Metaphors for Memory: Charting a Design Space of AI Memory Tools and Interfaces

Munyeong Kim
Montréal HCI
Université de Montréal
Montréal, Quebec, Canada
kim.munyeong@umontreal.ca

Michalis Famelis
DIRO
Université de Montréal
Montréal, Quebec, Canada
famelis@iro.umontreal.ca

Ian Arawjo
Montréal HCI
Université de Montréal
Montréal, Quebec, Canada
ian.arawjo@umontreal.ca

Abstract

AI memory is becoming central to AI systems that aim to support personal and professional work. Yet, designing interfaces for AI memory is not well understood. How should we design for user interaction with AI memories—for instance, what kinds of operations might users want to perform on memory, either now or in the future? We survey the design of tools and interfaces around "AI memory" to chart a design space of common design patterns, architectures, and operations, and identify dominant metaphors and gaps in the space. Then, we apply generative metaphorical design to expand the design space for AI memory, exploring less-dominant metaphors of software version control, Zettelkasten, requirements, personal diaries, community archives, cultural probes, and science fiction. Our work offers rich opportunities for gaps and emergent needs that future interfaces for AI memory might address.

CCS Concepts

• **Computing methodologies** → **Intelligent agents**; • **Human-centered computing** → *Natural language interfaces*; **HCI theory, concepts and models**; *Collaborative interaction.*

Keywords

memory, human-AI interaction, language models, design space, AI agents

ACM Reference Format:
Munyeong Kim, Michalis Famelis, and Ian Arawjo. 2026. Metaphors for Memory: Charting a Design Space of AI Memory Tools and Interfaces. In *Designing Interactive Systems Conference (DIS '26), June 13–17, 2026, Singapore, Singapore.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3800645.3812979

1 Introduction

Agentic AI systems, characterized by a large language model coupled with the ability to call programmatic functions ("tools"), increasingly have "memory" components. However, the consistency of "AI memory," both in its implementation and its linguistic definition, are in flux. "AI memory" is now used to refer to everything from in-context storage of past interactions, to pre-written user directives injected into system prompts, to databases of user preferences [75], to Markdown files stored on the file-system that an agent can retrieve, read, and modify [4]. Across these varied implementations, memory serves common purposes—chiefly to *remember* intent, preferences, and details of past interactions, whether with the user, or of agents themselves.

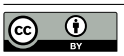
Alongside this influx of attention to memory, primarily from industry, there has been little attention paid to the corresponding *interface design* decisions around AI memory. AI memory is not merely a technical problem but an *affordance* for user control, transparency, and explainability over agent decision-making. Different choices of technical architectures for memory present different affordances and constraints for designing interfaces for users to read and modify that memory—for instance, storing memories by fine-tuning the model makes later inspection near-impossible, whereas storing memories as a flat list of preferences is more auditable by users, but may lead to performance drops as details are lost. Accordingly, AI memory design necessarily involves a rich design space of architectural, operational, and interaction design choices, all of which can affect one another.

From a human-AI interaction standpoint, however, we still lack a systematic account of these design choices. While the functional and technical aspects of AI memory have been studied [49, 93], an interaction-oriented breakdown of the design of memory remains comparatively underexplored. What design space exists across memory architectures, operations, and interaction mechanisms, and where do current AI memory systems fall short of what future interactive AI memory could provide? What are the downstream implications of memory structure, including how structural and operational constraints shape later decisions and affect interaction quality, agent capability, and temporal or computational costs?

To address these questions, we conducted a review of current AI memory tools and interfaces to arrive at a design space [14, 54] for AI memory, including how AI's memory is represented and structured, how these choices on AI memory design shape interaction, and what trade-offs they introduce. Using this design space as an analytic lens, we then show how existing systems instantiate these design choices, identify dominant patterns and metaphors, and discuss gaps that remain. Finally, we draw on generative metaphorical design [50, 74] to propose new, lesser-used metaphors for AI memory to spark ideas for addressing these gaps, and discuss how such metaphors could inform future AI memory systems.

2 Background

Here we provide a brief introduction to the history of AI memory, an overview of the current space of AI memory, and conceptual frameworks from communication theory that are increasingly employed to describe the problem of human-AI grounding [75, 83].



DIS '26, Singapore, Singapore

ACM ISBN 979-8-4007-2563-0/26/06
https://doi.org/10.1145/3800645.3812979

2.1 A brief history of AI "memory"

What "memory" means in AI systems and agent design has shifted over time. In early NLP, a model's "memory" was mostly understood as internal recurrent state or weight-encoded representations (e.g. LSTM [37])—that is, something closer to the term implicit memory [49, 72] in its current meaning, rather than a separate memory architecture. Later, with higher-capability LLMs such as GPT-4 [1] and techniques such as Chain-of-Thought (CoT) [87], retrieval augmented generation (RAG) [48], and tool-use frameworks like CoALA [79], alongside concepts like the context window, system prompts, and prompt engineering [12, 17, 33, 63], "memory" expanded to include explicit, external data storage and a clearer distinction between implicit and explicit, yet these "memories" were often spoken of as run-time auxiliary inputs rather than as the agent's enduring memory (e.g., injecting a list of user preferences as the "memory" to a system prompt, upon every call). More recently, LLM-based coding agents [5, 25, 31, 32, 61], enabled by larger context windows and stronger reasoning models (e.g., GPT-5 [62], Claude Sonnet/Opus 4.5 [6, 7], and Gemini 2.5 Pro [29]), increasingly control how external memory is written, updated, and reorganized, shifting memory management from human curation to agent-driven workspace maintenance, with users primarily supervising [67]. However, how to build *long-term* external memory at roughly human (or superhuman) levels remains relatively underexplored [49].

Meanwhile, there have also been attempts to systematically survey the technical components of AI agent memory in the emerging landscape. CoALA [79] draws on cognitive science and symbolic AI to describe modular memory components for language agents, classifying memory into short-term versus long-term and further dividing long-term memory into procedural [22, 34], semantic [13], and episodic [80] types. Liu et al. [49] systematically analyze AI agents by decomposing their memory-related components and mapping them to human cognitive and perceptual functions. Zhang et al. [93] survey memory mechanisms in LLM-based agents, covering both the structural design of memory modules and their evaluation. Rolls [69] compares and reviews the human brain and generative artificial intelligence from a neuroscience perspective, highlighting differences in the structure and operations of their respective memory systems. While these works clarify how AI memory systems are designed at an architectural level, AI memory from an interaction and design perspective—such as how architectural choices systematically shape an interactive system's affordances and thus the overall interaction patterns users encounter, and what sets of architectural and operational design choices are required to enable intended user interactions and meet desired design goals—remains comparatively less studied, if at all remarked upon.

2.2 Interfaces for AI memory in HCI

Several HCI studies offer useful inspiration for designing the interaction and operations of AI memory interfaces. Memolet [91] reifies "memories" from prior conversations into reusable objects, and lets users explicitly specify and manipulate a memory space where both the user and the agent can work with shared, groundable memory units. Memory Sandbox [38] visualizes an agent's conversational memories as data objects, so users can treat "what the agent remembers now and how it sees things" as a shared context they can adjust. Semantic Commit [83] supports updating a natural-language specification of user intent in long-running projects by helping detect and resolve non-local impacts and semantic conflicts, enabling users and agents to jointly track intent changes and manage a shared information space. Recently, Knoll frames memories as knowledge bases that are transferrable across agents [94], similar to Anthropic's later plug-n-play notion of "Skills" [4].

Even though HCI work on human–AI collaboration does not explicitly frame its contribution as "AI memory," many such systems also grapple with related interaction challenges. Systems like CollaborativeGym [76] explore how to visualize and manage an information space that humans and agents can jointly ground in, and how, within that space, a user and an agent might jointly navigate, connect ideas, and iteratively build a shared body of knowledge. GhostWriter [90] likewise externalizes style and context as editable documents that users can inspect and iteratively refine, helping ground generated text in user-provided context and better align it with the user's voice over time. Sensemaking tools such as Sensecape [78], systems designed to support human memory such as Graphologue [39], and writing assistants such as TaleBrush [19] can all serve as relevant reference points for designing AI memory systems' interfaces and supported operations.

2.3 Conceptual Perspectives on Communication, Context, and Memory

In addition to mapping the design space of memory systems, researchers draw on perspectives from communication that can help in building interactive agent memory systems. Vaithilingam et al. [82] envision how a future AI game-design assistant might collaborate in design work and gradually build dynamic ground with a user over time, while Ma et al. [53] propose a Human-AI deliberation approach, derived from deliberation theory, in which humans and AI elicit each other's opinions through ongoing discussion and continually update decisions toward agreement.

Recent work in human-AI communication [11, 75, 82, 83] has applied the framing of "common ground" and "grounding" from Clark & Brennan [20, 21] towards the creation of agent systems that maintain analogous common ground between humans and AI. Common ground refers to the mutual understanding (intents, context, details) accumulated over time between communicating parties, such as when working together on a project [20]. Similar to a human actor, an AI agent must infer the user's intent and its context solely from the user's expression, and then make decisions and act accordingly; if the agent infers this incorrectly, it will fail to carry out what the user expects in the appropriate context. Therefore, it is important during interaction to verify that the agent's interpretation is aligned with the user's intent and context, and to adjust it when necessary. Moreover, to avoid the repeated cost of grounding, maintaining common ground formed across multiple sessions in a persistent and coherently connected form requires recording relevant knowledge in an additional memory store and continuously updating it, which is important from the perspective of long-term interaction. This knowledge store, like human

grounding, must remain consistent and aligned to the other parties' current understandings.

Over long-horizon interaction, such iterative cycles can help the agent sustain a more stable context, enabling the agent to "learn" about the world through interaction with it. However, this process cannot operate reliably without an appropriately designed agent memory system. It may fail, for example, when the design cannot support the required level of relational network structure, or when it lacks the manipulable operations needed to sustain the feedback loops that arise in interaction. Therefore, designing interactive AI memory systems requires considering the structural aspects of memory, the operations it affords, and how these will be used in interaction, so that memories can be recorded, revised, connected, and validated during interaction.

Building on these concepts, industrial products and academic systems increasingly aim to support agent's building an understanding of user intent over time. Chat-based agents like ChatGPT [60] support custom memory that allows users, or the agent itself autonomously, to add context and persist information across sessions. Code agents like Claude Code [5] similarly support persistent instructions via CLAUDE.md and allow users to extend behavior through Agent Skills [4] and subagents, which can be invoked in their own scoped memory context. On the research side, PAIL [92] explicitly surfaces users' requirements in software development and support the process of refining and updating them over time, while Semantic Commit [83] shows how users and agents can collaboratively address emerging conflicts around user intent and revise their grounding as those intents are established and updated. Finally, GUM [75] envisions that an AI agent, watching the user as they work, will accumulate knowledge about the user over time that systems can use to ground and tailor later AI outputs. In what follows, we will outline this landscape more comprehensively.

3 Mapping the Current Design Space of AI memory

Here we examine the design space of AI memory—how different AI memory systems, shaped by their goals, intended interaction patterns, and practical constraints (e.g., LLM capability and cost, latency, and deployment setting), make structural choices and thereby afford specific operations to both agents and users. Under recurring combinations of goals and technical constraints, systems often converge on dominant patterns and the trade-offs they implicitly accept, alongside atypical but meaningful exceptions. Our goal is to help designers of interactive AI agent systems make purpose-fit choices under their goals and constraints, and to help researchers identify gaps that motivate new interaction designs.

We conducted two surveys on AI memory systems' design choices, one focused on architectural design choices (Section 3.1; Table 1) and the other on operational design choices (Section 3.2; Table 2), and then derived the design space from these results (Figure 1).

To establish the axes of the design space and the options within each axis, we began with a set of seed axes, derived from a provisional framing of architectural choices, interaction patterns, and provided operations, and refined them during coding as needed. In the architectural survey, coding was conducted at the level of the memory system as a whole, whereas in the operational survey, distinct subsystems were sometimes treated separately when they exposed different memory roles, interfaces, or affordances. The first author led the coding process, while the coauthors reviewed emerging categories, discussed ambiguous cases, and provided feedback on changes to the scheme. Categories changed whenever recurring cases no longer fit the existing scheme cleanly or became too internally complex to be adequately described as a single option. Whenever an axis or its options changed, we revisited and updated the affected codes to keep the corpus consistent. We iterated this process until the axes and option sets stabilized, that is, until additional cases could be placed within the existing axes and option sets without requiring further changes to them.

Our corpus was assembled through snowballing from a small set of seed systems, well-known in the space (e.g., LangGraph [44], Claude Code [5], Mem0 [18]), rather than defined as an exhaustive survey. We intentionally focused on tools and APIs, rather than academic texts, which often lag behind industry and, in HCI, do not correspond to widely-used systems; among HCI systems, we preferred those whose designs are sufficiently well-documented, for example publications that describe the underlying structure in detail or accessible code-bases. For each case, we relied primarily on papers, documentation, and publicly accessible repositories, generally prioritizing directly inspectable implementations or APIs when available. When later product evolution had introduced features not present at the time of publication, however, we prioritized the paper's description and the implementation state corresponding to that publication period.

In the architectural survey, we included systems in which memory was a substantive design target and excluded cases whose architectures were not publicly available enough to analyze reliably, such as ChatGPT [60]. In the operational survey, we included additional cases beyond those in the architectural analysis whose operation affordances are observable through explicit specifications or publicly accessible platforms. For systems that combine multiple layers yet assign them clearly distinct roles and layer-specific interactions (e.g., LangGraph [44]/LangMem [45], or Claude Code [5]'s workspace, agent.md, and chat-memory layers), we analyzed the operations of each layer separately.[1] In our operation analysis, we excluded cases that do not have independent operations of their own, such as Claude's Agent Skills [4] and subagents [8].

In total, we conducted an architectural analysis of 14 distinct systems and an operational analysis of 15 distinct systems (22 subsystems in total). Based on these results, we constructed the design-space figure (Figure 1) and the corresponding tables that summarize each system's observed architectural (Table 1) and operational (Table 2) design choices.

[1] In contrast, we did not separate Mem0 and Mem0g [18] because, in the current implementation, Mem0g's functionality is not exposed via independent endpoints but is toggled within Mem0's existing API surface.

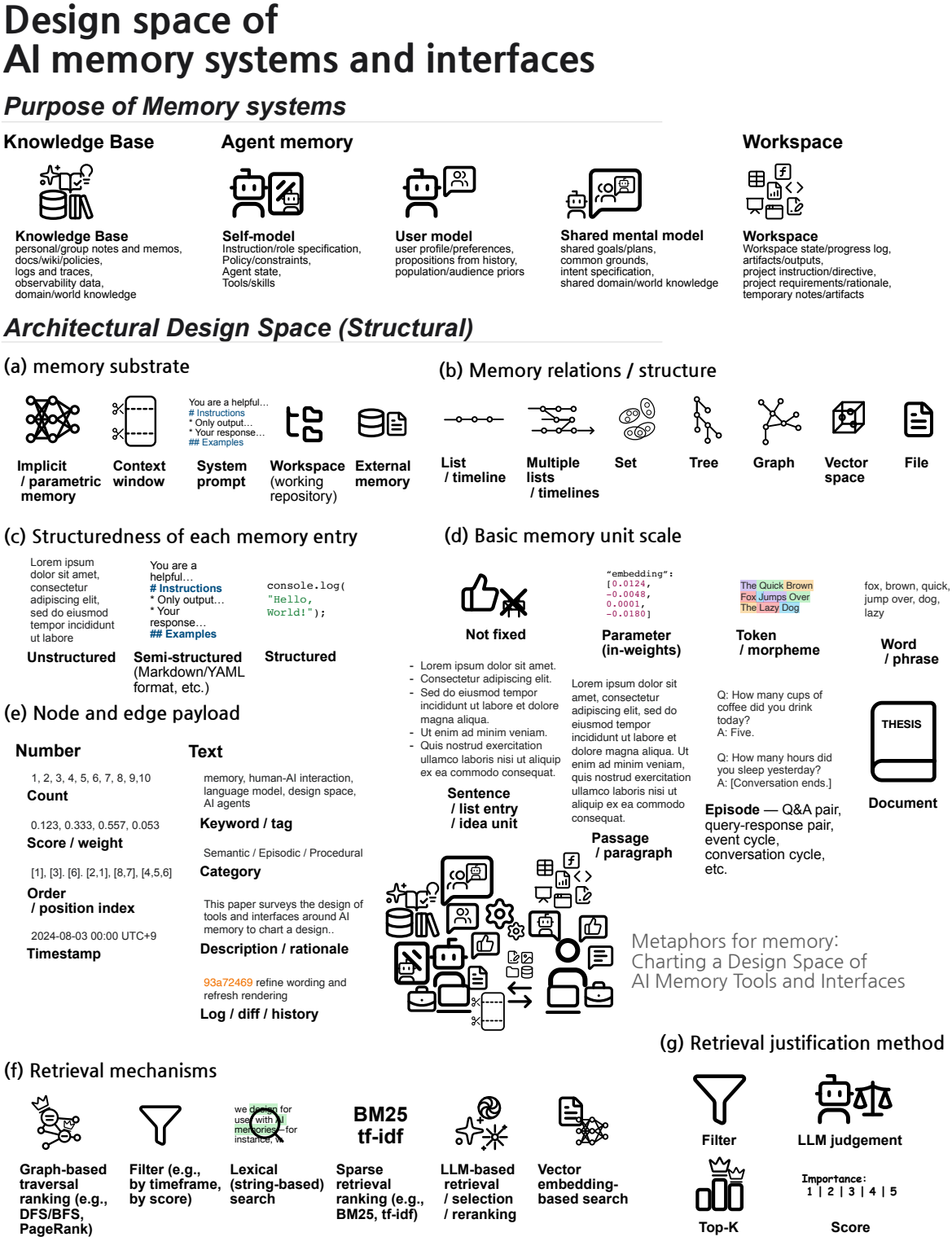


Figure 1: Design space of AI memory systems and interfaces

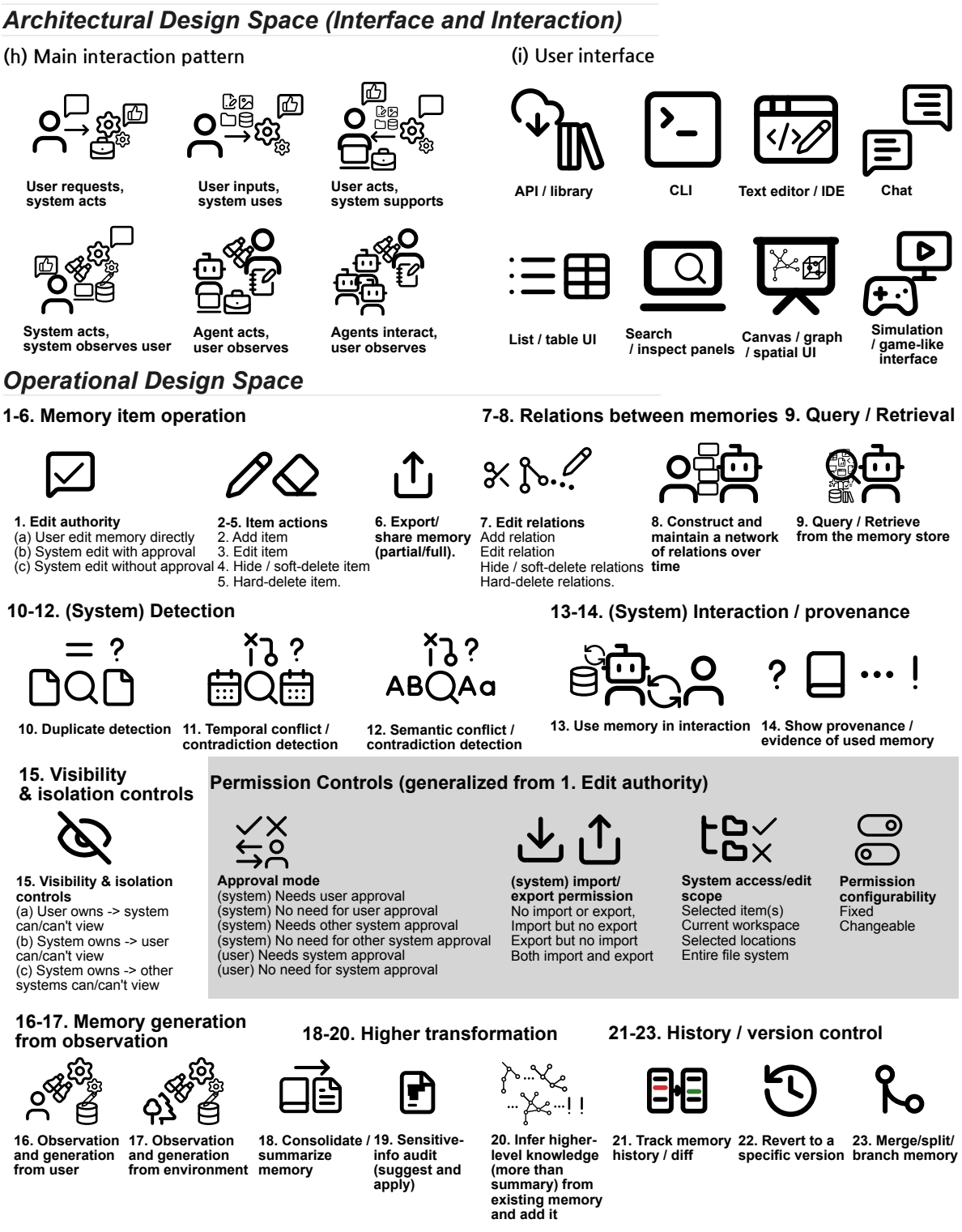


Figure 1: Design space of AI memory systems and interfaces (continued)

Figure 1 provides an overview of the resulting AI memory design space by synthesizing the dimensions identified in our architectural (Table 1) and operational (Table 2) surveys. The figure is organized into three parts: Purpose of Memory Systems, Architectural Design Space, and Operational Design Space. We present it upfront as an index and map to the dimensions that are later unpacked in Section 3.1 and Section 3.2. In the architectural part, the labels (a)–(i) correspond to the dimensions later instantiated in Table 1. In the operational part, the numbers and number ranges correspond to the operational rows and grouped ranges in Table 2. We place Purpose of Memory Systems first because it is important for understanding the overall AI memory design space, but do not present it as a separate system-level survey because these purposes did not form clear system-level distinctions and were typically pursued simultaneously within the same systems. Gray cells indicate theoretically relevant options that were retained in the design space even when they were not directly instantiated in the surveyed corpus.

3.1 Architecture

Table 1 summarizes the architectural design choices of the memory systems in our corpus. Rows correspond to systems, and columns correspond to key design dimensions. Each cell lists the option(s) adopted by a system; multiple items within a cell are separated by line breaks. When multiple items are listed, all of them are present in the system (often serving different roles or appearing in different parts of the system); however, earlier items typically reflect the system's most central or emphasized design choices, while later items tend to play supporting roles. For example, in HippoRAG [40]'s "Basic memory unit scale," we list the retrieval-level unit (word) before the storage-level unit (document), since the system's central design emphasis lies in the retrieval layer. A dash (–) indicates that the system does not include any corresponding structure or mechanism for that dimension.

3.1.1 Memory substrate (Table 1, column (a)). All of the analyzed systems include external memory as a main memory substrate. In addition, systems may provide a workspace, context window, and/or a system prompt, depending on what the system aims to demonstrate. However, even external-memory-only systems often propose end-to-end workflows that help users fully leverage the memory substrate design space with other systems.

3.1.2 Memory relations / structure (Table 1, column (b)). Systems whose core contributions are in memory structure generally adopt graph-based structures [18, 40, 66, 88]. In contrast, systems whose core contributions are not in structure but in interaction typically adopt simpler forms, such as a temporal list [38, 64, 65, 75] or embedding-vector-based grouping [91]. Semantic Commit [83] is an exceptional case, although its main contribution is in interaction, it identifies the limitations of list- or embedding-based approaches as a problem and introduces an additional graph structure.

3.1.3 Structuredness of each memory entry (Table 1, column (c)). While early systems (e.g. MemGPT [64], Memory Sandbox [38]) stored memories as unstructured strings, most systems adopted schema-based entries (e.g., JSON) to attach metadata. Claude's agent skills [4] and subagents [8], meanwhile, use Markdown with a YAML front matter header, reflecting an assumption that memory operations are mostly mediated by the LLM and thus prioritizing malleability over strict schema enforcement.

3.1.4 Basic memory unit scale (if (semi-)structured) (Table 1, column (d)). The scale of the memory unit varies by purpose. For storing full logs, most systems kept each memory item as a whole (at the document level). For interaction or editing, they tended to use smaller granularities (e.g. passages, episodes, or ideas) sometimes combining multiple units. Most additional retrieval layers operate at the word level.

3.1.5 Node and edge payload (Table 1, column (e)). Node and edge payload depend heavily on the memory structure a system chooses. Compared to graph-based systems, list-based and hierarchy-based systems have limited relational structure, which constrains payloads to minimal signals such as scores or timestamps. However, in our analysis, graphs in those graph-based systems largely took the form of triple-based knowledge graphs, so edge payloads were limited to keywords. In vector-embedding stores (e.g., Memolet [91]), the embedding space itself functions as the predetermined relational structure. Therefore, there are constraints on arbitrarily adding additional edge payloads, making this design option limited in practice.

3.1.6 Retrieval mechanisms (Table 1, column (f)). Retrieval mechanisms also depend heavily on the preceding design choices. Systems that include keywords, timeframes, or scores often provide filtering or search functions based on those fields. Graph-based systems similarly tend to support graph traversal (e.g., DFS/BFS). A notable case is HippoRAG [40], which performs Personalized PageRank [36] over a knowledge graph for retrieval. Most tools also adopt content-level retrieval over each memory unit, including lexical search, vector embedding search, BM25 [68], and/or LLM query-based retrieval.

3.1.7 Retrieval justification method (Table 1, column (g)). While retrieval mechanisms and retrieval justification methods are not strictly paired, they tended to co-vary in our corpus. Systems using deterministic retrieval usually also relied on deterministic justification criteria, such as filters, scores, and top-k cutoffs [2, 18, 40, 44, 64, 66, 88, 91]. By contrast, systems that involved LLMs in retrieval typically used LLM judgment as an additional basis for justification, for example by assigning importance scores [65], reranking retrieved candidates [75], or applying a final filtering step after an initial deterministic retrieval [83].

3.1.8 Main interaction pattern (Table 1, column (h)). Dominant interaction patterns fell into four types: (1) users request actions and the system performs memory operations [18, 40, 66, 88]; (2) users act and the system supports memory work [38, 83, 91]; (3) agents autonomously observe and interact with the environment while accumulating memory, and users mostly observe [2, 65]; or (4) users input memory into the store for agents' later use [4, 8, 44]. A distinctive case is the General User Model [75], where memory operations are driven primarily through observing the user and the user is positioned as the observed subject rather than the observer of an acting agent, unlike the other systems where memory operations are typically initiated by users through some instruction, data, or initial environmental conditions and users are more often positioned to observe and evaluate the agent.

3.1.9 User interface (Table 1, column (i)). The interface was most often provided in API and library. When GUIs were provided, memories were typically displayed as sequential (table) lists or chat logs, often with extra panels (e.g., inspector panel). Two distinctive cases are Memolet [91]'s 2D card-grid canvas and Generative Agent's simulation-game-like UI.[2]

3.2 Operations

Table 2 enumerates the memory operations supported by each system. The columns list the memory systems, grouped by layer, while the rows list operations together with the actor responsible for each operation. Each cell is coded as Yes (✔) when the system provides the capability and No (-) otherwise. For edge cases, we use a circle (○) when the operation is not supported as a specified feature but can still be achieved indirectly (e.g., via LLM calls or workaround workflows), and we use a triangle (△) to mark partially supported cases where the operation is implied by the system's definition or data model but is incomplete in practice (e.g., General User Model [75] is append-only with timestamps, so memory entries themselves form a rudimentary history, but it would be a stretch to say it supports "history tracking" as an explicit operation).

3.2.1 Item Operation (Table 2, rows 1-6). Memory-manipulation authority tended to follow the dominant interaction pattern of the system. User-led systems gave users more control over memory operations while agent authority was often more limited. Agent-led systems tended to have reversed pattern. User-approved edits were rare. When agents could edit memory autonomously, edits usually occurred without user consent. Soft delete/hide was absent in most systems except Memolet [91] and Memory Sandbox [38]. Beyond ChatGPT [60]'s chat sharing, we did not observe any in-system export/sharing features for memory. In most cases, users had to download memories or implement sharing themselves. We found no systems that allow agents to autonomously export or share memory.

3.2.2 Relations between Memories (Table 2, rows 7-8). Where explicit relationships and edges do exist, systems often allowed some relationship editing; however, in most cases this meant either users manually editing edges one by one or agents automatically updating edges during their update process. We did not observe any systems that support user-approved relationship edits. Meanwhile, it was notable that most graph-based memory systems aim to construct and maintain a network of relations over time.

3.2.3 Query and Retrieval (Table 2, row 9). Most systems provided query authority to both users and agents. Even when query functionality was absent for the user and/or the agent, in most cases it was because queries were not necessary under the system requirements rather than due to structural limitations.

3.2.4 Detection (Table 2, rows 10-12). Some systems provided additional detection operations beyond basic querying. Zep [66], Mem0 [18], Arigraph [2], and the General User Model [75] support write-time deduplication and handle duplicates by skipping, replacing, or grouping entries. In parallel, Zep [66] supports temporal-conflict detection to catch temporal inconsistencies across memory contents. Separately, Zep [66], Mem0 [18], Arigraph [2], and Semantic Commit [83] provide LLM-based semantic contradiction detection.

3.2.5 Interaction and Provenance (Table 2, rows 13-14). While most AI memory tools were ultimately designed to operate as part of a user–agent or agent–agent interaction pipeline, whether interaction-related operations were implemented in-system depended on the scope the system aimed to cover. Among them, Semantic Commit [83], Memolet [91], and the chat interfaces of Chatgpt [60] and Claude [5] also provided provenance about which memories the agent referenced during interaction.Some systems do not explicitly show evidence or provenance for interactions, but it is still possible to see which memories were retrieved at that point through logs or return values. We treated it as partially supported in such cases.

3.2.6 Visibility and Isolation (Table 2, row 15). There were few cases where in-system operations allowed controlled differential visibility or memory isolation between users and agents, aside from Memory Sandbox [38]'s selective use of memories in chat and ChatGPT [60]/ Claude Code [5] instruction prompts that are visible only to the agent and selectively disclosed to the user.

3.2.7 Observe (Table 2, rows 16-17). Systems often allowed agents to observe and record user behavior to build long-term memory without manual user curation. In simulation settings, observing and recording external environments and interactions was essential for agents to perceive, judge, and act autonomously.

3.2.8 Higher-level Memory Transformation (Table 2, rows 18–20). Higher-level memory transformations beyond atomic edits became more common as agents gained greater memory-control authority. Consolidation mainly served to streamline memory stores, either compressing memory store (Zep [66], ChatGPT [60], Claude Code [5]) or producing more readable memory sets for users (Memolet [91], Memory Sandbox [38]). Auditing appeared where sensitive disclosure is likely (General User Model [75], ChatGPT [60]), but in both systems the rule set was predetermined and not user-configurable. Inference supported autonomous action (Generative Agents [65], Arigraph [2]) or proposition extraction from observed user interactions (General User Model [75], SpecStory [16], ChatGPT [60]).

3.2.9 History Tracking and Version Control (Table 2, rows 21–23). In our analysis, even when systems supported memory deletion, support for version control or history was rare. Append-only memory systems were coded as partial support, since the current state functions as de facto versioning even without explicit history/version-control features. For this reason, most tools did not support higher-level version-control operations such as reverting, merging, splitting, or branching memory. Claude Code [5] can perform these operations by executing Git commands, but they are not supported via in-system operations; we coded this as supported only via a workaround.

[2]Note, however, that in Generative Agents the primary purpose of the UI was to display the simulation than the memory itself.

Table 1: Architectural design choices across AI memory systems

System	(a) Memory substrate	(b) Memory relations / structure	(c) Structuredness of each memory entry	(d) Basic memory unit scale	(e) Node and edge payload	(f) Retrieval mechanisms	(g) Retrieval justification method	(h) Main interaction pattern	(i) User interface
MemGPT[64]	Context window System prompt External memory	Timeline	Unstructured	Document Passage	Timestamp	Timeframe filter Lexical search Vector embedding	Filters Top-K	User requests, system acts	API/Library, Inspector panel+chat+list
LangGraph (BaseStore) [44]	External memory	Tree	Structured	Not fixed	Category Timestamp	Lexical search	Top-K Filters	User inputs, system uses	API/Library
HippoRAG [40]	External memory	Graph	Structured	Word Document	Count Keyword	Personalized PageRank	Top-K	User requests, system acts	API/Library
Zep/Graphiti [66]	External memory	Multi timeline Graph	Structured	Word Sentence Episode	Keyword Timestamp	Vector embedding BM25 Graph (BFS/DFS) Timeframe filter	Top-K Filters	User requests, system acts	API/Library
A-MEM [88]	External memory	Graph	Structured	Sentence	Keyword	Vector embedding Graph (BFS/DFS)	Top-K	User requests, system acts	API/Library
Mem0/ Mem0g [18]	External memory	Timeline Graph	Structured	Document Episode Sentence Word	Keyword	Vector embedding BM25 Graph (BFS/DFS)	Top-K	User requests, system acts	Table
AriGraph [2]	Context window External memory	Timeline Graph	Structured	Document Episode Sentence Word	Keyword	Vector embedding Graph (BFS/DFS)	Top-K	Agent acts, user observes	Text game interface
Generative Agents [65]	Context window System prompt External memory	Multi timeline	Structured	Word Sentence	Score Weights Timestamp	Lexical search Score Filter	Filters LLM judgement Scores Top-K	Agents interact, user observes	Sims-like observational interface
General User Model [75]	Workspace External memory	List Timeline	Structured	Sentence Episode	Timestamp Score Weights	BM25 LLM	Top-K LLM judgement	System acts, system observes user	API/Library
Semantic Commit [83]	Workspace External memory	List Graph	Structured	Word Sentence	Keyword	LLM Personalized PageRank	LLM judgement	User acts, system supports	Center list with search/inspect panels
Memolet [91]	Context window Workspace External memory	Vector space Set	Structured	Document Episode	-	Vector embedding	Top-K	User acts, system supports	Chat and card-grid canvas
Memory Sandbox [38]	Context window System prompt	List	Unstructured	Document Sentence	-	-	-	User acts, system supports	Chat
Claude Agent Skills [4]	External memory Context window	Tree	Semi-structured	Document	-	-	-	User inputs, system uses	-
Claude Code Subagents [8]	External memory System prompt Context window	Single file	Semi-structured	Document	-	-	-	User inputs, system uses	CLI and IDE

Note. Each entry lists the adopted options for a system; within a cell, earlier items indicate relatively higher importance.

3.3 Implications of architectural decisions of AI memory

Across these analyses, we derived several implications about how specific design choices in AI memory systems shape interaction-level affordances and trade-offs, while also identifying several gaps in current dominant design patterns. Sections 3.3 and 3.4 briefly discuss these implications and gaps.

3.3.1 Implicit versus explicit memory. Designers' decisions about whether information should live in implicit or an explicit memory can be clarified by analogy to human implicit versus explicit memory. Implicit memory corresponds to knowledge we want to surface reflexively and immediately through training, whereas explicit memory corresponds to a knowledge network that is accumulated over time and kept in a form that can be inspected and edited. The two approaches involve a familiar set of trade-offs. Implicit memory is fast, but difficult to update, and it is often hard to trace the reasoning or provenance behind a response. Explicit memory can be slower, but it is comparatively easier to edit, delete, or scope, and it better supports tracing which stored information contributed to a given output. Explicit memory also makes it easier to construct context-dependent responses by retrieving different combinations of memories depending on the situation and inferred intent.

In practice, the choice often comes down to whether memory is meant to enable reflex-like behavior (implicit) or to preserve inspectable context and rationale (explicit). For example, when memory updates are rare (e.g., language rules, basic mathematics, general programming knowledge), personalization is minimal (e.g., systems targeting large, heterogeneous user groups), speed is critical, and provenance is not strongly required, encoding the knowledge into implicit memory can be beneficial. Conversely, when memory updates are frequent, personalization matters, provenance during reasoning is important, or the same question requires

Table 2: Operational design choices across AI memory systems

		LangGraph [44]			Zep [66]											ChatGPT [60]				Claude Code [5]		
Operation	Mem GPT [64]	Base Store	Lang Mem [45]	Hippo RAG [40]	Zep (cloud)	Graph iti	A-MEM [88]	Mem0 [18]	AriGraph [2]	Genera -tive agents [65]	General User Model [75]	Semantic Commit [83]	Memo let [91]	Memory Sandbox [38]	Spec Story [16]	Instruc tion	Memory	Project files	Chats	Work space	CLAUDE .md	Chats
1-6. Memory item operation																						
1. Edit authority																						
(a) user edit memory directly	✔	✔	✔	✔	✔	✔	✔	✔	-	✔	-	✔	✔	✔	-	✔	✔	✔	✔	✔	✔	✔
(b) system can edit under user approval	-	-	-	-	-	-	-	-	-	-	-	✔	-	-	-	-	-	-	-	✔	✔	-
(c) system can edit without user approval	✔	-	✔	✔	✔	✔	-	✔	✔	✔	✔	-	✔	✔	-	-	✔	-	-	✔	✔	-
2. Add item																						
(a) user	✔	✔	✔	✔	✔	✔	✔	✔	-	✔	-	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
(b) system with approval	-	-	-	-	-	-	-	-	-	-	-	✔	✔	-	-	-	-	-	✔	✔	✔	-
(c) system without approval	✔	-	✔	-	-	✔	-	✔	✔	✔	✔	-	✔	△	✔	-	✔	-	-	✔	✔	✔
3. Edit item																						
(a) user	△	✔	✔	-	-	✔	✔	✔	-	-	-	✔	-	✔	✔	-	-	-	-	-	-	-
(b) system with approval	-	-	-	-	-	-	-	-	-	-	-	✔	-	-	-	✔	-	-	✔	✔	✔	-
(c) system without approval	△	-	✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	-	✔	✔	✔	-
4. Hide / soft-delete item					*																	
(a) user	-	-	-	-	-	-	-	-	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-
(b) system with approval	-	-	-	-	-	-	-	-	-	-	-	-	-	△	-	-	-	-	-	-	-	-
(c) system without approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5. Hard-delete item																						
(a) user	✔	✔	✔	✔	✔	✔	✔	✔	-	-	-	✔	-	✔	✔	✔	✔	✔	✔	✔	✔	-
(b) system with approval	-	-	-	-	-	-	-	-	-	-	-	✔	-	-	-	-	-	-	-	✔	✔	-
(c) system without approval	△	-	✔	-	-	✔	-	✔	✔	-	-	-	-	-	-	-	✔	-	-	✔	✔	-
6. Export/share memory (partial/full)																						
(a) user	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✔	-	-	-
(b) system with approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(c) system without approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7-8. Relations between memories																						
7. Edit relations (add/delete/update; single or multiple relations)																						
(a) user	-	△	-	-	✔	✔	✔	-	-	-	-	-	✔	-	-	-	-	-	-	-	-	-
(b) system with approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(c) system without approval	-	-	-	-	-	-	✔	✔	✔	-	-	✔	✔	-	-	-	-	-	-	-	-	-
8. (system) Construct and maintain a network of relations over time	-	-	-	✔	✔	✔	✔	✔	✔	-	-	✔	✔	-	-	-	-	-	-	-	-	-
9. Query / Retrieval																						
9. Query / retrieve from the memory store																						
(a) user	✔	✔	✔	✔	✔	✔	✔	✔	-	✔	✔	-	✔	△	✔	✔	✔	✔	✔	✔	✔	✔
(b) system with approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✔	✔	-
(c) system without approval	✔	-	✔	✔	-	✔	-	✔	✔	✔	✔	-	✔	△	-	✔	✔	✔	✔	✔	✔	-
10-12. Detection																						
10. (system) Duplicate detection	-	-	✔	△	✔	✔	-	✔	✔	-	✔	-	-	-	-	-	-	-	-	○	○	-
11. (system) Temporal conflict/contradiction detection	-	-	-	-	✔	✔	-	-	△	-	-	-	-	-	-	-	-	-	-	○	○	-
12. (system) Semantic conflict/contradiction detection	-	-	✔	-	✔	✔	-	✔	✔	-	-	✔	-	-	-	-	-	-	-	○	○	-
13-14. Interaction & provenance																						
13. (system) Use memory in interaction	✔	-	✔	✔	-	-	-	✔	✔	✔	-	✔	✔	✔	-	-	-	-	✔	-	-	✔
14. (system) Show provenance/evidence of used memory	△	-	-	✔	-	-	-	-	△	-	-	✔	✔	-	-	-	-	-	✔	-	-	✔
15. Visibility & isolation																						
15. Visibility & isolation controls																						
(a) User owns → system can/can't view	-	-	-	-	-	-	-	-	-	-	-	-	-	✔	-	-	-	-	-	-	-	-
(b) system owns → user can/can't view	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✔	-	-	-	-	-	✔
(c) system owns → other systems can/can't view	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Legend: ✔ = Yes; - = No; ○ = Workaround (not a built-in feature; achievable via LLM calls using the system's supported functions); △ = Partially yes (yes in principle, but limited implementation)
* Legacy Zep soft-delete was removed in the current release.

Table 2: Operational design choices across AI memory systems (continued)

		LangGraph [44]			Zep [66]											ChatGPT [60]				Claude Code [5]		
Operation	Mem GPT [64]	Base Store	Lang Mem [45]	Hippo RAG [40]	Zep (cloud)	Graph iti	A-MEM [88]	Mem0 [18]	AriGraph [2]	Genera -tive agents [65]	General User Model [75]	Semantic Commit [83]	Memo let [91]	Memory Sandbox [38]	Spec Story [16]	Instruc tion	Memory	Project files	Chats	Work space	CLAUDE .md	Chats
16-17. (system) Memory generation from observation																						
16. Observation and generation from user	✔	-	✔	-	-	-	-	-	-	-	✔	-	-	-	✔	-	✔	-	-	-	-	-
17. Observation and generation from environment	-	-	-	-	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	-	-	-	-
18-20. Higher transformation																						
18. Consolidate / summarize memory																						
(a) system with user approval	-	-	-	-	-	-	-	-	-	-	-	-	✔	✔	-	-	-	-	-	○	○	○
(b) system without user approval	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	-	-	✔	-	-	-	-	✔
19. Sensitive-info audit (suggest and apply)																						
(a) system with user approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	○	○	○
(b) system without user approval	-	-	-	-	-	-	-	-	-	-	✔	-	-	-	-	-	✔	-	✔	-	-	-
20. Infer higher-level knowledge (more than summary) from existing memory and add it																						
(a) system with user approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✔	○	○	○
(b) system without user approval	-	-	✔	-	-	-	-	-	△	✔	✔	-	-	-	✔	-	-	-	✔	-	-	-
21-23. History / version control																						
21. (system) Track memory history / diff	-	-	-	-	-	-	-	✔	✔	△	△	-	-	-	-	-	✔	-	-	○	○	✔
22. Revert memory to a specific version																						
(a) user	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✔	-	-	○	○	△
(b) system with approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	○	○	-
(c) system without approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	○	○	-
23. Merge/split/branch memory																						
(a) user	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	○	○	-
(b) system with approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	○	○	-
(c) system without approval	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	○	○	-

Legend: ✔ = Yes; - = No; ○ = Workaround (not a built-in feature; achievable via LLM calls using the system's supported functions); △ = Partially yes (yes in principle, but limited implementation)

different answers depending on context and intent, it can be advantageous to store the relevant knowledge in an explicit layer as an editable network, as long as the latency cost remains acceptable.

3.3.2 How should we scope and target memory substrates? As Figure 1 and Table 1 suggest, even when a system uses explicit memory, there are multiple places where memory can be situated and manipulated. The same information can live in the system prompt, in an external store, or in a shared workspace where humans and an AI co-produce artifacts. Alternatively, a system may rely on additional agents with independent context windows, as in Claude Code subagents [8]. In practice, an agent ends up using all of these substrates. Even if a particular system covers only a subset of them, designers should still consider how it connects to the rest.

For example, suppose a memory system is designed as a work-oriented database that both the agent and users can reference. In real use, the agent will not rely on the database alone, but will act within its context window under a system prompt that specifies roles, rules, and instructions, and it will retrieve from the database as needed to compose responses. At the same time, users typically want more than a Q&A interface: they want the agent to understand the workspace they are operating in as if it were "looking at it together" with them, and to retain task context so it can bring in the right information at the right time. In other words, even if one build just a database, users will still expect the system to connect that store with documents, chats, workspace artifacts, and traces used in other context windows in a coherent way. Similarly, even if one develops only one part of the interaction pipeline, it is not enough to focus solely on the system-prompt instructions, and users will want the system to integrate with external stores and workspaces.

Ignoring these connections may not break the system immediately, and the problem may not be visible at first, but it almost inevitably creates friction over time. For instance, if two incompatible systems exist, one strong at storage and another strong at dialogue, reasoning, or tool instructions, users end up switching back and forth and manually copy-pasting, or they eventually give

up on one of them. Therefore, even when designing a single system, it is worth keeping the entire agent pipeline in mind, including how users will actually interact with it and how naturally it interoperates with other tools, stores, and workspaces.

3.3.3 Granularity and density of memory components. When designing structures to hold and manage knowledge, there is an inherent trade-off between fidelity and usability: larger memory units preserve more context but are less likely to be read, while smaller units are easier to scan but make faithful reconstruction harder. The same trade-off applies to relations: simple edges (e.g., scalar weights) are easy to process but carry little explanatory content, whereas richer edges (e.g., predicates, keywords, short rationales) preserve "why" at higher processing cost. The trade-off is similar in terms of relational density: sparse structures (e.g., lists and independent notes) are easy to navigate but cannot explicitly encode complex, long-range relations. In contrast, denser structures (e.g., trees and graphs) can capture such relations but make navigation more difficult.

These granularity- and density-related processing trade-offs also apply to users, but in a somewhat different way. In general, people are accustomed to reading text at the sentence level, consuming information one unit at a time and largely in a single direction, and they also tend to avoid overly long texts. Therefore, when designing memory systems that humans are expected to consult directly alongside agents, one must consider not only what kinds of data an agent can process well, but also how humans actually read. For example, ontology-style knowledge graphs expressed as noun–verb triples often do not make it clear how the content should be read, their traversal direction is not consistent, and each unit may contain only one or two words, which makes it hard for humans to reconstruct the underlying meaning. For similar reasons, presenting too much information at once can also increase cognitive workload, especially when the content is not written in a form that is easy to read.

3.3.4 Representing relations among memories. Designing agent memory is not only about expanding what can be stored, but also about how memories and their relations are represented so that the agent can later reconstruct context. Stored information becomes useless if retrieval or relational inference fails, and such retrieval cannot work well without making the relations among memory fragments explicit.

From this perspective, the choice of relational structure introduces downstream trade-offs in retrieval, interaction-time inference, and update workflows. List- or timeline-based memories can be the easiest to navigate because their relational expressiveness is largely confined to ordering, but they offer limited support for preserving richer contextual links across entries. Hierarchical folder structures or grouped sets offer more freedom than flat lists, yet they force each item into a single classification, making nuanced, cross-cutting relations hard to represent and misclassifications costly to recover. Graph-based structures, by contrast, offer the greatest flexibility for representing relations, and a well-maintained graph can enable highly efficient retrieval by traversing explicitly encoded links. However, graph construction and maintenance are expensive, and in practice graphs are often difficult to keep clean enough to realize their theoretical advantages. This overhead becomes especially salient in human–AI collaboration settings, where human attention and reading speed are far slower than agent-side processing.

To mitigate these trade-offs, some systems stack multiple layers rather than committing to a single structure, for example, adding a timeline- and keyword-oriented graph layer over list-like memory entries [18], or overlapping multiple lists and timelines to compensate for sparsity in any one view [65, 66, 75]. Because these structural choices are difficult to revert once a system is implemented and populated with data, it is important to clarify the intended goals and interaction scope early, and to select a representation that matches the expected update and retrieval workflows.

3.3.5 Choice of retrieval mechanisms. Retrieval mechanisms introduce an additional layer of trade-offs and are often combined across layers. Systems commonly pair deterministic methods—such as lexical search, time-based filtering, or graph traversal (e.g., BFS/DFS)—with additional retrieval layers, including embedding-based retrieval, BM25-style lexical retrieval, and/or LLM-mediated query interpretation and reranking. External vector embeddings often have lower build and per-query costs than knowledge-graph approaches, but their induced relational structure is difficult to update. Therefore, for example, even when a user believes A should be closer to C than to B, the system cannot easily "edit" that relationship if it is determined by a fixed embedding space. Embeddings also struggle with context-dependent meaning, such as name disambiguation or indirect references (e.g., "you know who"), where explicit relational representations can be more reliable.

When context sensitivity and controllable relations are central to a system's goals, paying the higher construction and maintenance costs of graph-based structures may be justified. In such settings, using an LLM to construct and operate over the graph can help produce relations that better reflect intended meaning and context. However, designers must anticipate higher and more variable costs (and often less predictable latency) when LLMs are in the loop.

3.3.6 Interdependence between architecture and operations. Importantly, some operations and interaction patterns are contingent on earlier architectural choices. For example, traversing a relation graph and expanding a memory network is not feasible in a list structure because there are no explicit edges to traverse. Likewise, as noted earlier, editing edges is not possible in a list-based structure or when relying on precomputed vector embeddings. More broadly, higher-level capabilities depend on whether nodes and edges carry appropriate payloads: tracing and basic versioning are prerequisites for higher-level version-control operations such as merge, branch, and revert.

Missing capabilities can also implicitly discourage desirable user behavior. Without adequate versioning, users may become reluctant to delete memories, causing the store to grow without pruning. Without appropriate visibility controls, isolation, and mechanisms for managing links between memories, users may instead create private, disconnected stores that never get incorporated into the shared memory, leading to fragmentation and lower overall utility. Similarly, users may want to share a specific part of the memory store but fail to find it due to weak query functionality. In the worst case, the system devolves into a workflow where users rely on a few retrieved snippets and reconstruct the rest in their heads, negating the intended benefits of external memory. In many cases, poor

query performance is itself downstream of structural or operational issues, such as an ill-organized internal representation.

Because these bottlenecks often arise from the interplay between architecture and operations, designing an interactive memory system requires anticipating the interaction flows and operations users will actually perform, and then choosing architectures and operations that best support those flows. This also means considering trade-offs holistically, since the effects of each design choice can compound across the system. It is often valuable to explore multiple design directions and iterate, rather than committing early to a single representation.

3.4 Gaps in Current Dominant Design Choices

3.4.1 Underappreciated coupling between architecture and operation. Some operations in our corpus are not fully supported by the underlying architecture, so they may not produce the interactions users expect. For example, in Memolet [91], grouping or interpolation may seem to affect retrieval, but these interactions primarily reorganize interface-level state while memories remain anchored to fixed vector embeddings. This matters because when user manipulations do not translate into system operations, users may misread a design mismatch as poor performance or a limitation of the AI technology itself. Therefore, designers should recognize that choices of memory architecture and operations involve clear trade-offs that shape user interactions.

3.4.2 Gap in recording relationships among memories. Many systems in our corpus provide limited support for recording and maintaining relations between memories, though such relations are essential for later context reconstruction. For example, triple-based knowledge graphs [18, 40] may lose needed relations because subject-predicate-object encoding strips away nuances in the original expression. Once lost, these cues must be reconstructed at each use, causing repeated inference costs while risking distortion. When constraints allow, designers should preserve richer contextual links within the memory network.

3.4.3 Lack of user-agent common grounding steps. Many systems also lack a process for checking whether the memories and contextual relations inferred by the agent align with what the user actually means. For example, in General User Model [75, p. 16], participants expressed a need for clarification when the system turned observations into propositions or suggestions with low confidence. Without this clarification and grounding step, users may remain uncertain whether the agent's interpretation matches their intent, and the agent may act in ways that conflict with it. Interactive agent memory systems should incorporate clarification and grounding steps into human-AI collaboration workflows to prevent misalignment.

3.4.4 Accumulate-only lifecycle with no versioning. Most systems in our corpus do not support history tracking or version-control operations, and instead adopt a single-version memory lifecycle that is largely accumulate-only, with limited editability and, at most, hard deletion. In such settings, users may focus only on accumulating data while avoiding edits to the memory network to prevent irreversible loss, which can gradually degrade its overall quality. Although any memory might become useful at some point, users should still be able to distinguish higher- and lower-importance memories and move less important ones into a lower-salience space for retrieval only when necessary.

3.4.5 Lack of selective access control. Most systems do not support selective access control for specific memories across different user or agent groups, which becomes a limitation when they are used in collaborative settings. Unless access-control features are added externally, members may end up keeping their own versions of the same memory system, preventing memory from accumulating as a unified team resource. This can interrupt shared accumulation and reuse, and increase the later cost of gathering and reconciling information across individual memory stores. If a system aims to support organization-wide memory, it should therefore support access-control features at multiple levels.

3.4.6 Lack of approval control over memory operations. Many systems provide little support for finely configuring the scope of authority between users and agents over memory operations. As a result, users are often pushed either into unnecessarily handling memory operations manually or into granting the agent too much authority over memory operations. In this case, users may avoid upkeep under excessive manual workload, or allow unintended agent actions by granting more authority than they actually intended to delegate. Memory operations should therefore be paired with richer permission controls so that users can flexibly configure what an agent is allowed to do, under what conditions, and with what oversight.

3.4.7 Gray zone in NLP and HCI. Since system architecture, afforded operations, and resulting interaction patterns are tightly interdependent, interactive AI memory design requires balanced attention to all three. Yet NLP-oriented work often under-specifies users' interactional needs, while HCI-oriented work may underspecify the structural and performance conditions needed to support those interactions robustly. For example, strong retrieval performance does not guarantee a good memory system if users cannot inspect provenance or judge reliability. Conversely, compelling interaction concepts can still fail without sufficient structural support. Because failure on either side can undermine the whole system, designing memory systems for interactive AI agents requires attention from both sides.

4 Metaphors for AI memory

4.1 Dominant Metaphors on AI Memory and Their Gaps

Notably, systems in our corpus repeatedly draw on three dominant metaphorical framings: memory as records in a database, memory as human cognition, and memory as a set of skills and tools. As metaphors shape how designers conceptualize what "memory" should be and should do, and can thereby influence downstream architectural and operational choices [42, 56], we introduce these dominant metaphors and show how their framings are reflected in systems' design choices and rationales.

4.1.1 Records, repositories, and database systems. One dominant framing of AI memory systems is to view them as repositories or database systems that collect and organize records. This framing is

common partly because AI memory systems are built using existing abstractions for computers' memory and persistent storage (e.g., databases, file systems). Some systems adopt this metaphor explicitly. Mem0 [18] and Memolet [91] directly invoke note- and memo-based metaphors, and MemGPT [64] conceptualizes an agent's memory by analogy to a computer's virtual memory. Not limited to these examples, building a reliable repository or database through the long-term accumulation of records is one of the goals of most AI memory systems.

4.1.2 Neurobiology of human cognition. Another dominant framing treats AI memory through metaphors of human cognition. This is unsurprising since Turing's foundational work on machine intelligence framed the problem explicitly in human terms, "Can machines think?" [81], and later efforts have often defined and evaluated AI using criteria based on how well it can imitate human cognitive capacities. Just as AI capabilities are often anthropomorphized [57, 70] and common terms such as learning and training borrow from human ability, AI memory systems and their operations are frequently described using human memory operations, such as learning, memorizing, forgetting, and recalling. The implicit and explicit memory-store metaphor we have used throughout draws on cognitive accounts of human implicit and explicit memory [72], and similar moves recur in AI memory research, where system design is often grounded in analogies to components of human cognition. For example, HippoRAG [40] frames its memory indexing mechanism by analogy to the hippocampus.

4.1.3 Skills and tools. Building on the two major metaphor lineages that frame AI in terms of machine memory and human cognition, recent work increasingly characterizes agent memory as skills or tools that can be equipped and swapped. As LLM capability improves and system/context engineering matures, such memories are treated more as modular, actionable packages consisting of procedures, rules, and tool-usage know-how that reconfigure what the same base model can do. Systems that adopt this metaphor often appear to merge machine and human framings. They manage skills like machine components that are durable, controllable, and swappable on demand, while describing their ongoing creation, maintenance, and use in more human terms, such as "learning" new skills or "holding" different tools. ReAct [89] and Anthropic's Agent Skills [9] provide clear examples of this tool/skill-oriented metaphor for agent memory.

These metaphors help us define and make sense of AI memory, and we also draw on them throughout this paper to explain many concepts. However, they also shape how AI memory is imagined and constituted in practice by "centring particular ideas" while "marginalising others" [56]. When memory is fixed as a record or repository, "the agent" is often reduced to the LLM alone and external memory becomes an auxiliary store rather than a component of the agent system, which can limit the interactive benefits of explicit memory, such as on-the-fly updates and long-horizon context structuring. Conversely, when designers focus too narrowly on reproducing human memory, they may overlook non-human but useful properties of AI memory, such as persistence without forgetting and richer, user- or community-driven forms of memory manipulation. For these metaphors, it is therefore important to reflect on their bounds and assumptions, what features of AI memory they center and marginalize, and what possibilities they open up or constrain for both designers and users [56, pp. 6–8].

4.2 Exploring less-used metaphors

To seed a design space of actions we might perform on memory, we adopt generative metaphor [74] as a design tool [50] to imagine what future AI memory systems could be like. In this chapter, we introduce seven less-used metaphors for AI memory as a way of "connecting previously disassociated domains to generate new ideas" [50], focusing on less-used framings and their implications to expand the space beyond dominant metaphors, with a set of design implications derived from our metaphors.

These alternative metaphors were generated and refined through a researcher-led iterative design process conducted over approximately three months. All authors participated in proposing, discussing, and refining candidate metaphors throughout the broader research process, alongside the work of mapping the design space and completing the architectural and operational surveys. We repeated this process through regular weekly meetings, as well as additional ad hoc meetings and chat-based discussions.

During these iterations, we considered which metaphors to include, exclude, merge, rescope, or further elaborate, and which properties of AI memory each metaphor most productively foregrounded. Across these iterations, we refined both the metaphors themselves and the specific properties and implications associated with them, aiming to identify metaphors that foregrounded distinct and useful qualities of AI memory, could inspire concrete design directions, and did not substantially overlap with the dominant metaphors already shaping the space.

Some candidate metaphors, such as software version control [77], Zettelkasten [51, 52], requirements [84], and science fiction tales/games, were already part of our early framing of AI memory and had recognizable, if non-dominant, precedents in adjacent research and practice. Software version control, for example, has long informed workspace management in agentic coding systems [3, 24, 59], even if it has not typically been framed as a metaphor for AI memory itself. Requirements likewise shares important affinities with prompt engineering [17, 63], context engineering, and intent specification [83]. Zettelkasten also had precedent in the space, including in systems such as A-MEM [88] that draw on it directly as a metaphor for its memory system. Science-fictional framings, while not commonly used as metaphors for AI memory itself, have often been used to imagine AI agents, their roles, and their interaction settings more broadly. Other metaphors, including personal diary and record [47], and community archive [27], and cultural probes [28], were newly proposed here as metaphors for AI memory and were further developed through this iterative process by extending adjacent discussions in HCI.

Candidates were excluded when they overlapped too heavily with other metaphors, were too broad or too narrow to be generatively useful, or led to implications that felt overly speculative or insufficiently actionable. For instance, an early metaphor framing AI memory as a library was not retained as a final metaphor because its design implications could be more clearly and productively

distributed across memory as records, repositories, and database systems and memory as a community archive.

We also adjusted the scope of candidate metaphors when needed. For example, a metaphor initially framed around Git [30] was broadened to software version control to avoid an overly narrow association with one specific tool, while a broader framing around fiction was narrowed to science fiction novels and games to better match the intended design implications and reduce ambiguity. The final set was selected based on whether each metaphor (1) foregrounded a distinct and useful property of AI memory, (2) was appropriately scoped for design discussion, (3) generated concrete interaction or design implications, and (4) was not overly redundant with either dominant existing metaphors or other candidates.

4.2.1 Software version control. The version control systems (VCS), such as git, used to track a project's commit history [77] are crucial repositories of memory about the project's evolution [23, 41] that can be leveraged for a huge variety of tasks [35].

- *Higher-level memory manipulation.* Versioning enables higher-level operations over AI memory, such as diffing, merging, branching, reverting, and rebasing.
- *Revertibility.* Versioning makes memory revertible, lowering the friction to experiment with different approaches.
- *Traceability and accountability.* Versioning improves transparency and accountability by recording who did what, when, and why.

Implications. Software version control is useful not only for coding but also for many tasks that require iterative refinement, such as writing, design, and planning. Memory systems can support this process by adding version-control operations such as diff, branch, merge, and revert. By reducing users' fear of information loss when memory is modified, these features can help them explore more alternatives and collaborate with the agent system more confidently. Directly adopting version-control systems is one strong option, but for broader users, drawing on save, versioning, and history-tracking interfaces familiar from operating systems, cloud services, and commercial tools may also be a good option.

4.2.2 Zettelkasten. The Zettelkasten ("slip-box") is a personal knowledge management method developed by Niklas Luhmann [51, 52]. Each day, he captured atomic thoughts on slips, assigned unique IDs, linked them to related notes, and filed them into a growing network.

- *Atomicity of unit thoughts.* Each slip should capture a single atomic idea, so later citations point unambiguously to one claim [73, p. 203, 306].
- *Looseness.* The network stays deliberately loose, tolerating overlap that later reveals each entry's value [73, p. 300].
- *Human-in-the-loop.* Notes are staged and placed manually to vet atomicity and preserve long-term quality [73, p. 300] [51, 52].
- *Tangibility and spatial manipulability.* Physical notes enable hand-to-hand manipulation, retrieval, storage, and sharing.
- *Reproducibility.* Zettelkasten externalizes connections between ideas, keeping their context retrievable and reconstructible over time.

Implications. Zettelkasten is useful when an AI memory system needs to preserve rich, reconstructible context across many loosely connected ideas. This would require representing memory as atomized nodes with richer node and edge payloads, together with operations for editing, linking, scoping, and traversing the network. Such a structure can better preserve contextual relations while remaining more readable and editable for both users and agents than highly flattened or overly granular representations. At the same time, designers should consider that exposing too much of such dense memory structures at once could easily overwhelm users. Useful inspiration may come from wiki-like or recommendation-based interfaces that reveal only the most relevant linked ideas first and let surrounding context be expanded on demand.

4.2.3 Requirements. Requirements [84] describe what functions a system should perform, agreed-upon constraints, its non-functional priorities, and the scope and assumptions under which it should operate. Depending on the intended audience, requirements can be expressed at different degrees of formality [15] to negotiate, develop, and document shared understanding. They are also used as run-time components in self-adaptive systems [71].

- *Shared grounding and interpretability.* Requirements should be clear, legible, and consistently interpretable by relevant stakeholders at an appropriate level of abstraction.
- *Reducing redundancy and ambiguity.* Requirements aim to minimize ambiguity and unnecessary duplication.
- *Verifiability.* Verifiability should be treated seriously from the start when defining requirements.
- *Normative reference.* Well-formed requirements aim to be a stable reference that can later govern decisions.
- *Conflict-free.* Requirements should not conflict with one another.
- *Providing rationale.* Requirements should capture not only *what* is required but also *why*.

Implications. Requirements can serve as a productive metaphor for collaborative situations in which memory is meant to guide or constrain later agent behavior. Supporting this direction would require list- or table-based requirement items plus semantic operations for checking conflict, satisfaction, and rationale, along with retrieval and reasoning support for reconstructing relevant context. This requirements-based treatment of memory can make user intentions and expectations more explicit, inspectable, and revisable. Useful interfaces include table-based views that support comparison, matching, and verification. Evaluation tools for prompts, memories, or policies, such as ChainForge [10], Semantic Commit [83], and Policy Maps [43], may likewise offer useful interface ideas.

4.2.4 Personal diary and record. AI agents are often used as a confidant for thoughts, secrets, or emotions that users would not easily disclose to others [47]. In such cases, AI agents and their memory systems should have different interaction and operational characteristics than when dealing with public information.

- *Honesty and privacy.* Because diaries presume no external audience, they invite candor but demand stricter privacy and disclosure controls.
- *Entanglement of the private and the public.* Diary entries often interweave private details with public and social contexts,

making shareability audience- and context-dependent and shifting over time.
- *Personal record for later reflection.* Diaries create a persistent record of an "observed self" that can be revisited for reflection, sometimes revealing aspects that differ from one's usual self-concept.

Implications. An agent working with external systems while holding a user's sensitive memory is like someone entrusted with the user's diary and discretion over what to share. Such an agent would need stronger access separation, selective disclosure controls, and richer memory representations that preserve the contextual nuance needed to keep its judgments aligned with the user's intent. With these conditions, users could ground deeper information in memory and delegate higher-stakes tasks more confidently. They would then need interfaces for inspecting representations, checking context-dependent privacy boundaries, and tracing downstream effects as memories change. Useful interface directions may include compare views, policy panels, and impact-analysis workflows.

4.2.5 Community archive. AI memory itself can be metaphorically framed as a community archive [27] because contributions to an agent's memory are not bound to a single user and can instead be formed through interactions and contributions from multiple participants.

- *Transmission and re-access/reconstruction.* Archives preserve records for future members and enable ongoing return, reinterpretation, and reconstruction over time.
- *Voluntary participation and production.* Community archives are driven from within the community, requiring member participation, control, and ownership in what gets documented [26, p. 153].
- *Cross-referencing and knowledge networks.* As records accumulate, they increasingly cross-reference and link to one another, forming a contextualized knowledge network.

Implications. Treating memory as a community archive is useful in group settings where relevant traces are scattered across people and places, yet the group still needs a durable shared record. Meanwhile, building such an archive requires not only a collective space but also clearly bounded structures so that personal and collective memory do not become unintentionally conflated. It also requires operations for semantic duplicate detection and evaluation of the usefulness of new contributions. This can shift more manual filing and retrieval work to the agent, letting members focus more on meaningful contribution to and use of the archive. Such archive interfaces may range from more conventional shared repositories with attached editors, discussion views, or terminal panels to forms embedded closer to group activity, such as chat-integrated agents in Social-RAG [86] and CHOIR [46].

4.2.6 Cultural Probes. Cultural Probes [28] is a method for catalyzing design ideation. Rather than asking direct needs-based questions, it uses ambiguous, indirect cues that leave room for interpretation.

- *Fragmentary cues as catalysts.* Cultural probes use fragments to spark recall and ideation.
- *Subjectivity of uptake and interpretation.* The same cue can be interpreted differently across people and over time.
- *Creativity as a generative process.* Creativity often arises by recombining new stimuli with prior knowledge [55].

Implications. Cultural probes are useful for ideation settings in which an agent must generate outputs that respond to the user's contextual constraints, avoid merely repeating what already exists, and still become meaningfully novel. This requires richer contextual memory, strong retrieval and reasoning, and enough autonomy to maintain evolving context, identify underexplored areas, and surface directions worth testing. Under these conditions, suggestions can act as probe-like stimuli for users, for the agent itself, or even across multiple agents catalyzing one another. Therefore, to let users perceive and inspect this process, interfaces should expose provenance, reasoning paths, and citation-like links without making these relations too abstract or overly dense.

4.2.7 Science fiction tales/games. AI agents and their memory can also be framed through science fiction and games, since many agent settings contain partly fictional elements that should be considered in their design.

- *Fictionality.* In explicitly fictional settings, an agent's memory and actions need not mirror real-world facts [85].
- *Verisimilitude within the world.* Even if not realistic, events should remain plausible within the world's internal logic to keep the setting coherent [58].
- *Narrative coherence and justification.* Narrative coherence matters more than factual completeness. Interactions should maintain context and provide "why" to sustain immersion.
- *The world's continuity.* Long-running interactions should preserve continuity, coherence, and the world's verisimilitude over time.

Implications. Fiction metaphors are useful not only in fictional settings but also in partially fictionalized real-world domains such as branding and storytelling. Sustaining such narratives over time requires long-lived, extensible memory that preserves links across entities, events, and temporal phases while remaining navigable as these accumulate. It also requires operations for tracking narrative change, detecting and repairing inconsistency, and supporting reframing as the narrative evolves through internal development or external intervention. This helps sustain a distinctive narrative over time without letting it fracture through serious inconsistency. Useful interfaces may therefore include timelines, key-event chronologies, time-based graphs, and entity- or keyword-centered wiki-like views for inspecting narrative change over time.

5 Conclusion

This study investigated the design space and metaphors employed by current AI memory systems to enable more effective human-AI interaction. We proposed a design space account of interactive AI memory systems across architecture, operations, and interface. Using this framework as an analytic lens, we examined how existing systems instantiate memory design choices, highlighted dominant patterns, and surfaced gaps that constrain interaction capabilities or introduce practical trade-offs. We also applied generative metaphorical design to widen what "AI memory" can imply, proposing alternative less-used metaphors that suggest new directions for designing future interactive memory systems.

Our work can help practitioners design interactive AI memory systems by making appropriate design choices that fit their intended goals. For those analyzing and improving existing tools, our approach can also clarify the implications of specific design choices and make gaps more visible. For researchers spanning HCI and NLP, our account offers a shared vocabulary for recognizing this gray zone and for reasoning about memory as a system-level design problem, which can in turn seed new research ideas.

Interactive AI memory system design remains at an early stage, with new techniques and ideas continuing to emerge. As this area develops, we expect the design space to be explored in broader ways than what we articulated here. We hope this work helps open further possibilities and new discussions about how agent memory should be designed, governed, and experienced.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. 2024. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363* (2024).

[3] Anthropic. [n. d.]. Claude Code Overview. https://code.claude.com/docs/en/overview. Claude Code Docs. Accessed: 2026-03-30.

[4] Anthropic. 2025. Agent Skills. Claude Code Docs. https://code.claude.com/docs/en/skills Accessed: 2026-01-13.

[5] Anthropic. 2025. Claude Code. https://claude.com/product/claude-code. Accessed: 2025-12-12.

[6] Anthropic. 2025. *Claude Opus 4.5 System Card.* Technical Report. Anthropic. https://assets.anthropic.com/m/64823ba7485345a7/Claude-Opus-4-5-System-Card.pdf System card. Dated November 2025..

[7] Anthropic. 2025. *Claude Sonnet 4.5 System Card.* Technical Report. Anthropic. https://assets.anthropic.com/m/12f214efcc2f457a/original/Claude-Sonnet-4-5-System-Card.pdf System card. Dated September 2025..

[8] Anthropic. 2025. Create custom subagents (Claude Code Docs). https://code.claude.com/docs/en/sub-agents. Accessed: 2026-01-16.

[9] Anthropic. 2025. Equipping Agents for the Real World with Agent Skills. Anthropic Engineering blog post. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills Published: 2025-10-16; Accessed: 2026-01-15.

[10] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena L Glassman. 2024. Chainforge: A visual toolkit for prompt engineering and llm hypothesis testing. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–18.

[11] Gagan Bansal, Jennifer Wortman Vaughan, Saleema Amershi, Eric Horvitz, Adam Fourney, Hussein Mozannar, Victor Dibia, and Daniel S Weld. 2024. Challenges in human-agent communication. *arXiv preprint arXiv:2412.10380* (2024).

[12] Dave Bergmann. 2024. What is a Context Window? IBM Think. https://www.ibm.com/think/topics/context-window Accessed: 2025-12-12.

[13] Jeffrey R Binder and Rutvik H Desai. 2011. The neurobiology of semantic memory. *Trends in cognitive sciences* 15, 11 (2011), 527–536.

[14] Michael Mose Biskjaer, Peter Dalsgaard, and Kim Halskov. 2014. A constraint-based understanding of design spaces. In *Proceedings of the 2014 conference on Designing interactive systems*. 453–462.

[15] Jean-Michel Bruel, Sophie Ebersold, Florian Galinier, Manuel Mazzara, Alexandr Naumchev, and Bertrand Meyer. 2021. The Role of Formalism in System Requirements. *ACM Comput. Surv.* 54, 5, Article 93 (May 2021), 36 pages. doi:10.1145/3448975

[16] Greg Ceccarelli. 2024. SpecStory Launch. SpecStory Blog. https://specstory.com/blog/specstory-launch Published: 2024-12-16. Accessed: 2026-01-15.

[17] Boris Cherny. 2025. Claude Code: Best practices for agentic coding. https://www.anthropic.com/engineering/claude-code-best-practices. Published: 2025-04-18. Accessed: 2025-12-12.

[18] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413* (2025).

[19] John Joon Young Chung, Wooseok Kim, Kang Min Yoo, Hwaran Lee, Eytan Adar, and Minsuk Chang. 2022. TaleBrush: Sketching Stories with Generative Pretrained Language Models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 209, 19 pages. doi:10.1145/3491102.3501819

[20] Herbert H Clark. 1996. *Using language*. Cambridge University Press.

[21] Herbert H Clark and Susan E Brennan. 1991. Grounding in communication.

[22] Neal J Cohen and Larry R Squire. 1980. Preserved learning and retention of pattern-analyzing skill in amnesia: Dissociation of knowing how and knowing that. *Science* 210, 4466 (1980), 207–210.

[23] D. Cubranic, G.C. Murphy, J. Singer, and K.S. Booth. 2005. Hipikat: a project memory for software development. *IEEE Transactions on Software Engineering* 31, 6 (2005), 446–465. doi:10.1109/TSE.2005.71

[24] Cursor. [n. d.]. Git. https://cursor.com/help/integrations/git. Cursor Docs. Accessed: 2026-03-30.

[25] Cursor. 2023. Cursor. https://cursor.com/. Accessed: 2025-12-12.

[26] Andrew Flinn. 2007. Community histories, community archives: Some opportunities and challenges. *Journal of the Society of Archivists* 28, 2 (2007), 151–176.

[27] Andrew Flinn. 2015. Community Archives. In *Encyclopedia of Archival Science*, Luciana Duranti and Patricia C. Franks (Eds.). Rowman & Littlefield, Lanham, MD, 145–149.

[28] Bill Gaver, Tony Dunne, and Elena Pacenti. 1999. Design: cultural probes. *interactions* 6, 1 (1999), 21–29.

[29] Gemini Team. 2025. *Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities.* Technical Report. Google. https://storage.googleapis.com/deepmind-media/gemini/gemini_v2_5_report.pdf Technical report..

[30] Git Project. [n. d.]. Git. https://git-scm.com/. Official website. Accessed: 2026-03-30.

[31] GitHub. 2022. GitHub Copilot. https://github.com/features/copilot. Accessed: 2025-12-12.

[32] Google Cloud. 2025. Gemini CLI. https://docs.cloud.google.com/gemini/docs/codeassist/gemini-cli. Last updated: 2025-12-11 (UTC). Accessed: 2025-12-12.

[33] Google for Developers. 2025. Prompt Engineering for Generative AI. https://developers.google.com/machine-learning/resources/prompt-eng. Last updated: 2025-08-25 (UTC). Accessed: 2025-12-12.

[34] Prahlad Gupta and Neal J Cohen. 2002. Theoretical and computational analysis of skill learning, repetition priming, and procedural memory. *Psychological review* 109, 2 (2002), 401.

[35] Ahmed E. Hassan. 2008. The road ahead for Mining Software Repositories. In *2008 Frontiers of Software Maintenance*. 48–57. doi:10.1109/FOSM.2008.4659248

[36] Taher H Haveliwala. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*. 517–526.

[37] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[38] Ziheng Huang, Sebastian Gutierrez, Hemanth Kamana, and Stephen MacNeil. 2023. Memory sandbox: Transparent and interactive memory management for conversational agents. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–3.

[39] Peiling Jiang, Jude Rayan, Steven P Dow, and Haijun Xia. 2023. Graphologue: Exploring large language model responses with interactive diagrams. In *Proceedings of the 36th annual ACM symposium on user interface software and technology*. 1–20.

[40] Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems* 37 (2024), 59532–59569.

[41] Sunghun Kim, Kai Pan, and E. E. James Whitehead. 2006. Memories of bug fixes. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Portland, Oregon, USA) *(SIGSOFT '06/FSE-14)*. Association for Computing Machinery, New York, NY, USA, 35–45. doi:10.1145/1181775.1181781

[42] George Lakoff and Mark Johnson. 2003. *Metaphors We Live By*. University of Chicago Press, Chicago. Originally published 1980; updated edition with afterword.

[43] Michelle S. Lam, Fred Hohman, Dominik Moritz, Jeffrey P Bigham, Kenneth Holstein, and Mary Beth Kery. 2025. Policy Maps: Tools for Guiding the Unbounded Space of LLM Behaviors. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25)*. Association for Computing Machinery, New York, NY, USA, Article 178, 24 pages. doi:10.1145/3746059.3747680

[44] LangChain. 2024. LangGraph overview. https://docs.langchain.com/oss/python/langgraph/overview. Accessed: 2025-12-12.

[45] LangChain-AI. 2025. LangMem. https://langchain-ai.github.io/langmem/. Accessed: 2025-12-12.

[46] Sangwook Lee, Adnan Abbas, Yan Chen, Young-Ho Kim, and Sang Won Lee. 2026. CHOIR: A Chatbot-mediated Organizational Memory Leveraging Communication in University Research Labs. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery. doi:10.1145/3772318.3791314 To appear.

[47] Yi-Chieh Lee, Naomi Yamashita, Yun Huang, and Wai Fu. 2020. "I Hear You, I Feel You": Encouraging Deep Self-disclosure through a Chatbot. In *Proceedings*

of the 2020 CHI Conference on Human Factors in Computing Systems (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. doi:10.1145/3313831.3376175

[48] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[49] Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, et al. 2025. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems. *arXiv preprint arXiv:2504.01990* (2025), 20.

[50] Nick Logler, Daisy Yoo, and Batya Friedman. 2018. Metaphor Cards: A How-to-Guide for Making and Using a Generative Metaphorical Design Toolkit. In *Proceedings of the 2018 Designing Interactive Systems Conference* (Hong Kong, China) *(DIS '18)*. Association for Computing Machinery, New York, NY, USA, 1373–1386. doi:10.1145/3196709.3196811

[51] Niklas Luhmann. 1981. Kommunikation mit Zettelkästen: Ein Erfahrungsbericht. In *Öffentliche Meinung und sozialer Wandel/Public Opinion and Social Change*. Springer, 222–228.

[52] Niklas Luhmann. 2023. *Communicating with Slip Boxes: An Empirical Account*. https://zettelkasten.de/communications-with-zettelkastens/ English translation of Luhmann (1981), available online.

[53] Shuai Ma, Qiaoyi Chen, Xinru Wang, Chengbo Zheng, Zhenhui Peng, Ming Yin, and Xiaojuan Ma. 2025. Towards human-ai deliberation: Design and evaluation of llm-empowered deliberative ai for ai-assisted decision-making. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–23.

[54] Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. 1991. Questions, Options, and Criteria: Elements of Design Space Analysis. *Human–Computer Interaction* 6, 3-4 (1991), 201–250. doi:10.1080/07370024.1991.9667168

[55] Sarnoff Mednick. 1962. The associative basis of the creative process. *Psychological review* 69, 3 (1962), 220.

[56] Dave Murray-Rust, Iohanna Nicenboim, and Dan Lockton. 2022. Metaphors for designers working with AI. *DRS2022: Bilbao* (2022).

[57] Clifford Nass, Jonathan Steuer, and Ellen R Tauber. 1994. Computers are social actors. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 72–78.

[58] Graham Oddie and Gustavo Cevolani. 2022. Truthlikeness. In *The Stanford Encyclopedia of Philosophy* (Winter 2022 ed.), Edward N. Zalta and Uri Nodelman (Eds.). Metaphysics Research Lab, Stanford University.

[59] OpenAI. [n. d.]. Use Codex in GitHub. https://developers.openai.com/codex/integrations/github. OpenAI Developers. Accessed: 2026-03-30.

[60] OpenAI. 2022. ChatGPT | AI Chatbot to Discover, Learn & Create. Web page. https://openai.com/chatgpt/overview/ Accessed: 2025-12-12.

[61] OpenAI. 2025. Codex. https://openai.com/codex/. Accessed: 2025-12-12.

[62] OpenAI. 2025. *GPT-5 System Card*. Technical Report. OpenAI. https://cdn.openai.com/gpt-5-system-card.pdf System card. Published August 13, 2025..

[63] OpenAI. 2026. Best practices for prompt engineering with the OpenAI API. https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api. Accessed: 2026-04-06.

[64] Charles Packer, Zeyuan Sun, Max Goldstein, Avishkar Basu, Soham Choudhury, Amog Kamsetty Pasarkar, Emma Strubell, and Andrew McCallum. 2023. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560* (2023). https://arxiv.org/abs/2310.08560

[65] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*. 1–22.

[66] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956* (2025).

[67] Partha Pratim Ray. 2025. A Review on Vibe Coding: Fundamentals, State-of-the-art, Challenges and Future Directions. *Authorea Preprints* (2025).

[68] Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond*. Vol. 4. Now Publishers Inc.

[69] Edmund T Rolls. 2024. The memory systems of the human brain and generative artificial intelligence. *Heliyon* 10, 11 (2024).

[70] Arleen Salles, Kathinka Evers, and Michele Farisco. 2020. Anthropomorphism in AI. *AJOB neuroscience* 11, 2 (2020), 88–95.

[71] Pete Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. 2010. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *2010 18th IEEE International Requirements Engineering Conference*. 95–103. doi:10.1109/RE.2010.21

[72] Daniel L. Schacter. 1987. Implicit Memory: History and Current Status. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 13, 3 (1987), 501–518. doi:10.1037/0278-7393.13.3.501

[73] Johannes Schmidt. 2016. Niklas Luhmann's Card Index: Thinking Tool, Communication Partner, Publication Machine. In *Forgetting Machines. Knowledge Management Evolution in Early Modern Europe*. Vol. 53.

[74] Donald A Schön. 1979. Generative metaphor: A perspective on problem-setting in social policy. *Metaphor and thought* 2 (1979), 137–163.

[75] Omar Shaikh, Shardul Sapkota, Shan Rizvi, Eric Horvitz, Joon Sung Park, Diyi Yang, and Michael S Bernstein. 2025. Creating general user models from computer use. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*. 1–23.

[76] Yijia Shao, Vinay Samuel, Yucheng Jiang, John Yang, and Diyi Yang. 2024. Collaborative gym: A framework for enabling and evaluating human-agent collaboration. *arXiv preprint arXiv:2412.15701* (2024).

[77] D. Spinellis. 2005. Version control systems. *IEEE Software* 22, 5 (2005), 108–109. doi:10.1109/MS.2005.140

[78] Sangho Suh, Bryan Min, Srishti Palani, and Haijun Xia. 2023. Sensecape: Enabling multilevel exploration and sensemaking with large language models. In *Proceedings of the 36th annual ACM symposium on user interface software and technology*. 1–18.

[79] Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *Transactions on Machine Learning Research* (2023).

[80] Endel Tulving. 2002. Episodic memory: From mind to brain. *Annual review of psychology* 53, 1 (2002), 1–25.

[81] A. M. Turing. 1950. Computing Machinery and Intelligence. *Mind* 59, 236 (Oct. 1950), 433–460. doi:10.1093/mind/LIX.236.433

[82] Priyan Vaithilingam, Ian Arawjo, and Elena L Glassman. 2024. Imagining a future of designing with ai: Dynamic grounding, constructive negotiation, and sustainable motivation. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*. 289–300.

[83] Priyan Vaithilingam, Munyeong Kim, Frida-Cecilia Acosta-Parenteau, Daniel Lee, Amine Mhedhbi, Elena L Glassman, and Ian Arawjo. 2025. Semantic Commit: Helping Users Update Intent Specifications for AI Memory at Scale. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*. 1–18.

[84] Axel Van Lamsweerde. 2009. *Requirements engineering: From system goals to UML models to software*. Vol. 10. Chichester, UK: John Wiley & Sons.

[85] Kendall L Walton. 1993. *Mimesis as make-believe: On the foundations of the representational arts*. Harvard University Press.

[86] Ruotong Wang, Xinyi Zhou, Lin Qiu, Joseph Chee Chang, Jonathan Bragg, and Amy X. Zhang. 2025. Social-RAG: Retrieving from Group Interactions to Socially Ground AI Generation. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 162, 25 pages. doi:10.1145/3706598.3713749

[87] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[88] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110* (2025).

[89] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X

[90] Catherine Yeh, Gonzalo Ramos, Rachel Ng, Andy Huntington, and Richard Banks. 2024. Ghostwriter: Augmenting collaborative human-ai writing experiences through personalization and agency. *arXiv preprint arXiv:2402.08855* (2024).

[91] Ryan Yen and Jian Zhao. 2024. Memolet: Reifying the reuse of user-ai conversational memories. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–22.

[92] JD Zamfirescu-Pereira, Eunice Jun, Michael Terry, Qian Yang, and Björn Hartmann. 2025. Beyond code generation: Llm-supported exploration of the program design space. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–17.

[93] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems* 43, 6 (2025), 1–47.

[94] Dora Zhao, Diyi Yang, and Michael S Bernstein. 2025. Knoll: Creating a knowledge ecosystem for large language models. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*. 1–23.