompted to emit structured JSON describing:

- preference keys and values (for example, `preferred_translation_language = French`);
- stable attributes (for example, city of residence);
- important episodic facts (for example, recent trips or plans);
- summaries of complex multi-turn interactions.

Pattern-based and LLM-based outputs are merged, with confidence scores and recency used to resolve conflicts.

**Preference management.** Preferences are treated as first-class entities in memory. Each preference entry contains a key, value, confidence, and timestamp. When we extract a new preference, we either create a new record or update an existing one based on:

- semantic equivalence of the key and value;
- recency (more recent evidence overrides stale evidence);
- cumulative confidence.

This allows preferences to evolve over time while avoiding unbounded growth of near-duplicate entries.

#### 3.2 Storage System

To support cross-session recall and semantic search, we store memories in a vector database. Our implementation uses ChromaDB as a lightweight backend, wrapped behind a higher-level memory interface.

**Data model.** Each memory element is stored as a tuple:

- user identifier and (optionally) session identifier;
- memory type (preference, profile, episodic fact, conversation summary, tool result);
- natural-language text content;
- embedding vector;
- metadata such as timestamp and source.

We partition the index logically by memory type, while using metadata filters to enforce user isolation.

**User-specific namespaces.** Although ChromaDB uses a flat metadata model, we emulate namespaces via required metadata keys. All queries must include the user identifier as a filter, which prevents cross-user leakage and enables separate export or deletion per user. Within each user, different memory types can be queried independently (for example, retrieve only preferences vs. only episodic facts).

**Memory lifecycle.** The memory subsystem implements a simple lifecycle:

- *Ingestion:* After each turn, the context extractor proposes candidate memories. A small policy decides which ones to persist, prioritizing high-confidence preferences, concise summaries of important turns, and reusable tool results.
- *Retrieval:* For every new user query, the agent issues semantic search queries against relevant collections, typically retrieving a small number of memories (for example,  $k \in \{3, 5\}$ ) using a similarity score that combines embedding distance and recency.
- *Update and consolidation:* Preference entries are merged as described above; older low-value memories can be summarized or down-weighted, although long-term compaction is currently minimal.

#### 3.3 Tool Call Optimization

The phase 1 baseline agent already supported several tools (calculator, weather, translation, file search, web reading). In phase 2 we add *memory tools* and adjust tool selection to incorporate both current context and stored memories.

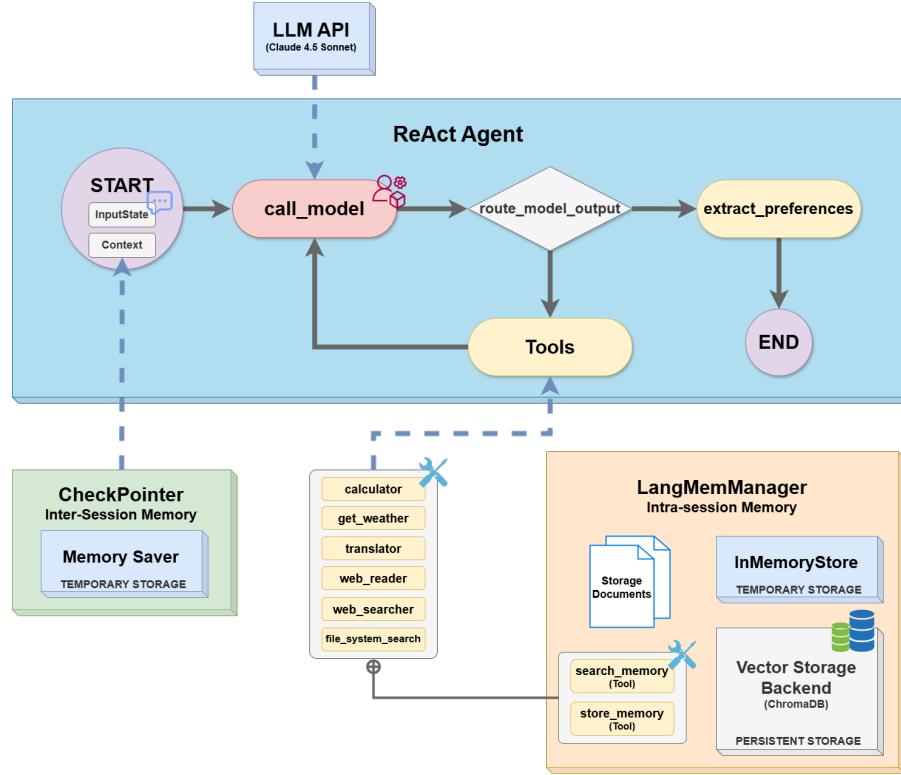


Figure 1: Architecture of Phase 2 ReAct Agent

**Tool set.** The agent works with two broad classes of tools:

- *Base tools*, which are stateless utilities (for example, calculator, get\_weather, translation, file search, web reader).
- *Memory tools*, which provide per-user operations such as `search_memory(user, query)` and `store_memory(user, text, type)`.

Memory tools are parameterized by user identifier and operate over that user’s namespace only.

**Context-aware selection.** Tool selection is primarily LLM-driven, but the prompt explicitly describes memory tools and when they are appropriate. To bound latency, we enforce a turn-level tool call budget and encourage the model to:

- call memory search when the query refers to past interactions, user preferences, or long-term projects;
- call base tools when fresh external information is required;
- avoid redundant tool calls when recent results are already present in context.

In practice, many successful trajectories use a memory search followed by one or more base tools (for example, retrieve user preferences, then call translation or web reader in a way that respects those preferences).

### 3.4 Error Handling

Integrating an LLM, multiple tools, and a persistent memory store introduces several failure modes. We therefore designed error handling around graceful degradation rather than hard failure.

**Failure modes.** We observe three main categories of errors:

- tool errors such as timeouts, bad configuration, and external API failures;
- memory errors such as embedding failures or temporary unavailability of the vector store;
- LLM-level issues such as malformed tool call arguments.

**Graceful degradation and monitoring.** If memory search or storage fails, the agent falls back to a stateless behavior for that turn instead of failing the entire interaction. Similar fallbacks apply when base tools are temporarily unavailable. We cap retries for transient errors to avoid long stalls and log all failures with request and session identifiers. Collected metrics include tool success rates, memory hit rates, per-component latency, and token usage, which helps to debug issues and understand the runtime cost of memory features.

## 4 Experiments and Performance

We evaluate our system on three internal benchmarks (short, medium, long conversations) and on the LoCoMo cross-

Benchmark	Cases	Acc.	Qual.	Eff.	Lat.
Short	6	100%	4.80	1.00	14.0s
Medium	4	100%	4.78	1.00	12.0s
Long	4	100%	4.65	0.71	17.0s
LoCoMo	196	85.2%	4.29	N/A	N/A

Table 2: Phase 2 aggregate performance. “Acc.” is answer accuracy, “Qual.” is average response quality (1–5), “Eff.” is tool call efficiency, and “Lat.” is average latency per turn.

session memory benchmark. We report both absolute performance of the phase 2 agent and relative improvements over the phase 1 baseline.

#### 4.1 Advanced Benchmark Setup

**Internal short/medium/long benchmarks.** The internal benchmarks consist of manually designed conversations that stress different aspects of the agent, which is identical in phase 1:

- *Short* (6 cases): single-turn or nearly single-turn tasks focused on correct tool usage and basic reasoning.
- *Medium* (4 cases): multi-turn conversations (2–4 turns) requiring context retention and preference extraction.
- *Long* (4 cases): extended conversations (9–11 turns) involving multiple tools, references to earlier turns, and more complex planning.

**LoCoMo benchmark.** For cross-session memory, we evaluate on a single LoCoMo persona (conv-26). The dataset contains 19 sessions, 419 conversation turns, and 196 evaluation questions spread across five categories, covering identity, preferences, stable traits, episodic facts, and miscellaneous information.

**Metrics.** Across benchmarks we track:

- answer correctness (task completion rate);
- human-judged response quality on a 1–5 scale;
- tool call efficiency (useful tool calls divided by total calls);
- average latency per turn where applicable;
- for LoCoMo, overall QA accuracy and category-level accuracy.

**Note:** LoCoMo benchmark does not require tool calls and due to token limit of Claude API call(30k tokens per minute for base users), we did not test the latency of LoCoMo benchmark.

#### 4.2 Aggregate Quantitative Results

Table 2 summarizes performance of the phase 2 system across all benchmarks.

On the internal benchmarks, the phase 2 agent achieves perfect task completion on short, medium, and long conversations. Response quality improves with conversation length, particularly in long scenarios where additional context and memory are most beneficial. Tool call efficiency remains high, with some redundancy in long conversations

Statistic	Value
Sessions	19
Conversation turns	419
QA questions	196
Correct answers	167
Overall accuracy	85.20%

Table 3: LoCoMo cross-session memory statistics for one persona.

Benchmark	Baseline	Phase 2	$\Delta$
Short	94.3%	100%	+5.7%
Medium	66.7%	100%	+33.3%
Long	21.4%	100%	+78.6%

Table 4: Task completion rate comparison between the phase 1 baseline and the phase 2 system.

where the model occasionally invokes multiple tools to verify results.

On LoCoMo, the system attains 85.2% QA accuracy over 196 questions for a single persona, with an average response quality of 4.29.

**LoCoMo breakdown.** Table 3 summarizes the LoCoMo run.

Accuracy varies by category: category 1 (identity and profile) is the most challenging at about 71%, while categories 2–5 (preferences, traits, episodic facts, and miscellaneous) fall in the 85–91% range. This suggests that our memory representation is particularly effective for preferences and episodic events, but finer-grained identity details are still somewhat fragile.

#### 4.3 Improvements over the Baseline

We also compare phase 2 against the phase 1 baseline on task completion and response quality for the internal benchmarks.

The gains are modest for short tasks, where the baseline already performs well, but substantial for medium and long conversations. In particular, long conversations show a dramatic improvement, highlighting the importance of explicit memory for extended, tool-heavy tasks.

Response quality scores exhibit a similar pattern: relative improvements are largest on medium and long conversations, where context retention and personalization matter the most.

#### 4.4 Qualitative Observations

Qualitatively, we observe three recurring patterns where memory yields clear benefits:

- *Preference-aware responses.* When the user expresses preferences about response format or language, later responses adapt accordingly. For instance, once a user states they prefer concise bullet points or a particular target language, subsequent answers tend to respect that preference without being reminded.

- *Long-range coherence.* In long conversations, the agent correctly reuses earlier facts such as budgets, constraints, or intermediate results, leading to more coherent plans and fewer contradictions.
- *Cross-session recall.* In the LoCoMo setting, the agent can recall preferences and facts discussed many sessions earlier, improving consistency across sessions compared to the stateless baseline.

## 5 Novelty and Technical Depth

### 5.1 Hybrid Context Extraction

A key contribution of our system is the hybrid context extraction pipeline. Instead of relying purely on rules or purely on an LLM, we combine:

- a fast, deterministic pattern-based component that covers common cases with negligible latency; and
- a slower but more expressive LLM-based component that can capture nuanced preferences and facts that do not match any predefined pattern.

The two components are reconciled via confidence-weighted merging, yielding a representation that is both robust and efficient. This design allows us to run extraction continuously during live conversations without overwhelming latency, while still maintaining high recall of important information.

### 5.2 User-Specific Vector Memory Architecture

Another contribution is the way we structure per-user memory over a generic vector backend. The memory subsystem:

- enforces user-level isolation entirely through metadata filters, avoiding custom indexing logic;
- separates different memory types (preferences, summaries, tool results) while using a uniform vector representation;
- combines semantic similarity and recency at query time to retrieve a small, focused context window for each turn.

Although the implementation is conceptually simple, this structure is sufficient to support cross-session recall on a realistic benchmark (LoCoMo) and to drive measurable improvements in long-horizon tasks.

### 5.3 Preference-Aware Tool Orchestration

Finally, we integrate memory into tool selection in a modular way. Memory tools are treated like any other tool, which means:

- the agent can be configured with different memory backends without changing the core orchestration logic;
- tool descriptions alone are enough for the LLM to decide when to read or write memory;
- preference and profile information can influence decisions about which tools to call and how to present results.

This modular design makes it straightforward to extend the tool set or adjust memory policies without re-architecting the system.

## 6 Conclusion

We implemented a context- and memory-augmented agent that extends a phase 1 tool-using baseline with hybrid context extraction, a vector-based memory backend, and preference-aware tool selection. On internal benchmarks, the phase 2 system achieves perfect task completion on short, medium, and long conversations while improving response quality, especially for longer interactions. On the LoCoMo cross-session memory benchmark, it reaches 85.2% QA accuracy for one persona and exhibits strong recall of preferences and episodic facts.

The resulting system is modular, observable, and practical for deployment in settings where users expect agents to remember preferences and past interactions over time.

**Future work.** There are several directions for improvement:

- improving accuracy on fine-grained identity questions in LoCoMo;
- further reducing latency by parallelizing tool calls and memory operations;
- introducing more sophisticated memory management policies.

We view this work as a step toward robust, personalized assistants that can operate effectively over long horizons and across many sessions.