Managing What to Remember: A Semantics–Realization–Dynamics Framework for External Long-Term Memory Systems in Large Language Model Agents

Anonymous authors
Paper under double-blind review

Abstract

External memory systems have become a core component of Large Language Model (LLM) agents for supporting long-context reasoning, personalization, continual interaction, and persistent task execution. Existing abstraction frameworks mainly separate representation from implementation, but do not explicitly isolate the operational memory mechanisms through which memory is accessed, selected, indexed, updated, consolidated, forgotten, and governed during agent execution. As a result, important runtime memory behaviors are often mixed with representation or storage concerns, making systematic analysis of external memory architectures difficult. This survey analyzes 81 external memory systems for LLM agents through a three-layer architectural decomposition consisting of the Memory Semantics Layer, the Memory Realization Layer, and the Memory Dynamics Layer. The Semantics Layer characterizes what memory units represent and how they are logically structured. The Realization Layer characterizes how memory is physically instantiated across storage substrates and infrastructural backends. The Dynamics Layer characterizes how memory is operated during runtime. Using this decomposition, the survey identifies several architectural patterns and imbalances across current systems. Procedural memory is largely absent from structured representational forms despite the prevalence of structured declarative memory. Sparse indexing appears in only 1 of 81 surveyed systems, with most architectures heavily favoring dense retrieval mechanisms. In addition, forgetting operations are primarily governed through heuristic policies, while consolidation mechanisms are predominantly model-driven, suggesting a systematic asymmetry between memory removal and memory integration strategies in current LLM external memory architectures. By separating memory operation, memory meaning, and physical realization into interacting architectural layers, this survey provides a unified framework for analyzing external memory systems and clarifies several open challenges in scalable long-term memory management for LLM agents.

1 Introduction

The effectiveness of a Large Language Model (LLM)-based agent depends not only on the capabilities of the underlying model but on its ability to accumulate, organize, and selectively access information across extended interaction horizons. LLMs face significant challenges in sustaining long-term memory for agentic applications due to limited context windows, and this constraint becomes acute in precisely the settings where agents are most valuable: multi-session personal assistants that must remember user preferences across weeks of interaction, task-oriented coding agents that must recall project-specific conventions and prior debugging decisions, and open-world game-playing agents that must build on skills acquired across thousands of episodes[1], [2]. In each case, the inability to retain information beyond the current context window is not a peripheral inconvenience but a structural barrier to adaptive, coherent behavior over time.

External memory systems address this barrier by persisting information outside the model's context window and retrieving it on demand. The most widely deployed form of such systems, retrieval-augmented generation (RAG), has demonstrated substantial value in grounding model outputs in externally stored knowledge[3].

However, RAG addresses only one dimension of the memory problem, namely retrieval, leaving open the questions of what units of information should be stored and in what form, how those units should be physically persisted and organized for efficient access, and how they should be governed over the course of extended use. A true external memory system must answer all three questions, not only the last. This survey is organized around the recognition that these three questions are analytically separable and that conflating them, as most existing treatments do, obscures the design trade-offs that determine whether a memory system succeeds in practice.

We cover 81 method-oriented papers on external long-term memory for LLM-based agents, published through early 2026, and organize our analysis around a three-layer decomposition of the design space. The decomposition is grounded in two theoretical precedents from established disciplines. The ANSI/SPARC three-schema architecture establishes the principle that any persistent data system benefits from separating the logical definition of stored entities from their physical realization, allowing each to evolve independently. Our first two layers instantiate this principle directly: the Semantics Layer defines what memory units exist and in what logical form, and the Realization Layer specifies how they are physically persisted and accessed, independently of that logical definition[4]. Marr's tri-level analysis argues that any information-processing system must be understood at the levels of representation, implementation, and computational control, and that understanding at one level does not substitute for understanding at another[5]. Both frameworks converge on the same architectural prescription: the design of any persistent information-processing system requires reasoning separately about what is stored, how it is stored and accessed, and how it is operated upon over time.

We instantiate this prescription as three layers. The Semantics Layer characterizes what units of memory mean and how they are logically encoded along two orthogonal axes: the cognitive role the unit serves, following the taxonomy of CoALA[6], and the structural form in which it is encoded, following and extending the taxonomy of Zeng et al.[7]. The Realization Layer characterizes how memory units are physically persisted and made accessible through substrates and access structures, across two independently variable sub-dimensions. The four storage substrate types extend the canonical NoSQL classification of Cattell[8], adding vector stores as a substrate type that emerged with the adoption of neural encoders. The five indexing structure types are grounded in the foundational sparse-dense distinction established by Lin et al.[9], supplemented by graph traversal, metadata filtering, and hierarchical indexing structures identified across the surveyed systems. The orthogonality of the two sub-dimensions is supported empirically by Zeng et al.[7], who demonstrate that structural form choices and retrieval method choices produce independently measurable effects on task performance. The Dynamics Layer characterizes the six lifecycle operations through which memory is written, updated, forgotten, retrieved, assembled, and consolidated, and the four governance policy types through which those operations are triggered and regulated. The six-operation vocabulary synthesizes and revises prior operational taxonomies: Zhang et al.[10] organize memory activity into three coarse stages of writing, management, and reading, while Du et al.[11] enumerate six atomic operations including Indexing and Compression. We depart from Du et al. in two respects. Indexing is relocated to the Realization Layer on the grounds that it is the construction of a static access structure rather than a dynamic operation on memory content, and Compression is replaced by Assemble, defined as the composition of retrieved memory items into a context suitable for LLM input, which better reflects the operation as it appears in the surveyed systems. The four governance policy types, rule-based, heuristic-based, model-driven, and learned/adaptive, are derived from a convergence of findings across several recent works: Sun et al.[12] distinguish the dominant heuristic paradigm from decision-theoretic and learned approaches; Zhang et al.[13] identify the transition from static rule-based governance to adaptive self-refining systems; and A-MAC[14] contrasts hand-crafted heuristic scoring with LLM-native model-driven judgment. The governance policies are not a parallel classification to the lifecycle operations but the decision mechanisms that govern when and how each operation is invoked, a relationship supported by the empirical finding that 28 of the 81 surveyed systems apply different governance types to different operations within the same system.

This decomposition addresses three design questions that existing frameworks tend to conflate. First, indexing, the construction of a static access structure, and retrieval, the dynamic execution of a query against that structure, differ in temporal character, computational cost, and optimization target; our decomposition assigns them to separate layers rather than treating them as operations of the same kind. Second, the logical

form of a memory unit and the physical substrate that persists it are independently variable: the same atomic fact can be stored in a vector store, a key-value table, or a graph database, and the same vector store can hold chunks, summaries, or atomic fact embeddings; our decomposition makes this many-to-many relationship explicit. Third, governance policy operates at the operation level rather than the system level: analysis of 81 surveyed systems reveals that 28 apply different governance types to different operations within the same system, a variation that system-level characterizations cannot capture.

Beyond the three-layer decomposition, the survey contributes three empirical findings derived from systematic analysis of the 81 surveyed method papers. First, the cross-product matrix of cognitive role and structural form reveals that procedural memory is almost entirely absent from the four declarative structural forms identified by Zeng et al. and is instead realized in executable code, a structural regime orthogonal to the textual and relational forms that dominate episodic and semantic memory research. Second, among the five indexing structure types identified in the surveyed literature, sparse keyword indexing appears in only 1 of 81 systems, despite documented evidence that sparse methods outperform dense embedding retrieval on exact-match and out-of-distribution queries[15], representing a systematic gap in the field's exploitation of the available retrieval toolkit. Third, analysis of per-operation governance reveals a structural asymmetry: Consolidate is governed by model-driven policies in 93 percent of systems that implement it, while Forget is governed by heuristic-based policies in 69 percent, the least adaptive governance type applied to the operation with the most consequential long-term effects on memory coherence.

The remainder of this paper is organized as follows. Section 2 situates our contribution within the existing literature on agent memory and identifies the specific gaps that motivate our analytical choices. Section 3 develops the theoretical and empirical motivation for the three-layer decomposition. Section 4 presents the Semantics Layer, including the two-axis representation matrix and its empirical instantiation across surveyed systems. Section 5 presents the Realization Layer, analyzing substrate and indexing structure choices and their interactions. Section 6 presents the Dynamics Layer, characterizing the six lifecycle operations and four governance policy types and their empirical distribution. Section 7 identifies open challenges surfaced by the three-layer analysis. Section 8 concludes.

2 Related Work

Research on memory for LLM-based agents has expanded rapidly since 2023, producing a heterogeneous body of surveys that differ in scope, classification axis, and analytical focus. We organize the related work into four groups according to the primary dimension along which each survey organizes its analysis, and we situate our own contribution relative to each group.

2.1 Operational Taxonomies

The most widely cited prior survey, Zhang et al.[10], organizes memory mechanisms around a write-manage-read framework, distinguishing the stage at which information enters the memory store, the stage at which it is maintained and transformed, and the stage at which it is accessed. This framework provides a useful high-level vocabulary but, as Du et al.[11] note, treats management as a coarse intermediate stage that subsumes finer-grained operations such as forgetting, consolidation, and indexing without distinguishing their different functional roles or governance requirements. Du et al.[11] refine the operational taxonomy into six atomic operations, Consolidation, Updating, Indexing, Forgetting, Retrieval, and Compression, and map them to emerging research directions. Our survey departs from both in two respects. First, we relocate Indexing from the operational tier to the Realization Layer, on the grounds that indexing is the construction and maintenance of a static access structure over stored content, whereas retrieval is the dynamic execution of a query against that structure. The two differ in temporal character, computational cost, and optimization target: indexing is performed at write time or during consolidation and determines what retrieval modalities are available, while retrieval is performed on demand and determines which available modality is applied to a given query. Treating them as operations of the same kind, as Zhang et al. and Du et al. do, conflates a structural design decision with a runtime behavioral decision and makes it impossible to reason about index design and query strategy independently. Second, we analyze governance policy at the operation level rather than at the system level. A system-level characterization, labeling a system as heuristic-based or

model-driven in the aggregate, collapses the intra-system variation that our empirical analysis reveals: 28 of the 81 surveyed systems apply different governance types to different operations, and the distribution is highly non-uniform, with Consolidate governed by model-driven policies in 93 percent of implementing systems and Forget governed by heuristic-based policies in 69 percent. Operation-level analysis makes this variation visible and provides a more accurate account of how governance decisions are actually made in deployed systems.

2.2 Multi-Dimensional Classification Taxonomies

Several surveys have moved beyond single-axis classification toward multi-dimensional taxonomies. Wu et al.[16] propose a three-dimensional, eight-quadrant taxonomy that organizes AI memory along the dimensions of object (personal versus system), form (parametric versus non-parametric), and time (short-term versus long-term). He et al.[17] ground their taxonomy in human memory theory, classifying agent memory into non-parametric and parametric forms and mapping them to episodic, semantic, and procedural categories. Du[18] introduce a three-dimensional taxonomy spanning temporal scope, representational substrate, and control policy, and examine five mechanism families in depth. These works share our motivation of moving beyond single-axis classification, but their dimensions differ substantially from ours. Wu et al.'s form dimension distinguishes parametric from non-parametric memory rather than structural data shapes. He et al. do not separately characterize storage substrates or indexing structures. Du's representational substrate dimension conflates physical storage media with representational forms, a conflation that our separation of Semantics Layer and Realization Layer is designed to address. None of these surveys analyzes governance policy at the operation level or provides an empirical cross-distribution of governance types across lifecycle operations.

2.3 Topically Scoped Surveys

A third group of surveys restricts its scope to a particular substrate type, application domain, or research challenge. Yang et al.[19] present a comprehensive survey of graph-based agent memory, proposing a taxonomy that includes short-term versus long-term memory, knowledge versus experience memory, and non-structural versus structural memory, with an implementation view focused on graph-based architectures. This survey provides the most thorough treatment of graph storage and graph traversal retrieval available in the literature, and our Semantics Layer analysis draws on its characterization of graph-based memory systems. However, its scope is restricted to graph substrates and does not cover the full range of storage substrate and indexing structure choices analyzed in our Realization Layer. The multi-agent memory survey of Zhang et al.[20] addresses the distinct challenges of memory coordination across agent teams, including synchronization, access control, and consistency, which our survey identifies as an open challenge but does not analyze in depth. The empirical survey of Jiang et al.[21] focuses specifically on evaluation methodology, documenting benchmark saturation and metric misalignment in current agentic memory evaluation, and provides the primary evidence base for our discussion of layer-specific evaluation as an open challenge.

2.4 Cognitive Architecture Foundations

CoALA[6] provides the cognitive architecture framework that underlies our memory-type axis in the Semantics Layer, distinguishing working, episodic, semantic, and procedural memory in the tradition of cognitive science taxonomies of human memory. CoALA is not a survey of external memory systems but a framework for characterizing the action and memory spaces of language agents; we inherit its four-way cognitive taxonomy as one axis of our Semantics Layer and extend it with a second axis derived from the empirical structural-form taxonomy of Zeng et al.[7], who systematically evaluate four memory structures, chunks, knowledge triples, atomic facts, and summaries, across a range of retrieval and reasoning tasks.

2.5 Positioning of This Survey

Our survey differs from the existing literature along three dimensions. First, the three-layer decomposition separates concerns that existing surveys conflate: logical representation from physical substrate, static indexing structure from dynamic retrieval operation, and system-level governance characterization from

operation-level governance analysis. Second, the two-axis representation matrix in Semantics Layer is the first to combine a cognitive-role axis and a structural-form axis in a single classification scheme, making explicit the many-to-many relationship between cognitive roles and structural forms and using the cross-product to surface gaps in the research landscape. Third, the empirical analysis is grounded in the systematic classification of 81 method papers across all three layers, including per-operation governance data, enabling quantitative findings about the distribution and co-occurrence of design choices that prior surveys, which are organized as narrative reviews of individual systems, do not provide.

Having established how existing surveys organize the agent memory design space and where their treatments leave specific analytical questions unresolved, we now develop the theoretical and empirical motivation for the three-layer decomposition that structures the remainder of this survey.

3 Motivation and Theoretical Grounding for the Three-Layer Decomposition

3.1 The Problem: Conflated Concerns in Existing Frameworks

The design of external memory systems for LLM-based agents involves at least three qualitatively distinct classes of decisions: decisions about what kind of information to store and in what form, decisions about how that information physically persisted and made accessible, and decisions about how stored information is operated upon and regulated over time. Each of these classes has its own vocabulary, its own set of trade-offs, and its own body of relevant prior work. Yet across existing surveys and taxonomies, these classes are routinely conflated, with the result that the design space is difficult to navigate and comparisons across systems are hard to make principled.

Three conflations in the literature are particularly consequential. The first concerns the relationship between indexing and retrieval. Du et al.[11] enumerate six memory operations, namely Consolidation, Updating, Indexing, Forgetting, Retrieval, and Compression, treating all six as operations of the same kind. However, indexing, which Du et al. define as efficiently organizing memory for retrieval, is the construction and maintenance of a static access structure over stored content, whereas retrieval is the dynamic execution of a query against that structure. The two differ in temporal character, computational cost, trigger condition, and optimization target. Indexing is performed at write time or during consolidation; retrieval is performed at read time in response to a query. Treating them as co-equal operations in a flat list obscures these differences and makes it impossible to reason independently about index design and query strategy.

The second conflation concerns the relationship between the logical form of a memory unit and its physical storage substrate. Du[18] propose a three-dimensional taxonomy of agent memory, one dimension of which is representational substrate. Under this dimension, they distinguish context-resident text, vector-indexed stores, structured stores such as SQL databases, key-value maps, and knowledge graphs, executable repositories, and hybrid stores. These examples are physical storage media, not representational forms. The question of whether a memory unit is encoded as an atomic fact, a summary, or a knowledge triple is logically prior to, and independent of, the question of whether it is stored in a vector database or a relational store. A single atomic fact can be embedded in a vector store, inserted into a key-value table, or added as a triple to a graph database. A single vector store can hold raw text chunks, summaries, or atomic fact embeddings. Conflating representational form with storage substrate prevents a principled analysis of this many-to-many relationship and obscures the independent contribution of each factor to system performance.

The third conflation is broader and has been acknowledged within the literature itself. Du et al. [11]observe that systematic perspectives unifying memory storage, retrieval, and memory-grounded generation into cohesive memory architectures are still in their early stages of development, citing He et al.[17] as evidence. Zhang et al.[10], whose write-manage-read framework represents the most widely cited operational taxonomy, cover only three coarse-grained stages and, as Du et al.[11] note, miss finer-grained operations. The consequence is that practitioners designing a new memory system lack a shared vocabulary for describing and comparing design decisions at the appropriate level of granularity.

3.2 The Principle: Separation of Concerns

The motivation for a layered decomposition that addresses these conflations is not merely empirical. It rests on a foundational principle of system design that has been stated with particular clarity by Dijkstra[22]: separation of concerns is the practice of organizing a system so that each part addresses a distinct and separate concern, with the recognition that nothing is gained, and something is actively lost, by tackling multiple aspects of a problem simultaneously. In Dijkstra's formulation, the separation is not a claim that concerns are independent in their effects, but rather a claim that they are best analyzed and designed in isolation, each on its own terms.

Applied to external memory systems, separation of concerns identifies the three conflations described in Section 2.1 as precisely the kinds of confusion that arise when distinct concerns are addressed without a principled decomposition. The logical form of a stored memory unit is one concern; the physical medium that holds it is a second, separable concern; and the operations that govern how memory evolves over time are a third. Each can, in principle, be varied independently of the others, as evidenced by the empirical observation of Zeng et al.[7] that the choice of structural form and the choice of retrieval method produce independently measurable effects on task performance across four memory structures and three retrieval methods. This independent measurability is precisely the hallmark of genuinely separable concerns.

3.3 The Precedents: Two Established Frameworks

Two theoretical frameworks from established disciplines independently support the specific three-way partition that the principle of separation of concerns motivates.

The first is the ANSI/SPARC three-schema architecture[4], the foundational model for database management systems, which decomposes any persistent data system into three levels: a conceptual level that specifies the logical structure of stored entities and their relationships, an internal level that specifies the physical storage structures and access paths, and an external level that specifies the views exposed to applications and users. The architecture's defining goal is data independence: logical data independence ensures that changes to the conceptual schema do not require changes to external views, while physical data independence ensures that changes to physical storage do not require changes to the conceptual schema[23]. Our three-layer decomposition instantiates an analogous form of independence. The Semantics Layer specifies what cognitive role a memory unit serves and what form it takes, independently of how it is stored. The Realization Layer specifies physical persistence and access path organization, independently of what content is stored or how operations are governed. The Dynamics Layer specifies lifecycle operations and governance policies, independently of the representational and storage choices made in the preceding layers.

The second framework is Marr's tri-level analysis[5], developed to characterize any information-processing system through three distinct levels of analysis. The computational level addresses the goal of the computation: what is being computed and why. The algorithmic and representational level addresses how the computation is carried out: what representations are used and what algorithm transforms them. The implementation level addresses how the algorithm is physically realized. Marr's central methodological claim is that understanding at one level does not substitute for understanding at another. Knowing how a system is physically implemented does not tell you what it computes and knowing what it computes does not tell you how the computation is organized algorithmically. This mutual irreducibility maps onto the three-layer decomposition in a direct way. Knowing that a system uses a vector store, a fact about the Realization Layer, does not determine what cognitive role or structural form its memory units take, which are facts about the Semantics Layer. Knowing that a system stores episodic summaries, a fact about the Semantics Layer, does not determine what decay function governs their removal, which is a fact about the Dynamics Layer.

Taken together, ANSI/SPARC and Marr's tri-level analysis provide cross-disciplinary grounding for the same architectural prescription: any persistent information-processing system should be understood and designed at separable levels corresponding to what is stored, how it is stored and accessed, and how it is operated upon over time.

3.4 The Decomposition and Its Benefits

Building on the above foundations, we organize the design space of external memory systems for LLM-based agents into three layers. The Semantics Layer characterizes memory units along two orthogonal axes: the cognitive role the unit serves, following the taxonomy of CoALA[6], and the structural form in which the content is encoded, following and extending the taxonomy of Zeng et al.[7]. The Realization Layer characterizes how memory units are physically persisted in one of four substrate types and how access paths are organized through one of five indexing structures, with the two sub-dimensions treated as independently variable. The Dynamics Layer characterizes the six lifecycle operations through which memory is written, updated, forgotten, retrieved, assembled, and consolidated, together with the four classes of governance policy through which those operations are triggered and regulated.

This decomposition yields three concrete benefits over existing frameworks.

The first benefit is design-space navigation. Each layer exposes an independent set of design decisions that a practitioner can evaluate against layer-specific criteria. The choice of structural form at Semantics Layer can be evaluated against criteria of expressivity, encoding cost, and downstream reasoning precision, drawing on the empirical findings of Zeng et al.[7]. The choice of storage substrate and indexing structure at Realization Layer can be evaluated against criteria of retrieval accuracy, consistency, and scalability. The choice of operation and governance policy at Dynamics Layer can be evaluated against criteria of latency, semantic fidelity, and memory growth management. Because the layers are decoupled in the sense established by ANSI/SPARC, a decision at one layer does not determine decisions at the others, and practitioners can substitute one layer's components without redesigning the system as a whole.

The second benefit is gap identification. The cross-product representation enabled by the two-axis structure of Semantics Layer reveals systematic patterns of coverage and absence in the literature. In particular, the procedural memory row of the Semantics Layer matrix exhibits a markedly different structural form distribution from the other three cognitive roles. Among the 10 surveyed systems whose primary memory type is procedural, six store procedural content as natural-language summaries of plans or strategies (e.g., ExpeL[24], ReasoningBank[25], Agentic Plan Caching[26]), two store it as executable code (Voyager[27], Skill-Pro[28]), and the remaining systems combine summaries with chunks or other declarative forms. No surveyed system instantiates procedural memory as atomic facts or knowledge triples. This pattern is not incidental: it reflects a structural asymmetry between knowledge-oriented and skill-oriented memory. Procedural knowledge concerns how to act, and its functional properties of executability and compositional reuse are difficult to preserve under decomposition into minimal propositions (atomic facts) or subject-predicate-object relations (triples). The two viable representations are coarse-grained natural-language descriptions of procedures, which preserve semantic context at the cost of executability, and executable code, which preserves executability at the cost of natural-language flexibility. Single-axis classifications, which collapse either the cognitive role or the structural form, cannot surface this asymmetry between knowledge representation and skill representation. This finding constitutes an extension of Zeng et al.'s four-form taxonomy[7] and reveals a gap in the coverage of prior structural analyses.

The third benefit is layer-specific evaluation. The decomposition provides a conceptual basis for evaluation at the appropriate level of granularity. As Jiang et al.[21] document, existing evaluations of agentic memory systems suffer from benchmark saturation, metric misalignment, and failure to isolate the contributions of individual system components. The three-layer framework provides a basis for designing evaluations that hold two layers constant while varying the third, enabling component-level attribution of performance differences that is currently absent from the literature.

3.5 The Decomposition Mechanism and Layer Relationships

3.5.1 The Mechanism: Three Separable Concerns

The three-layer decomposition is not a stipulative partition imposed for organizational convenience. It is grounded in three qualitatively distinct classes of design decisions that any external memory system must make, distinguished along multiple dimensions that warrant their separation.

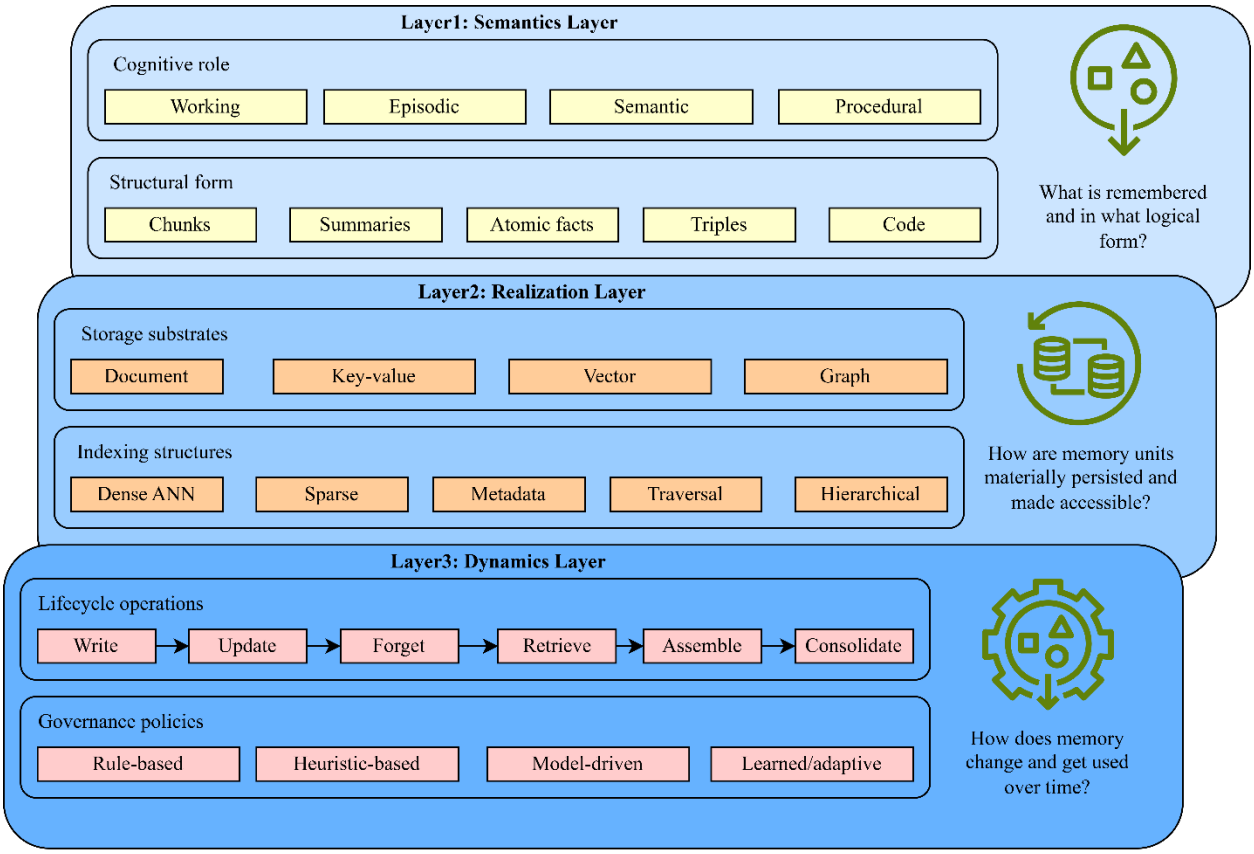

Figure 1: Semantics-Realization-Dynamics decomposition of external long-term memory systems. The Memory Semantics Layer defines memory roles and structural forms; the Memory Realization Layer specifies storage substrates and indexing structures; and the Memory Dynamics Layer characterizes lifecycle operations and governance policies.

The first class concerns the logical content of memory: what units of information the system stores and in what representational form. The second class concerns physical realization: how those units are persisted in a storage substrate and how access paths are organized to support retrieval. The third class concerns dynamic behavior: how stored memory is operated upon over the system's lifecycle and how those operations are regulated by governance policies. These three classes differ along three dimensions that justify treating them as distinct concerns.

They differ in temporal character. Logical content decisions are static commitments made at design time and rarely revised. Physical realization decisions are semi-static, fixed at write time when memory units are first inserted into the substrate. Dynamic behavior decisions are made at runtime, triggered repeatedly throughout the system's operational lifetime.

They differ in optimization target. Logical content decisions optimize for expressive adequacy, the ability of the chosen form to represent the information of interest at the appropriate level of granularity. Physical realization decisions optimize for retrieval efficiency, the ability of the chosen substrate and indexing structure to return relevant memory units within acceptable latency. Dynamic behavior decisions optimize for lifecycle quality, the ability of the chosen operations and policies to maintain memory coherence and utility over extended interaction horizons.

They differ in evaluation criteria. The quality of a logical content decision is assessed against criteria such as expressiveness, encoding cost, and downstream reasoning precision, drawing on the empirical findings of Zeng et al. The quality of a physical realization decision is assessed against criteria such as retrieval accuracy,

consistency, and scalability. The quality of a dynamic behavior decision is assessed against criteria such as memory growth control, conflict resolution effectiveness, and adaptation to changing task distributions.

This division is consistent with two well-established cross-disciplinary precedents. The ANSI/SPARC three-schema architecture decomposes any persistent data system into conceptual, internal, and external levels, with the explicit goal of allowing each level to evolve independently[4]. Marr's tri-level analysis decomposes any information-processing system into representation, implementation, and computational control, with the methodological claim that understanding at one level does not substitute for understanding at another. Both frameworks converge on the same architectural prescription: separating logical specification from physical implementation from operational control[5].

3.5.2 The Relationships Among the Three Layers

The three layers are independent in the sense of non-determination, not causal isolation. A choice at one layer constrains but does not uniquely determine choices at another layer. The same semantic memory form can be realized through multiple storage/access configurations; the same realization can support different lifecycle policies; and the same runtime operation can be governed by different decision regimes. Orthogonality therefore means that each layer exposes a distinct design variable whose effects can be studied while holding the others fixed.

The three layers stand in three distinct relationships to one another: logical independence, unidirectional dependency, and cross-layer trade-off.

Logical independence. Each layer can vary without requiring redefinition of the others. The same logical representation at Semantics Layer can be realized through different combinations of substrate and indexing structure at Realization Layer: an atomic fact about user preference can be embedded in a vector store with dense ANN indexing, inserted into a key-value table with exact-match lookup, or added as a triple to a graph database with traversal-based retrieval. The same physical realization at Realization Layer can be regulated by different governance policies at Dynamics Layer: a vector store with dense ANN indexing can be paired with heuristic-based forgetting, model-driven consolidation, or learned write admission. This independence is the formal property that ANSI/SPARC's data independence principle preserves, and it is empirically supported by Zeng et al., who demonstrate that structural form choices and retrieval method choices produce independently measurable effects on task performance.

Unidirectional dependency. Although the three layers are logically independent in their definitions, they exhibit a unidirectional dependency in their constructive order: Realization Layer decisions presuppose Semantics Layer decisions, and Dynamics Layer decisions presuppose Realization Layer decisions, but not the reverse. A storage substrate and indexing structure cannot be selected without first knowing what is being stored: graph traversal is meaningful only if the memory is organized as nodes and edges; dense ANN indexing requires a vector representation as input. Similarly, lifecycle operations and governance policies cannot be designed without first knowing the physical realization: a conflict resolution policy for a graph substrate must operate on edge integrity, while the same conflict for a vector substrate requires similarity-based detection. The reverse implications do not hold. The decision to store user preferences as atomic facts at Semantics Layer does not determine whether they reside in a vector store or a graph store, nor whether the system applies heuristic decay or model-driven retention. This asymmetry is precisely what makes the layered decomposition useful for design: each layer can be reasoned about in terms of the layers above it without committing to choices in the layers below.

Cross-layer trade-offs. Independence and dependency do not preclude interaction. In practice, choices at one layer impose constraints on the feasible or efficient choices at adjacent layers. Three cross-layer trade-offs are visible in the surveyed literature. Between Semantics Layer and Realization Layer, the choice of structural form interacts with the choice of indexing structure: atomic facts and triples favor structured indexing primitives (metadata filtering, graph traversal), while chunks and summaries favor dense ANN, as the empirical distribution in Section 5 demonstrates. Between Realization Layer and Dynamics Layer, the choice of storage substrate interacts with the complexity of governance required: graph substrates impose stronger consistency requirements on update and forget operations, while vector substrates defer consistency to the operation layer or leave it unaddressed. Between Semantics Layer and Dynamics Layer, the choice

of cognitive role and structural form interacts with the appropriate operations and policies: procedural memory in code form requires different update and consolidation strategies than declarative memory in textual form, because code admits compositional reuse but not the linguistic merging operations available to natural-language summaries.

These trade-offs are not evidence against the layered decomposition; they are evidence that the decomposition correctly isolates the design dimensions along which choices must be coordinated. A framework that conflated these layers would make the trade-offs invisible by collapsing the dimensions along which they vary. The three-layer decomposition makes the trade-offs explicit, providing a vocabulary for describing where the design constraints of one layer propagate to the next and where they do not.

With the theoretical grounding established, we turn to the first layer of the decomposition. The Semantics Layer addresses the most fundamental design question: what units of information are stored, and in what logical form are they encoded.

4 Semantics Layer

4.1 Why a Two-Axis Decomposition

The representation layer specifies what units of memory are stored and how they are encoded from raw input. Existing surveys have approached this question along a single classification axis. CoALA organizes memory by cognitive function, distinguishing working memory, episodic memory, semantic memory, and procedural memory in the tradition of Tulving and the SOAR cognitive architecture[6]. In contrast, Zeng et al. organize memory by structural form, evaluating chunks, knowledge triples, atomic facts, and summaries as alternative shapes that the same information can take, and report that each shape affords distinct trade-offs between recall precision, contextual unity, and reasoning exactness[7].

Both axes have explanatory power, yet neither is sufficient on its own. Cognitive-role classification answers what functional purpose a memory unit serves but is silent about how it is materially encoded. A piece of episodic memory may be stored as a verbatim observation log, as a post-hoc summary, or as nodes in a temporal graph, and these implementation choices have first-order consequences for retrieval cost, conflict resolution, and downstream reasoning. Conversely, structural-form classification specifies the data shape but is silent about the functional role: a triple may encode a piece of semantic world knowledge, a user preference, or an episodic fact about a past interaction, and these distinctions matter for write policy, decay schedule, and access control. Bridging these two perspectives requires a representation that is simultaneously addressable along both axes.

Recent surveys have begun to move beyond single-axis classification. Wu et al. propose a three-dimensional, eight-quadrant taxonomy along the dimensions of object (personal versus system), form (parametric versus non-parametric), and time (short-term versus long-term)[16]. He et al. introduce a taxonomy of AI long-term memory grounded in human memory theory, classifying agent memory into non-parametric and parametric forms and mapping them to episodic, semantic, and procedural categories[17]. While these efforts share our motivation of moving beyond a single classification axis, none of them combines the cognitive-role axis (in the CoALA tradition) with the structural-form axis (in the sense of Zeng et al.) that we adopt as our two dimensions. The combination is non-trivial because the two axes have distinct conceptual provenances: cognitive role is grounded in cognitive-science taxonomies of human memory, whereas structural form is grounded in empirical comparisons of data shapes for retrieval-augmented systems. Bringing them together as a single representation layer makes explicit a relationship that is otherwise left implicit, namely that the design of any memory unit involves a joint choice along both axes.

4.2 The Two Dimensions

We organize the representation layer along two orthogonal dimensions.

Cognitive role. Following CoALA[6], we distinguish four functional roles that parallel classical cognitive-science taxonomies of human memory. Working memory holds the active context required for the current

decision cycle and is bounded by the model's context window; it provides immediate accessibility at the cost of strict capacity limits. Episodic memory records time-stamped events and interaction trajectories, preserving the temporal and contextual specificity of individual experiences. Semantic memory stores abstracted, decontextualized factual or conceptual knowledge, including user profile information that has been distilled from interaction history into stable attributes. Procedural memory stores reusable skills, plans, or strategies that can be invoked to guide future action without re-deriving the underlying reasoning from scratch.

Structural form. Following Zeng et al. theory[7], we evaluate four data shapes under which memory content can be encoded: chunks, which preserve verbatim or minimally processed text at session, turn, or sentence granularity; summaries, which compress raw content into shorter natural-language passages; atomic facts, which are minimal self-contained propositions extracted from input; and knowledge triples, which encode subject-predicate-object structured relations. Zeng et al. establish empirically that these forms are not interchangeable: chunks and summaries excel in tasks requiring extensive contextual fidelity, while knowledge triples and atomic facts are particularly effective for relational reasoning and precision in multi-hop and single-hop question answering[7].

We extend Zeng et al.'s four-form taxonomy with a fifth structural form: code. Code is defined as executable programs or scripts that encode procedural knowledge as runnable artifacts rather than as declarative text. This extension is motivated by a category of procedural memory systems whose content is functionally rather than linguistically encoded. Voyager[27] stores an ever-growing skill library of executable JavaScript code for storing and retrieving complex behaviors, explicitly choosing code as the action space because programs can naturally represent temporally extended and compositional actions that no declarative form can provide. Skill-Pro[28] formalizes skills as Skill-MDPs with activation, execution, and termination conditions, again in an executable rather than declarative form. Code is therefore not merely a syntactic variant of existing forms but a structurally distinct category characterized by executability, compositionality, and the absence of the semantic ambiguity that characterizes natural language forms. This extension constitutes a contribution beyond the original Zeng et al. taxonomy, which was developed in the context of QA and dialogue tasks and does not account for procedural memory systems.

The two dimensions are orthogonal: any concrete memory unit can be characterized as a (role, form) pair, and moving along one axis while holding the other fixed yields a different system design with different retrieval, update, and governance properties.

4.3 Cross-Product Instantiations

Table 1 populates the 4 x 5 product space with representative systems from the surveyed literature. Each cell, where populated, names a system whose primary memory representation falls into the corresponding (role, form) pair.

Table 1: The 4 x 5 representation matrix with representative systems

	Chunks	Summaries	Atomic facts	Triples	Code
Working	MemoBrain[29]	ReadAgent[30]	-	GraphReader[31]	-
Episodic	Wang et al.[32]	MemoChat[33]	A-MEM[34]	D-SMART[35]	-
Semantic	RAG[3]	RECOMP[36]	Inside Out[37]	EchoKG[38]	-
Procedural	Agent KB[39]	APC[26]	-	-	Voyager[27]

Brief justifications for each populated cell:

Working x Chunks: MemoBrain[29] maintains an executive memory that holds verbatim active context for the current reasoning cycle, organized as working memory chunks for immediate access during complex tasks.

Working x Summaries: ReadAgent[30] compresses long documents into gist memories, which are natural-language summaries that fit within the active context window for the current decision cycle.

Working x Triples: GraphReader[31] builds a graph-based agent where working context is represented as structured nodes and relations, using graph traversal over triples to enhance the model's ability to reason over long contexts within the current decision cycle.

Episodic x Chunks: Wang et al.[32] store raw time-stamped interaction records as verbatim text chunks, representing the foundational design for episodic memory in retrieval-augmented agents.

Episodic x Summaries: MemoChat[33] tunes LLMs to use memos for consistent long-range dialogue, storing episodic interaction history as compressed summary memos.

Episodic x Atomic facts: A-MEM[34] applies the Zettelkasten principle to episodic memory, storing each memory as an atomic note capturing a single self-contained unit of experience.

Episodic x Triples: D-SMART[35] enhances dialogue consistency by encoding episodic interaction events as structured triples in a dynamic memory graph.

Semantic x Chunks: RAG[3] retrieves relevant knowledge chunks from an external document corpus, treating the corpus as a semantic memory store of verbatim text passages.

Semantic x Summaries: RECOMP[36] improves retrieval-augmented generation by compressing retrieved documents into concise summaries that distill the semantic content relevant to a query.

Semantic x Atomic facts: Inside Out[37] maintains a user-centric core memory tree where stable semantic attributes about the user are stored as atomic facts that evolve across sessions.

Semantic x Triples: EchoKG[38] builds a dynamic user-preference knowledge graph that stores semantic user knowledge as subject-predicate-object triples updated from interaction history.

Procedural x Chunks: Agent KB[39] stores reusable procedural knowledge including tool-use strategies and task-solving recipes as verbatim text chunks retrieved to guide future agent actions.

Procedural x Summaries: Agentic Plan Caching[26] caches compressed natural-language plan summaries from successful task executions as reusable procedural memory for future similar tasks.

Procedural x Code: Voyager[27] stores an ever-growing skill library of executable JavaScript code, enabling compositional reuse and direct execution of acquired skills in novel task contexts.

4.4 What the Empty Cells Reveal

Of the 20 cells in the product space, 14 are populated by at least one system among the 81 surveyed; 6 remain empty. These empty cells are not incidental features of the classification scheme. They encode three substantive observations about the structure of the current research landscape.

First, working memory resists structured forms beyond triples. Working memory appears in chunks, summaries, and, with medium confidence, triples (GraphReader[31]), but no surveyed system employs atomic facts as a primary working memory representation. The absence of atomic facts in the working memory row reflects a genuine architectural constraint: working memory must be cheap to read and write at every decision step, and the extraction overhead required to produce atomic facts from raw observations is incompatible with this requirement. Triples, as in GraphReader, represent an exploratory design point at the boundary of this constraint.

Second, procedural memory is largely absent from structured declarative forms and is more often represented as natural-language summaries than executable code. No surveyed system stores procedural memory as atomic facts, triples, or code within the coverage of the surveyed literature excluding Voyager[27] and Skill-Pro[28]. This pattern reflects two distinct reasons. For atomic facts and triples, the absence is a genuine structural constraint: procedural knowledge is fundamentally about how to act, and its functional properties of executability and compositionality cannot be preserved under decomposition into minimal propositions or subject-predicate-object relations. For code, the absence among the surveyed systems reflects a literature gap rather than an architectural impossibility: Voyager[27] demonstrates that code is a viable and powerful form for procedural memory, but code-based procedural memory systems are underrepresented in the surveyed literature relative to declarative procedural memory systems, which predominantly use summaries.

Third, procedural memory is dominated by summaries rather than code, contrary to the expectation set by the cognitive science literature. Among the ten systems whose primary memory type is procedural, six store procedural memory as summaries (plans, strategies, compressed task solutions), one as a summary–chunk combination, and one as raw chunks, while Voyager and Skill-Pro are the only two storing procedural memory as code. This distribution reveals a practical divergence from the theoretical characterization of procedural memory as inherently executable: the majority of systems that claim to implement procedural memory in fact store declarative descriptions of procedures rather than executable procedures themselves. This distinction matters for the evaluation of such systems, because a declarative description of a plan and an executable implementation of a plan impose different requirements on the retrieval and execution pipeline.

4.5 Benefits of the Two-Axis View

The two-axis decomposition yields three concrete benefits over single-axis classifications.

Dissolving category-boundary ambiguity. A unit such as a stored user preference can be simultaneously described as an atomic fact, a piece of long-term semantic memory, and a stable user attribute: none of these labels is wrong, yet none uniquely identifies the unit under a single-axis taxonomy. The two-axis view dissolves this ambiguity by assigning the unit to a single (Semantic, Atomic facts) cell, simultaneously recognizing its functional role and its structural shape. This specificity matters not only for classification but for system design, because the appropriate write policy, decay schedule, and access control for a (Semantic, Atomic facts) unit differ from those for a (Episodic, Chunks) unit even if both might be described as long-term memory under a cognitive-role-only taxonomy.

Exposing design trade-offs invisible under single-axis views. The same cognitive role can be instantiated in markedly different structural forms with markedly different downstream properties. Episodic memory as raw chunks (Wang et al.[32]) supports faithful temporal recall but accumulates linearly with interaction length, imposing growing retrieval latency and storage cost over time. Episodic memory as summaries (MemoChat[33]) controls storage growth at the cost of information loss during compression. Episodic memory as triples (D-SMART[35]) supports relational multi-hop queries at the cost of an expensive triple-extraction write path. These trade-offs are first-class design concerns that are invisible when cognitive role alone is used as the organizational axis, because all three systems would be classified identically as episodic memory systems.

Providing a structured tool for gap identification. The empty cells in Table 1 encode either genuine architectural constraints (atomic facts in working memory) or genuine gaps in the research literature (code in episodic and semantic memory, structured forms in procedural memory). Single-axis classifications cannot surface these gaps because they collapse one of the two relevant dimensions. The matrix makes explicit that the research community has almost entirely avoided code-form memory outside the procedural role and has almost entirely avoided structured forms inside the procedural role: two asymmetries that point toward concrete and distinct directions for future work.

Having characterized the logical content of memory units along the cognitive-role and structural-form axes, we turn to the second layer, which addresses how those units are physically persisted and how access paths over them are organized. The Realization Layer is logically downstream of the representation layer in the sense that substrate and indexing structure choices presuppose a decision about what is being stored, but are independent of it in the sense that the same representation can be served by different substrates and indexing structures without altering its logical definition.

5 Realization Layer

5.1 Why Two Sub-dimensions

The Realization Layer comprises two analytically distinct but partially coupled sub-dimensions: storage substrates, which specify where memory units physically persisted, and indexing structures, which specify the semi-static access paths through which those units are made available to runtime retrieval. Both belong to the physical implementation tier in the sense of ANSI/SPARC's internal level, but they vary independently within

that tier—the same substrate can support multiple indexing structures, and the same indexing structure can be built over different substrates, as we document empirically in Section 5.4. These two questions correspond to two well-established axes of analysis in prior literature, and we treat them as independent sub-dimensions within a single layer for the same reason they are separated in database systems research: the choice of substrate constrains but does not determine the choice of indexing structure, and the choice of indexing structure constrains but does not determine the substrate.

Theoretical grounding for storage substrates. The classification of persistent data stores by data model has a long history in database research. Cattell[8], in a widely cited survey of scalable data stores, identifies three principal non-relational data model categories: key-value stores, document stores, and extensible record stores (also known as column-family stores). We adopt the first two categories directly and adapt the remaining structure in two ways motivated by the specific requirements of LLM-based agent memory systems. First, we replace extensible record stores with graph stores, since extensible record stores do not appear among the surveyed systems whereas graph stores are a major and rapidly growing substrate type in the agent memory literature; Cattell himself notes that graph databases were excluded from his survey only due to space considerations, not because they fall outside the data-model classification scheme. Second, we add vector stores as a fourth substrate type, reflecting the emergence of embedding-based persistence as the dominant storage mechanism in the agent memory literature, one that did not exist as a distinct category at the time of Cattell's survey. The use of multiple substrates within a single system, observed in 20 of the 81 surveyed systems, is treated as a composition of these four base types rather than as a separate category, on the grounds that "hybrid" describes how substrates are combined rather than what substrate type is used.

Theoretical grounding for indexing structures. The classification of retrieval mechanisms has a parallel body of theoretical work in information retrieval. Lin et al.[9] survey text ranking techniques and establish the fundamental distinction between sparse retrieval, which operates via inverted indexes and lexical matching methods such as BM25, and dense retrieval, which maps queries and documents into continuous vector spaces and performs approximate nearest-neighbor search. This sparse-dense distinction is the theoretical anchor for two of our five indexing structure types. The remaining three, graph traversal, metadata filtering, and hierarchical indexing, are motivated by empirical findings in the agent memory literature itself: graph traversal addresses the relational retrieval failures of embedding-based methods identified by HippoRAG[40] and SYNAPSE[41]; metadata filtering addresses the need for structural constraint propagation that neither sparse nor dense methods provide natively; and hierarchical indexing addresses the scalability problem of searching over large accumulated memory stores.

Indexing structures as composable primitives. A critical distinction between the two sub-dimensions is that storage substrates are mutually exclusive, whereas indexing structures are composable primitives rather than mutually exclusive categories. Among the 81 method papers surveyed, the majority of systems deploy two or more indexing structures in combination: 14 pair dense ANN with metadata filtering, 10 pair dense ANN with graph traversal, and 9 pair dense ANN with hierarchical structures. Dense ANN functions as the near-universal baseline, augmented by one or more complementary structures depending on the retrieval requirements of the system. This composability distinguishes the indexing sub-dimension from the storage substrate sub-dimension and means that a given system's indexing configuration is best characterized as a combination of primitives rather than assignment to a single category.

5.2 Sub-dimension A: Storage Substrates

Vector stores are the dominant substrate, used by 51 of the 81 method papers. A vector store persists memory units as dense embedding vectors alongside the original text, enabling semantic similarity retrieval without requiring structured content. Systems including A-MEM[34], MemoryBank[42], and Mem0[43] rely on vector stores as their primary persistence substrate. The principal theoretical limitation of this substrate has been formally characterized: Weller et al.[15] prove that the number of top-k document subsets returnable for any query is bounded by the embedding dimension, establishing a geometric limit on what embedding-based retrieval can represent regardless of model scale. This limitation is particularly consequential for queries requiring exact temporal match, relational traversal, or reasoning over entities that are semantically dissimilar at surface level.

Document and textual repositories are used by 27 of the 81 method papers, storing memory units as text in flat files, relational databases, or document stores without imposing structural constraints on the content. Document stores offer maximal fidelity to the original content and impose no structural overhead at write time. Systems including ReadAgent[30] and standard RAG architectures[3] store externally retrieved or accumulated text as document collections, relying on an indexing layer built separately over the text to provide semantic access. The primary limitation of document stores is the absence of native support for structured or relational queries.

Graph stores are used by 19 of the 81 method papers, persisting memory units as nodes and edges with typed relationships between them. Systems including Zep[44], D-SMART[35], EchoKG[38], and MAGMA[45] store memory in graph substrates. Graph stores natively support multi-hop traversal and relational queries that vector stores cannot support through similarity alone. SYNAPSE explicitly motivates its graph-based substrate by the failure of standard embedding-based retrieval to recover linked memories absent surface-level similarity between the query and the relevant memory entries[41]. The principal costs of graph stores are the higher write-path overhead of graph construction and the complexity of maintaining consistency as memory grows and existing entries are updated or contradicted.

Key-value stores are used by 8 of the 81 method papers, persisting memory units as discrete entries addressable by a structured key. KV stores are best suited to memory units with stable identifiers, such as cached plans, skill entries, or session-level summaries. Systems including Agentic Plan Caching[26] and Chain-of-Memory[46] use KV stores for procedural or episodic content with well-defined identifiers.

Hybrid substrates are used by 20 of the 81 method papers. A hybrid substrate combines two or more of the above backends within a single system, routing different content types or query types to different backends. AriGraph[47] integrates episodic and semantic content in a combined graph-and-vector substrate. MemGPT[1] separates in-context working memory from external archival storage as two distinct persistence tiers. EverMemOS[48] organizes memory across document, graph, and working-context tiers. Hybrid systems accept added architectural complexity in exchange for broader retrieval coverage, a trade-off reflected in the data: hybrid-substrate systems account for the most diverse indexing structure combinations in the surveyed literature.

5.3 Sub-dimension B: Indexing Structures

Dense vector indexing appears in 67 of the 81 method papers as a primary or component index and functions as the near-universal baseline in the surveyed literature. Dense indexing represents both memory units and queries as continuous embedding vectors and retrieves by approximate nearest-neighbor search using algorithms including FAISS, HNSW, and ScaNN. It is effective for topical and semantic similarity retrieval but subject to the geometric retrieval bound established by Weller et al.[15], which motivates the complementary structures described below.

Metadata filtering appears as a component in 21 systems and restricts the retrieval candidate set using structured attributes attached to memory units, such as timestamps, session identifiers, topic labels, user identifiers, or confidence scores. Metadata filtering does not rank candidates by content similarity but eliminates candidates that cannot be relevant given the query's structural constraints. It appears almost exclusively in combination with dense ANN rather than as a standalone structure, confirming its role as a pre-filter that narrows the candidate set before similarity ranking. SwiftMem[49] exemplifies this pattern, combining metadata-based temporal scoping with semantic content retrieval to achieve sub-linear latency over accumulated interaction records.

Graph traversal appears in 19 systems and is the defining indexing structure for graph-substrate systems. Graph traversal retrieves memory units by following typed edges from seed nodes identified by an initial query, enabling multi-hop retrieval of related units not directly similar to the query at surface level. HippoRAG[40] runs Personalized PageRank on its knowledge graph using query concepts as seeds, achieving single-step multi-hop retrieval without iterative querying. In the surveyed data, graph traversal appears exclusively with graph or hybrid substrates, never with vector or document stores alone, which is expected given its structural dependency on an explicit edge representation. This makes the choice between graph and non-graph substrate effectively a choice about whether graph traversal is available as a retrieval modality.

Hierarchical indexing appears in 11 systems and organizes memory units into multi-level structures, routing queries to the appropriate level based on recency, abstraction, or importance. ReadAgent[30] stores gist memories at different levels of compression. GraphRAG[50] pre-computes hierarchical community summaries at multiple levels of abstraction, allowing queries to be resolved at coarse or fine granularity depending on their scope. Hierarchical indexing appears primarily in document and hybrid substrates, reflecting its suitability for systems that accumulate large volumes of content over extended time horizons and manage retrieval cost through selective access to the appropriate level of the hierarchy.

Sparse and keyword indexing appears in only 1 of the 81 method papers as a component index, paired with KV storage in Agentic Plan Caching[26]. This near-absence is notable given that the theoretical distinction between sparse and dense retrieval is the most fundamental partition in the information retrieval literature[9], and given that sparse methods such as BM25 have been shown to outperform embedding-based methods on queries involving exact match, rare entities, and out-of-distribution vocabulary[51]. A benchmark for procedural memory retrieval[51] finds that BM25 outperforms embedding-based methods on queries involving novel object instantiations precisely because it does not depend on learned embedding similarity. The rarity of sparse indexing in agent memory systems constitutes a gap in the literature that the field has yet to address systematically.

5.4 Observed Patterns and Their Implications

Analysis of the 81 method papers reveals three structural patterns in the joint distribution of substrate and indexing structure choices.

Pattern one: Dense ANN as universal baseline. Dense ANN indexing appears across all four substrate types and in combination with every other indexing structure. Of the 67 systems using dense indexing, 14 pair it with metadata filtering, 10 pair it with graph traversal, and 9 pair it with hierarchical structures. This universality reflects both the practical maturity of ANN libraries and the semantic flexibility of embedding-based retrieval, but it also reflects a field-wide tendency to default to dense indexing rather than systematically evaluating alternatives. The near-complete absence of sparse indexing from the surveyed systems suggests that hybrid dense-sparse configurations, well established in the general information retrieval literature, have not yet been systematically explored in the agent memory context.

Pattern two: Graph traversal is substrate-constrained. Graph traversal appears exclusively with graph or hybrid substrates, confirming its structural dependency on an explicit edge representation. Of the 9 systems that use a graph store as their sole substrate, all use graph traversal as at least one component of their indexing structure; and all 19 systems whose substrate set includes a graph store likewise employ graph traversal. This is the strongest substrate-index coupling in the surveyed data. The implication is that the choice between graph and non-graph substrate is effectively a choice about whether relational multi-hop retrieval is available as a modality, making substrate selection a higher-order decision than indexing structure selection.

Pattern three: Sparse indexing is systematically underexplored. The near-absence of sparse keyword indexing from agent memory systems, despite its well-documented complementarity with dense indexing in the general retrieval literature, represents a gap in the design space that the field has not addressed. Given the geometric limitations of embedding-based retrieval established by Weller et al.[15], and the documented superiority of sparse methods on exact-match and out-of-distribution queries, this gap points toward a concrete direction for future work: systematic evaluation of hybrid dense-sparse indexing configurations for agent memory systems.

5.5 Design Trade-offs at the Realization Layer

The substrate and indexing choices at Realization Layer impose trade-offs that interact with the representation choices made at Semantics Layer and the operation and governance choices at Dynamics Layer.

Retrieval accuracy versus write-path cost. Graph substrates with graph traversal indexing provide the highest retrieval accuracy on relational and multi-hop queries, as demonstrated by HippoRAG's improvements of approximately 20 points over standard RAG on multi-hop QA benchmarks[40]. However, graph construction at write time requires triple extraction, entity resolution, and edge insertion, each of which is computationally

expensive. Vector substrates with dense ANN indexing provide near-constant-time write operations at the cost of the geometric retrieval limitations identified by Weller et al.[15]. This trade-off is partly determined by the ratio of write to read operations in the target deployment: interactive dialogue systems with high write frequency favor vector substrates, while knowledge-intensive reasoning systems with lower write frequency and high multi-hop query density favor graph substrates.

Consistency maintenance versus retrieval flexibility. Graph substrates must maintain referential integrity across nodes and edges as memory is updated and new information contradicts existing entries, placing a burden on the governance mechanisms at Dynamics Layer. Systems using graph substrates must implement conflict detection and resolution at the graph level, as Zep[44] and EchoKG[38] do explicitly. Vector substrates do not natively enforce consistency, deferring this problem to the operation layer or leaving it unaddressed. This asymmetry means that the substrate choice at Realization Layer directly determines the complexity of the governance policies required at Dynamics Layer.

Scalability versus reasoning support. Dense ANN indexing scales sub-linearly with corpus size for fixed recall targets, making it suitable for large memory stores. Graph traversal scales with path length and graph density, which can grow in highly connected memory graphs. Hierarchical indexing controls retrieval cost by routing queries to the appropriate level but requires additional write-time overhead to build and maintain the hierarchy. These scalability properties determine the practical ceiling on memory size before retrieval latency becomes operationally unacceptable, a constraint that evaluation benchmarks have only recently begun to measure systematically[21].

Having characterized the static structure of memory systems at the representation and storage levels, we turn to the dynamic dimension: how memory evolves over time through a vocabulary of lifecycle operations, and how those operations are regulated by governance policies. The Dynamics Layer is the layer at which the design decisions made in Layers 1 and 2 are actualized in running systems, and it is where the most consequential gaps in the current literature are concentrated.

6 Dynamics Layer

6.1 Why Two Sub-dimensions and Their Theoretical Grounding

The Dynamics Layer addresses two questions that together define how memory evolves over time: what operations are performed on stored memory units, and what policies govern when and how those operations are triggered. These two questions are analytically distinct and are best treated as independent sub-dimensions. The lifecycle operations sub-dimension describes the functional vocabulary of memory manipulation, the set of things a memory system can do. The governance policies sub-dimension describes the decision-making regime under which those operations are invoked, the set of ways a memory system decides what to do and when.

Theoretical grounding for lifecycle operations. The idea that memory systems can be characterized by a set of atomic operations has precedent in both the agent memory literature and in cognitive science. Zhang et al.[10] organize the memory mechanism of LLM-based agents into three coarse stages, writing, management, and reading, acknowledging that finer-grained operations such as indexing are not captured by this partition. Du et al.[11] refine this into six operations: Consolidation, Updating, Indexing, Forgetting, Retrieval, and Compression, with Indexing and Compression distinguished from the others on the grounds that they concern the organization of memory rather than its semantic content. We depart from Du et al. in one respect: we exclude Indexing from this layer on the grounds established in Section 3, namely that Indexing is the construction of a static access structure over stored content and belongs to the Realization Layer rather than the Dynamics Layer. We replace Compression with Assemble, defined as the composition of retrieved memory content into a context suitable for LLM input. This reflects a common architectural pattern in the surveyed systems, the assembly of retrieved items into a coherent context window, that Du et al.'s taxonomy does not explicitly capture. The resulting six-operation vocabulary, Write, Update, Forget, Retrieve, Assemble, and Consolidate, provides complete coverage of the operations observed in the surveyed literature: no operation reported in any of the 81 method papers falls outside these six categories.

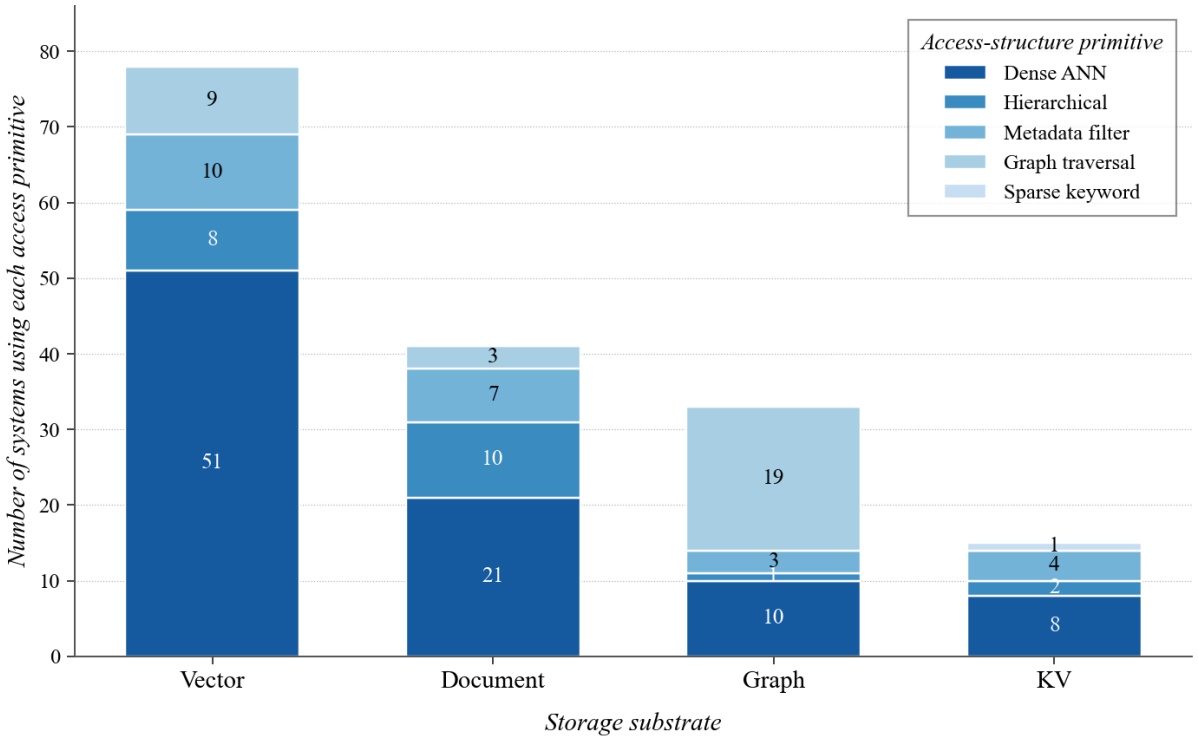

Figure 2: Distribution of indexing primitives across storage substrates in the 81 surveyed systems. Both substrates and indexing primitives are multi-select per system; 20 systems use multiple substrates.

Theoretical grounding for governance policies. The classification of governance policies has a parallel body of work motivating our four-way partition. Sun et al.[12] explicitly frame the problem: memory management is still largely driven by hand-designed heuristics, offering little insight into the long-term and uncertain consequences of memory decisions, and argue that it should be viewed as a sequential decision-making problem under uncertainty. A-MAC[14] similarly distinguishes between heuristic-based methods that rely on hand-crafted scoring functions combining factors such as recency, relevance, and importance, and LLM-native approaches that delegate memory admission entirely to language models. The SSGM framework[13] identifies rule-based governance, heuristic decay modeling, and adaptive self-refining memory systems as three distinct paradigms in the literature. These three sources independently converge on the same partition: static rules, hand-crafted heuristics, model-driven judgment, and learned optimization are qualitatively distinct decision regimes that require different infrastructure, offer different guarantees, and scale differently with memory size and interaction length.

The governance-operation relationship. Governance policies do not form an independent axis that intersects lifecycle operations in a clean matrix. Rather, governance policies are the decision mechanisms that determine when and how lifecycle operations are invoked: a policy determines whether to Write a new memory unit from an observation, whether to Forget an existing unit given its age and relevance, and how to trigger Consolidation. Every operation in a memory system is regulated by at least one governance policy. The data confirm this: no operation in the surveyed systems is governed exclusively by one policy type, and all four policy types appear across multiple operations. We therefore present governance policies as the control dimension of this layer, describing the decision regime applied to the lifecycle operation vocabulary, rather than as a parallel taxonomy.

6.2 Sub-dimension A: Lifecycle Operations

The six lifecycle operations, together with their frequency of occurrence across the 81 method papers, are described below.

Write is performed by 78 of the 81 method papers and is the foundational operation through which new information enters the memory store. Write converts observations, dialogue turns, task results, or other inputs into memory units in the appropriate structural form and deposits them in the storage substrate. The near-universal occurrence of Write reflects its status as a prerequisite for any external memory system. Key design decisions at the Write operation include the granularity of the unit written, whether to write verbatim content or to extract and encode it, and whether to apply an admission filter before writing. A-MAC[14] identifies this filtering decision as the central design challenge of the Write operation: overly permissive admission leads to memory pollution with hallucinated or irrelevant content, while overly conservative admission risks discarding information essential for future reasoning.

Retrieve is performed by 79 of the 81 method papers and is the operation by which stored memory units are accessed in response to a query. Retrieve is the read-path counterpart of Write and is the operation most directly served by the indexing structures described in the Realization Layer. The design decisions at Retrieve include the retrieval strategy, whether single-step or iterative, the number of items to retrieve, and the method of ranking and filtering retrieved items before passing them downstream.

The distribution of Write and Retrieve across the 81 method papers reveals an important structural characteristic of the research landscape: not all surveyed works propose complete memory systems that manage the full lifecycle. Three papers, RECOMP[36], GraphReader[31], and KG-Agent[52], address only the Retrieve and Assemble operations, contributing improvements to the read path of memory systems while assuming a pre-existing memory substrate constructed outside their scope. One paper, Agentic Context Engineering[53], addresses Write, Update, and Assemble without implementing a dedicated retrieval mechanism, managing working context through direct assembly rather than query-driven retrieval. These works are best understood as operation-specific contributions that target particular stages of the memory lifecycle rather than end-to-end memory management systems. This distinction has implications for evaluation: operation-specific contributions are appropriately assessed on the quality of the targeted operation in isolation, whereas full lifecycle systems require evaluation across multiple operations and their interactions over extended interaction horizons.

Update is performed by 35 of the 81 method papers and modifies existing memory units in response to new information that contradicts, refines, or extends previously stored content. Update is distinct from Write in that it operates on an existing memory entry rather than creating a new one. The prevalence of Update among approximately 44 percent of surveyed systems reflects that a significant proportion of memory systems are designed for long-horizon interactions where the same facts, user preferences, or environmental states change over time. A-MEM[34] and Chain-of-Memory[46] implement Update as a continuous process triggered when new observations are found to conflict with or elaborate on existing atomic entries.

Consolidate is performed by 29 of the 81 method papers and reorganizes memory content into higher-level, more stable representations through processes such as summarization, reflection, abstraction, or clustering. Consolidate operates on existing memory content rather than on raw inputs and is the operation most closely associated with the cognitive science concept of memory consolidation, the process by which episodic experience is abstracted into semantic knowledge. EverMemOS[48] explicitly models consolidation as the operation by which fragmented episodic experiences are transformed into coherent and stable semantic structures that support long-horizon reasoning, motivated by the failure of systems that lack explicit consolidation to reason consistently over accumulated experience. Generative Agents[2] implement periodic reflection as a consolidation operation that synthesizes higher-level insights from accumulated observations. The relatively lower prevalence of Consolidate, compared to Write and Retrieve, reflects its computational cost and its deployment primarily in systems designed for extended multi-session interactions.

Forget is performed by 16 of the 81 method papers and removes or deactivates memory units based on criteria such as staleness, irrelevance, contradiction, or capacity constraints. Forget is the least common of the six operations, reflecting the field's predominant focus on memory accumulation rather than memory pruning.

Sun et al.[12] note that existing heuristic approaches, including deletion based on recency or similarity thresholds, lack a principled mechanism to reason about the long-term consequences of the Forget operation, which shapes future retrieval and downstream behavior in ways that are difficult to anticipate. Among the 16 systems implementing Forget, 11 use heuristic-based policies, typically implementing decay functions inspired by the Ebbinghaus forgetting curve, and 1 uses model-driven policies in which an LLM judges whether a memory unit should be retained or discarded. MemoryBank[42] and FadeMem[54] represent the heuristic approach; MemGPT[1] represents the model-driven approach.

Assemble is performed by 18 of the 81 method papers and composes the retrieved memory items into a context representation suitable for LLM input. Assemble is the interface between the memory system and the generation process: it determines how retrieved items are ordered, weighted, and formatted before being incorporated into the prompt or working context. ReadAgent[30] exemplifies model-driven Assemble, compressing and ordering gist memories before injection. RECOMP[36] implements a learned Assemble operation that selects and compresses retrieved passages for context injection. The relatively low prevalence of Assemble as an explicitly designed operation, compared to the near-universal prevalence of Retrieve, reflects that many systems treat context assembly as a fixed post-retrieval step rather than as a design dimension in its own right, an observation that points toward an under-explored area of the design space.

6.3 Sub-dimension B: Governance Policies

Governance policies are the decision mechanisms through which lifecycle operations are triggered and regulated. We identify four primary types, ordered from least to most adaptive: rule-based, heuristic-based, model-driven, and learned/adaptive. A fifth category, hybrid, captures the subset of systems that explicitly combine multiple policy types within a single operation. A critical empirical finding from the 81 method papers is that governance policy choice is not uniform across operations within the same system: 28 of the 81 systems apply different governance types to different operations, reflecting the intuition that the appropriate decision regime for writing new information differs from that for forgetting existing information or for consolidating episodic experience into semantic knowledge. Table 2 reports the full cross-distribution of operations and governance types across the surveyed systems.

Table 2: Operation x Governance cross-distribution (81 method papers)

	Rule-based	Heuristic-based	Model-driven	Learned/Adaptive
Write	9	4	54	11
Update	1	4	23	9
Forget	0	11	1	4
Retrieve	6	18	51	11
Assemble	4	0	12	2
Consolidate	0	1	27	2

Note: Counts are constituent-policy counts rather than mutually exclusive system counts. Hybrid systems are counted once under each policy type they combine; therefore, row totals may exceed the number of systems implementing the operation.

Rule-based policies apply fixed logical conditions to trigger or constrain memory operations without any scoring, model judgment, or learned optimization. Rule-based policies appear in 9 systems for the Write operation and 4 systems for Assemble, but are absent from Forget, and Consolidate entirely. Their presence in Write and Assemble reflects the suitability of deterministic rules for operations with well-defined triggering conditions: a new dialogue turn always triggers a Write, and a retrieved set of items can be assembled according to a fixed ordering schema. MemArchitect[55] exemplifies this paradigm, enforcing explicit rule-based policies for memory lifecycle management through a governance layer that decouples policy enforcement from model weights. The absence of rule-based policies from Update, Forget, and Consolidate reflects the difficulty of specifying deterministic conditions for operations whose correctness depends on subtle semantic judgments about relevance, contradiction, and abstraction that cannot be reduced to fixed conditions at design time.

Heuristic-based policies apply hand-crafted scoring functions that combine multiple signals, such as recency, relevance, importance, and similarity thresholds, to evaluate candidate memory operations. Heuristic-based policies appear across Write, Update, Forget, and Retrieve, but their most striking concentration is in the Forget operation, where they account for 11 of 16 systems, representing 69 percent of all systems that implement forgetting. MemoryBank[42] applies a forgetting curve function derived from the Ebbinghaus model to govern memory decay. FadeMem[54] and FOREVER[56] implement biologically-inspired decay functions that modulate memory retention based on access frequency and recency. Zep[44] implements temporal decay over its knowledge graph edges. This concentration of heuristic governance in Forget reflects a field-wide tendency to treat memory removal as a tractable engineering problem amenable to fixed scoring functions, in contrast to the more semantically demanding operations of Write and Consolidate which are predominantly governed by model-driven policies. Heuristic-based governance is conspicuously absent from Assemble and accounts for only 1 of 29 systems in Consolidate. This pattern reflects a fundamental limitation of heuristic approaches: heuristics operate on the attributes of individual memory units, such as recency, access frequency, and similarity scores, and are effective for operations that evaluate one unit at a time. Assemble and Consolidate both require cross-unit semantic integration: Assemble must judge how multiple retrieved items relate to each other and compose them into a coherent context, while Consolidate must abstract patterns across accumulated episodic entries into higher-level semantic representations. Neither task can be reduced to per-unit attribute scoring, which explains why both operations are governed almost exclusively by model-driven policies that can reason over the semantic relationships between units rather than the surface properties of individual ones.

Model-driven policies delegate the judgment of memory operations to a language model, which evaluates observations or memory units and determines appropriate actions through natural language reasoning or structured output generation. Model-driven policies are the dominant governance paradigm overall, and their dominance is particularly pronounced in two operations: Write, where 54 of 78 systems apply model-driven governance at 69 percent, and Consolidate, where 27 of 29 systems do so at 93 percent, making Consolidate the most model-driven operation in the surveyed literature. The dominance of model-driven governance in Consolidate is theoretically coherent: consolidation requires abstracting episodic experience into semantic knowledge, judging which details to preserve and which to discard, and detecting thematic structure across accumulated memories, all tasks that demand open-ended semantic judgment beyond the reach of hand-crafted heuristics. A-MEM[34] and MemOS[57] use LLM judgment to govern Write; EverMemOS[48], Amory[58], and the majority of consolidation-oriented systems use LLM judgment to govern Consolidate. The notable exception is Forget, where model-driven governance appears in only 1 of 16 systems, MemGPT[1], with the remaining 15 relying on heuristic or learned policies. This asymmetry, 93 percent model-driven Consolidate paired with 6 percent model-driven Forget, is one of the most structurally significant patterns in the data and is discussed in Section 6.4.

Learned and adaptive policies optimize memory management through training, using reinforcement learning, meta-learning, or other forms of optimization to learn policies that maximize long-term task utility. Learned policies appear across all six operations but are concentrated in Write (11 systems), Retrieve (11 systems), Update (9 systems), and Forget (4 systems). For Forget specifically, learned policies account for 4 of 16 systems, making them the second most common governance type for this operation after heuristic-based policies, consistent with the theoretical argument of Sun et al.[12] that forgetting decisions require reasoning about delayed and uncertain consequences that heuristic approaches cannot adequately model. AgeMem[59] and AtomMem[60] optimize Write, Update, and Forget jointly through reinforcement-based methods. Memory-R1[61] explicitly formulates memory operations as actions optimized by a learned policy. Learned governance is nearly absent from Consolidate, appearing in only 1 system, a gap that reflects the difficulty of defining reward signals for consolidation quality in the absence of standardized evaluation protocols.

Hybrid policies combine multiple governance types within a single operation. Hybrid governance is rare overall but most concentrated in the Retrieve operation, where 7 of 79 systems apply it, representing 9 percent of all retrieval implementations. Systems such as Zep[44], TiMem[62], and EchoKG[38] combine heuristic-based temporal or similarity filters with model-driven reranking within their retrieval pipelines, applying each mechanism where it is most appropriate: heuristics for cheap structural filtering and model judgment for fine-grained semantic ranking. The concentration of hybrid governance in Retrieve, and its near-absence

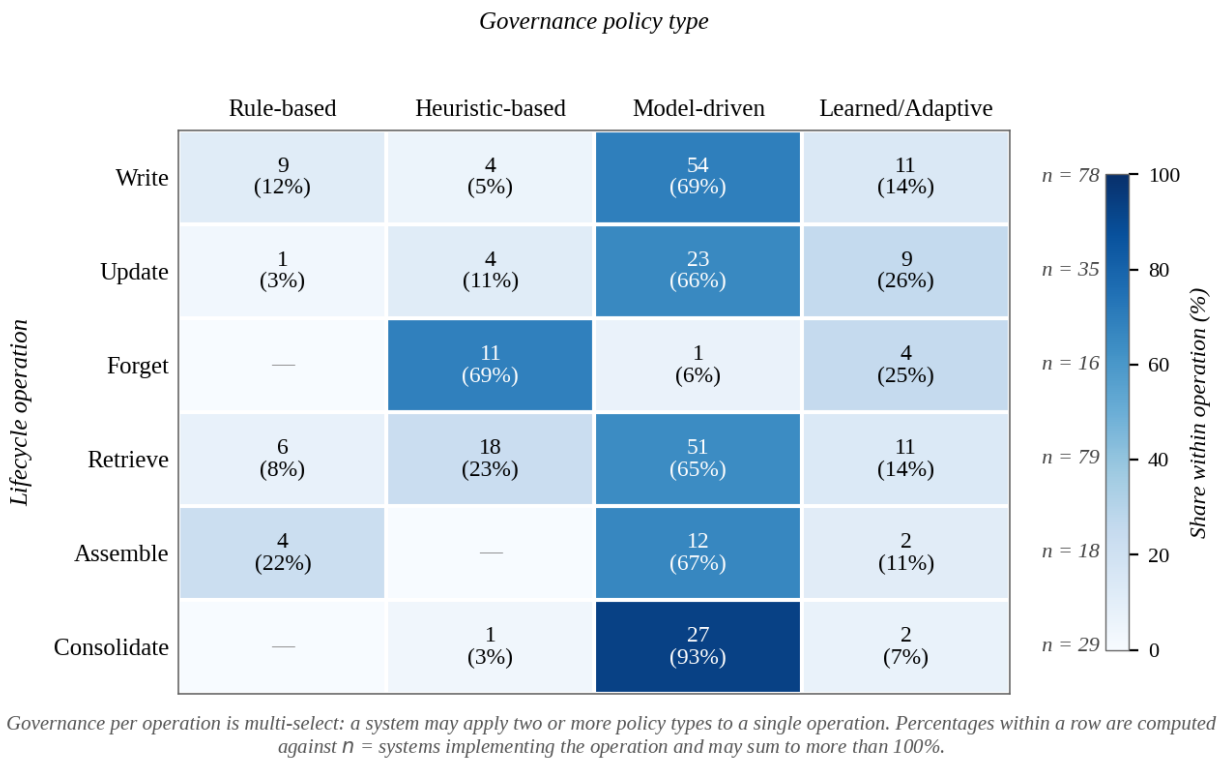

Figure 3: Distribution of governance policy types across memory lifecycle operations. The heatmap shows the proportion of studies adopting different governance policies for each lifecycle operation, highlighting the dominance of model-driven approaches across most memory management functions.

from other operations, reflects the distinctive two-phase nature of the retrieval problem: unlike Write or Forget, which involve a single decision about a single memory unit, Retrieve involves both a coarse candidate selection phase and a fine-grained ranking phase that naturally call for different governance mechanisms. GraphMem[63] and ReasoningBank[25] apply hybrid governance to Update, combining heuristic triggers for detecting potential conflicts with model-driven judgment for resolving them.

6.4 Three Empirical Patterns in the Governance Distribution

The cross-distribution in Table 2 reveals three patterns that are not visible when governance policy is treated as a single system-level attribute rather than an operation-specific one.

Pattern one: Consolidate is the most model-driven operation. At 93 percent model-driven governance (27 of 29 systems), Consolidate stands apart from all other operations. This pattern has a clear functional explanation: consolidation requires the kind of open-ended semantic judgment, abstracting, summarizing, detecting thematic structure, and resolving conflicts, that is beyond the reach of rule-based or heuristic approaches and has not yet been formulated as a tractable optimization problem for learned approaches. The near-total dominance of model-driven governance in Consolidate reflects a genuine functional constraint rather than a contingent design preference, and it implies that improvements to consolidation quality are primarily bottlenecked by the semantic capabilities of the underlying language model rather than by the governance mechanism itself.

Pattern two: Forget is the most heuristic-governed operation and the least model-driven. At 69 percent heuristic-based governance (11 of 16 systems) and only 6 percent model-driven (1 of 16 systems), Forget exhibits the opposite profile from Consolidate. This asymmetry is one of the most structurally significant findings in the data: LLMs are trusted to consolidate memory but not to forget it. The pattern likely reflects

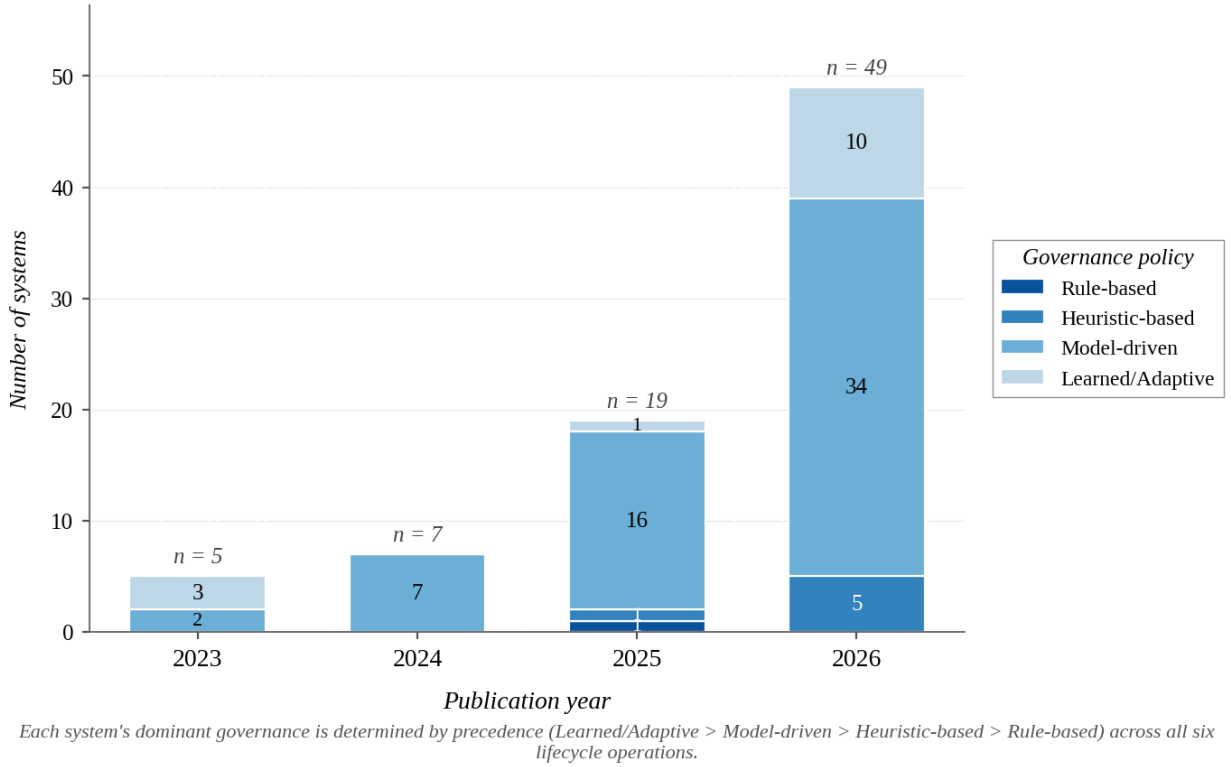

Figure 4: Governance policy types of surveyed systems by publication year. The number of systems grew rapidly from 5 in 2023 to 49 in 2026, with Model-driven systems expanding most prominently from 2 to 34 over the same period.

a combination of risk aversion, since incorrectly forgetting a memory unit is irreversible in most architectures, and the computational cost of invoking a full LLM judgment at every potential forgetting decision in a large memory store. The result is that the operation with the most consequential long-term effects on memory coherence and storage efficiency is governed by the least adaptive policies, a tension that Sun et al.[12] identify as one of the central open problems in memory governance.

Pattern three: 28 systems apply different governance types to different operations. This finding confirms that governance policy is properly characterized as an operation-level attribute rather than a system-level one. A system may apply model-driven judgment to Write and Consolidate while applying heuristic scoring to Forget and rule-based ordering to Assemble, each choice reflecting the functional demands and risk profile of the specific operation. Treating governance as a single system-level label, as most prior taxonomies do, collapses this intra-system variation and obscures the principled design logic that governs it. While prior surveys have identified memory operations as a unit of analysis, Du et al.[11] map operations to research topics and Du[18] organizes systems into mechanism families, neither provides a cross-distribution of governance policy types at the operation level. Lin et al.[64] apply a lifecycle-by-security-objective cross-tabulation to memory security analysis, but their focus is on adversarial threats rather than governance policy types. The operation-level governance analysis introduced in this survey makes this intra-system variation empirically visible across 81 surveyed systems, revealing that the governance choice for a given operation is not determined by the system's overall policy orientation but by the functional demands of that specific operation.

The three-layer analysis surfaces a set of specific, empirically grounded open challenges that are not visible under system-level or single-axis treatments of agent memory. We now examine each in turn, grounding each challenge in observations from the layer-level analysis rather than in abstract theoretical considerations.

7 Open Challenges

The three-layer analysis and the systematic classification of 81 method papers surface six open challenges that are not visible under single-axis or system-level treatments of agent memory. Each challenge is grounded in empirical observations from the surveyed literature rather than derived from abstract theoretical considerations.

7.1 The Governance Gap in Forgetting

The Forget operation is implemented by only 16 of the 81 surveyed method papers, the lowest prevalence of any of the six lifecycle operations. Among those 16 systems, 11 govern forgetting through heuristic-based policies, typically decay functions derived from the Ebbinghaus forgetting curve, and only 4 apply learned policies. No system governs forgetting through rule-based policies alone. This distribution reveals a structural tension: the operation with the most consequential long-term effects on memory coherence, storage efficiency, and retrieval quality is governed by heuristic-based policies, which apply fixed scoring functions designed at system construction time and cannot reason about the long-term consequences of individual forgetting decisions, and is absent entirely from 80 percent of surveyed systems. Sun et al.[12] identify this as a fundamental limitation of the field, arguing that forgetting decisions require reasoning about delayed and uncertain consequences that heuristic scoring functions cannot adequately model. The challenge is not merely to implement forgetting more frequently, but to develop governance mechanisms for forgetting that can reason about the long-term effects of memory removal on future retrieval quality and task performance, a problem that the dominance of greedy, per-unit heuristic scoring does not address.

7.2 The Systematic Underexploration of Sparse Indexing

Sparse keyword indexing appears in only 1 of the 81 surveyed method papers, despite well-documented evidence that sparse retrieval methods outperform dense embedding retrieval on exact-match queries, rare entity queries, and out-of-distribution vocabulary[9]. Weller et al.[15] establish a geometric bound on what embedding-based retrieval can represent regardless of model scale, and a benchmark for procedural memory retrieval[51] finds that BM25 outperforms embedding-based methods on queries involving novel object instantiations. The near-complete absence of sparse indexing from agent memory systems reflects a field-wide default to dense retrieval rather than an informed design choice based on systematic evaluation. Hybrid dense-sparse indexing, well established in the general information retrieval literature, has not been systematically evaluated in the agent memory context. The challenge is to determine empirically under what retrieval scenarios and memory content types sparse or hybrid indexing provides meaningful advantages over dense indexing alone, and to develop principled guidelines for indexing structure selection that go beyond the current default.

7.3 Procedural Memory and the Declarative-Executable Divide

Among the 10 surveyed systems whose primary memory type is procedural, 6 store procedural content as natural language summaries of plans or strategies, 2 store it as executable code, 1 as summary with chunk and 1 as raw chunks. The three structured declarative forms, atomic facts, triples, and graph structures, are entirely absent from the procedural memory row of the representation matrix. This distributional asymmetry reveals a conceptual tension at the heart of procedural memory design: the majority of systems that claim to implement procedural memory in fact store declarative descriptions of procedures rather than executable realizations of them. A natural language summary of a plan and an executable implementation of that plan impose different requirements on the retrieval pipeline, the execution environment, and the update and consolidation mechanisms. Systems such as Voyager[27] demonstrate that executable code as a structural form enables compositional reuse and direct invocation that no declarative form can provide, but code-based procedural memory remains rare in the surveyed literature. The challenge is to clarify the conditions under which declarative versus executable representations of procedural knowledge are appropriate, and to develop storage, indexing, and governance mechanisms suited to executable memory forms.

7.4 Consolidation Bottlenecked by Model Capability

Among the 29 systems that implement the Consolidate operation, 27 govern it through model-driven policies, representing 93 percent dominance by a single governance type. This near-total dependence on LLM judgment for consolidation reflects a genuine functional constraint: consolidation requires the kind of open-ended semantic abstraction, identifying thematic structure across accumulated episodic memories and distilling it into stable semantic representations, that rule-based and heuristic approaches cannot perform. However, this concentration also means that the quality of consolidation in deployed systems is directly bottlenecked by the semantic capabilities and reliability of the underlying language model. Model-driven consolidation introduces the risk of hallucinated abstractions, lossy compression, and sycophantic summarization that gradually distorts the memory store away from ground truth, a phenomenon that the SSGM framework[13] characterizes as semantic drift. The challenge is to develop consolidation mechanisms that combine the semantic flexibility of model-driven judgment with the verifiability and consistency guarantees that model-driven approaches alone cannot provide, and to design evaluation protocols that measure consolidation quality beyond surface-level recall accuracy.

7.5 Layer-Specific Evaluation

The three-layer decomposition surfaces a gap in current evaluation practice: existing benchmarks assess memory systems as end-to-end black boxes, measuring final task performance without attributing it to decisions at specific layers. A system that performs poorly on a multi-hop QA benchmark may be failing at the representation layer (storing atomic facts when triples would better support relational reasoning), at the Realization Layer (using dense ANN indexing when graph traversal is required), or at the operations layer (not implementing consolidation and therefore accumulating conflicting memories over time). Without layer-specific evaluation, these failure modes are indistinguishable, and the field cannot develop targeted improvements. Existing evaluations of agentic memory systems suffer from benchmark saturation, metric misalignment with semantic utility, and failure to account for system-level costs such as write-path latency and memory maintenance overhead[21]. The challenge is to design evaluation protocols that hold two layers constant while varying the third, enabling component-level attribution of performance differences. The distinction between operation-specific contributions, which address a single lifecycle stage, and full lifecycle systems, which manage memory end-to-end, further complicates evaluation: these two categories of systems are not comparable on the same benchmarks, yet the field currently has no principled methodology for evaluating them separately.

7.6 Multi-Agent Memory Coordination

The surveyed literature focuses almost entirely on single-agent memory systems. When multiple agents share a memory store or maintain coordinated private memories, the design challenges of all three layers are compounded in ways that single-agent systems do not face. At the representation layer, the same memory unit may need to carry provenance and role-attribution metadata so that agents can assess the trustworthiness and relevance of memories written by other agents. At the Realization Layer, shared graph or vector stores require consistency and synchronization protocols that single-agent systems can ignore. At the operations layer, Write and Update operations by one agent may invalidate memories held by another, requiring conflict resolution mechanisms that account for inter-agent authority and role boundaries. Memory in LLM-based multi-agent systems is not a trivial extension of single-agent memory but a distinct research frontier with new challenges in synchronization, access control, scalability, alignment, and safety. The most pressing open challenge in multi-agent memory is consistency: shared memory must support coordinated reads and writes across agents while maintaining a coherent and up-to-date view of the world state. The three-layer framework developed in this survey provides a vocabulary for specifying where multi-agent memory challenges arise, but the design solutions for each layer in the multi-agent setting remain largely undeveloped in the current literature.

The six challenges identified above share a common structural theme: the field has invested heavily in the write and retrieve path while underinvesting in the governance mechanisms that determine how memory

evolves over extended interaction horizons. We conclude by synthesizing the findings of the three-layer analysis and situating them within a broader view of the research trajectory.

8 Conclusion

This survey has presented a systematic account of external long-term memory systems for LLM-based agents, organized around a three-layer decomposition of the design space and grounded in the classification of 81 method-oriented papers published through early 2026. The decomposition distinguishes the Semantics Layer, which characterizes what units of memory are stored and in what logical form; the Realization Layer, which characterizes how those units are physically persisted and how access paths over them are organized; and the Dynamics Layer, which characterizes how memory is written, updated, forgotten, retrieved, assembled, and consolidated, and the governance policies that regulate those operations. The decomposition is motivated by two theoretical precedents: the ANSI/SPARC principle of data independence, which supports the separation of logical representation from physical substrate, and Marr's tri-level analysis, which argues that representation, implementation, and computational control constitute mutually irreducible levels of analysis for any information-processing system.

On the Semantics Layer, the two-axis representation matrix, combining cognitive role following CoALA with structural form following and extending Zeng et al., reveals systematic patterns of coverage and absence in the research landscape. Episodic and semantic memory are well instantiated across multiple structural forms, with episodic content appearing predominantly in raw chunks and summaries and semantic content appearing predominantly in atomic facts, triples, and graph structures. Procedural memory, by contrast, is almost entirely absent from the four declarative structural forms and is instantiated primarily in natural language summaries of plans, with executable code appearing in only two surveyed systems despite its demonstrable advantages for compositional reuse and direct invocation. This asymmetry between declarative and executable memory forms is the most structurally significant gap surfaced by the representation layer analysis.

On the Realization Layer, the analysis reveals that dense vector indexing functions as the near-universal baseline across all substrate types, appearing in 67 of 81 systems and in combination with every other indexing structure. Graph traversal is the second most prevalent indexing structure but is constrained exclusively to graph and hybrid substrates, making substrate selection effectively a choice about retrieval modality. The most consequential finding at this layer is the systematic absence of sparse keyword indexing, which appears in only 1 of 81 systems despite documented evidence of its superiority over dense retrieval on exact-match and out-of-distribution queries. This absence reflects a field-wide default rather than an informed design choice and points toward hybrid dense-sparse indexing as an underexplored direction.

On the Dynamics Layer, the per-operation governance analysis reveals a structural asymmetry that is invisible under system-level characterizations of governance policy. Consolidate is governed by model-driven policies in 93 percent of systems that implement it, reflecting the genuine functional requirement for open-ended semantic judgment in the abstraction of episodic experience into semantic knowledge. Forget, by contrast, is governed by heuristic-based policies in 69 percent of systems that implement it, and is absent from 80 percent of all surveyed systems. This asymmetry, LLMs are trusted to consolidate memory but not to forget it, encodes a field-wide risk aversion toward irreversible memory removal that leaves long-term memory coherence and storage efficiency inadequately addressed. The finding that 28 of 81 systems apply different governance types to different operations within the same system confirms that governance is an operation-level attribute rather than a system-level one, a distinction that prior taxonomies have not captured.

The six open challenges identified in Section 7 share a common structural theme: the field has invested heavily in the write and retrieve path while underinvesting in the operations and governance mechanisms that determine how memory evolves over time. Forgetting lacks principled governance. Consolidation is bottlenecked by model capability without verifiability safeguards. Sparse indexing remains unexplored. Procedural memory lacks a structural theory that distinguishes declarative from executable representations. Evaluation benchmarks measure end-to-end performance without layer-specific attribution. And multi-agent memory coordination introduces consistency and access-control challenges that single-agent frameworks are not designed to address.

Taken together, these findings suggest that the most productive direction for future work is not the development of yet more end-to-end memory systems but the systematic investigation of individual design decisions at each layer and their interactions. The three-layer framework proposed in this survey is intended as a shared analytical vocabulary for that investigation: a means of specifying what a memory system does at each level of abstraction, comparing systems that differ along one dimension while holding others constant, and identifying the combinations of layer-level choices that have not yet been explored. The field's rapid growth in the 2025 to 2026 period has produced a rich diversity of systems; the task now is to understand them well enough to build on them deliberately.

A Complete System Classification

A.1 Table A1: System Overview (81 Method Papers)

All 81 method-oriented papers surveyed, sorted by year of publication. Abbreviations — Memory Type: Ep=Episodic, Se=Semantic, Pr=Procedural, Wk=Working. Structural Form: Ch=Chunks, Su=Summaries, At=Atomic facts, Tr=Triples, Co=Code. Indexing: DANN=Dense ANN, Sp=Sparse keyword, Meta=Metadata filter, GT=Graph traversal, Hier=Hierarchical. Gov. Policy: M=Model-driven, H=Heuristic-based, L=Learned/Adaptive, R=Rule-based, Hyb=Hybrid.

#	System	Year	Memory Type	Structural Form	Storage	Indexing
1	Self-evolving Agents with reflective and memory-augmented abilities[65]	n.d.	Ep	Ch	Vector	DANN
2	Augmenting Language Models with Long-Term Memory[32]	2023	Ep	Ch	Document	DANN
3	Generative Agents: Interactive Simulacra of Human Behavior[2]	2023	Ep+Se+Pr	Ch+Su	Document	DANN+Met
4	MemoChat: Tuning LLMs to Use Memos for Consistent Long-Range Open-Domain. . . [33]	2023	Ep	Su	Document	DANN
5	RECOMP: Improving Retrieval-Augmented LMs with Compression and Selective. . . [36]	2023	Se	Su	Document	DANN
6	Voyager: An Open-Ended Embodied Agent with Large Language Models[27]	2023	Pr	Co	Document	DANN
7	My agent understands me better: Integrating Dynamic Human-like Memory Re. . . [66]	2024	Ep	Ch	Vector	DANN
8	A Human-Inspired Reading Agent with Gist Memory of Very Long Contexts[30]	2024	Wk	Su	Document	Hier
9	ExpeL: LLM Agents Are Experiential Learners[24]	2024	Pr	Su	Document	DANN
10	GraphReader: Building Graph-based Agent to Enhance Long-Context Abilitie. . . [31]	2024	Wk	Tr	Graph	GT
11	MemGPT: Towards LLMs as Operating Systems[1]	2024	Wk+Ep	Ch	KV + Document + Vector	DANN+Met
12	MemoryBank: Enhancing Large Language Models with Long-Term Memory[42]	2024	Ep+Se	Su	Vector	DANN+Met
13	SeCom: On Memory Construction and Retrieval for Personalized Conversatio. . . [67]	2024	Ep	Ch	Vector	DANN
14	A-MEM: Agentic Memory for LLM Agents[34]	2025	Ep	At	Vector	DANN
15	Agent KB: Leveraging Cross-Domain Experience for Agentic Problem Solving[39]	2025	Pr	Ch	Vector	DANN

16	AriGraph: Learning Knowledge Graph World Models with Episodic Memory[47]	2025	Ep+Se	Tr	Graph + Vector	GT+DANN
17	Collaborative Memory: Multi-User Memory Sharing in LLM Agents[68]	2025	Ep	Ch	Vector	DANN+Met
18	Cost-Efficient Serving of LLM Agents via Test-Time Plan Caching[69]	2025	Pr	Su	KV	DANN
19	D-SMART: Enhancing LLM Dialogue Consistency via Dynamic Structured Memor. . . [35]	2025	Ep	Tr	Graph	GT
20	Hello Again! LLM-powered Personalized Agent for Long-term Dialogue[70]	2025	Ep+Se	Su+At	Vector	DANN
21	Hierarchical Memory for High-Efficiency Long-Term Reasoning in LLM Agent. . . [71]	2025	Ep+Se	Ch+Su	Document + Vector	Hier+DANN
22	In Prospect and Retrospect: Reflective Memory Management for Long-term P. . . [72]	2025	Ep	Ch	Vector	DANN
23	KG-Agent: An Efficient Autonomous Agent Framework for Complex Reasoning . . . [52]	2025	Se	Tr	Graph	GT
24	LightMem: Lightweight and Efficient Memory-Augmented Generation[73]	2025	Wk+Ep	Su	Document	DANN
25	MIRIX: Multi-Agent Memory System for LLM-Based Agents[74]	2025	Ep+Se+Pr	Ch+Su+Tr	Vector + Graph + Document	DANN+GT
26	Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory[43]	2025	Se	At+Tr	Vector + Graph	DANN+GT
27	MemInsight: Autonomous Memory Augmentation for LLM Agents[75]	2025	Ep	Ch	Vector	DANN
28	MemOS: A Memory OS for AI System[57]	2025	Wk+Ep+Se	Ch+Su	Document + KV + Vector	Hier+DANN
29	Memento: Fine-tuning LLM Agents without Fine-tuning LLMs[76]	2025	Ep	Ch	Vector	DANN
30	ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory[25]	2025	Pr	Su	Vector	DANN
31	Structured Memory Mechanisms for Stable Context Representation[77]	2025	Wk+Ep	Not mentioned	Document	Meta
32	Zep: A Temporal Knowledge Graph Architecture for Agent Memory[44]	2025	Ep+Se	Tr	Graph	GT+Meta
33	A Dynamic Retrieval-Augmented Generation System with Selective Memory[78]	2026	Se	Ch	Vector	DANN+Met
34	AMA: Adaptive Memory via Multi-Agent Collaboration[79]	2026	Ep	Not mentioned	Vector	DANN
35	Aeon: High-Performance Neuro-Symbolic Memory Management for Long-Horizon. . . [80]	2026	Wk+Ep	Ch+Tr	Vector + Graph	DANN+GT
36	Agentic Context Engineering: Evolving Contexts for Self-Improving Langua. . . [53]	2026	Pr	Su	Document	Meta
37	Agentic Memory: Learning Unified Long-Term and Short-Term Memory Managem. . . [59]	2026	Wk+Ep	Not mentioned	KV + Vector	DANN+Meta
38	Agentic Plan Caching: Test-Time Memory for Fast and Cost-Efficient LLM A. . . [26]	2026	Pr	Su	KV	DANN+Sp
39	Amory: Building Coherent Narrative-Driven Agent Memory through Agentic R. . . [58]	2026	Ep+Se	Su+At	Vector	DANN
40	AtomMem: Learnable Dynamic Agentic Memory with Atomic Memory Operation[60]	2026	Ep	Not mentioned	Vector	DANN

41	BMAM: Brain-inspired Multi-Agent Memory Framework[81]	2026	Ep+Se	Ch+Su+Tr	Vector + Graph	DANN+GT
42	Beyond Dialogue Time: Temporal Semantic Memory for Personalized LLM Agen. . . [82]	2026	Se	At	Graph	GT+Meta
43	Beyond Static Summarization: Proactive Memory Extraction for LLM Agents[83]	2026	Ep	Su	Vector	DANN
44	Bi-Mem: Bidirectional Construction of Hierarchical Memory for Personaliz. . . [84]	2026	Ep+Se	Su+At	Document + Vector	Hier+DANN
45	Chain-of-Memory: Lightweight Memory Construction with Dynamic Evolution[46]	2026	Ep	Ch	KV	DANN+Met
46	Dep-Search: Learning Dependency-Aware Reasoning Traces with Persistent M. . . [85]	2026	Ep	Su	Vector	DANN
47	ES-Mem: Event Segmentation-Based Memory for Long-Term Dialogue Agents[86]	2026	Ep	Ch	Vector	DANN+Met
48	EchoKG: A Dynamic User Preference Knowledge Graph In-vehicle Dialogue Sy. . . [38]	2026	Se	Tr	Graph	GT+Meta
49	Embedded AI Companion System on Edge Devices[87]	2026	Ep	Su	Document	DANN
50	EverMemOS: A Self-Organizing Memory Operating System for Structured Long. . . [48]	2026	Ep+Se	At+Su	Document + Vector	Hier+DANN
51	FOREVER: Forgetting Curve-Inspired Memory Replay for Language Model Cont. . . [56]	2026	Se	Ch	Document	Meta
52	FadeMem: Biologically-Inspired Forgetting for Efficient Agent Memory[54]	2026	Ep	Ch	Vector	DANN+Met
53	Fine-Mem: Fine-Grained Feedback Alignment for Long-Horizon Memory Manage. . . [88]	2026	Ep	Ch	Vector	DANN
54	GLOVE: Global Verifier for LLM Memory-Environment Realignment[89]	2026	Ep	Not mentioned	Vector	DANN
55	GraphMem: Self-Evolving Graph-Based Memory for Production AI Agents[63]	2026	Se	Tr	Graph	GT
56	Grounding Agent Memory in Contextual Intent[90]	2026	Ep	Ch	Vector	DANN+Met
57	HARMONI: Multimodal Personalization of Multi-User Human-Robot Interactio. . . [91]	2026	Wk+Se	At	KV	DANN+Met
58	HiMeS: Hippocampus-inspired Memory System for Personalized AI Assistants[92]	2026	Wk+Se	Su	KV + Vector	Hier+DANN
59	HiMem: Hierarchical Long-Term Memory for LLM Long-Horizon Agents[93]	2026	Ep+Se	Ch+At	Document + Vector	Hier+DANN
60	Inside Out: Evolving User-Centric Core Memory Trees for Long-Term Person. . . [37]	2026	Se	At	Document + Graph	Hier+GT
61	Learning How to Remember: A Meta-Cognitive Management Method for Structu. . . [94]	2026	Pr	Su	Document + Vector	DANN+Hier
62	MAGMA: A Multi-Graph based Agentic Memory Architecture for AI Agents[45]	2026	Ep	Tr	Graph	GT+DANN
63	Me-Agent: A Personalized Mobile Agent with Two-Level User Habit Learning[95]	2026	Se	At	Vector	DANN
64	MemBuilder: Reinforcing LLMs for Long-Term Memory Construction via Attri. . . [96]	2026	Ep	At	Vector	DANN
65	MemCtrl: Using MLLMs as Active Memory Controllers on Embodied Agents[97]	2026	Ep	Ch	Document	Meta

66	MemRec: Collaborative Memory-Augmented Agentic Recommender System[98]	2026	Se	Tr	Vector	DANN
67	MemWeaver: Weaving Hybrid Memories for Traceable Long-Horizon Agentic Re. . . [99]	2026	Ep+Se	Tr+Su+Ch	Graph + Vector + Document	DANN+GT
68	Membox: Weaving Topic Continuity into Long-Range Memory for LLM Agents[100]	2026	Ep	Ch	Vector	DANN+Met
69	MemoBrain: Executive Memory as an Agentic Brain for Reasoning[29]	2026	Wk	Ch	Document	DANN
70	Memory-R1: Enhancing Large Language Model Agents to Manage and Utilize M. . . [61]	2026	Ep	Ch	Vector	DANN
71	Memp: Exploring Agent Procedural Memory[101]	2026	Pr	Su+Ch	Document	DANN+Hier
72	PromptX: A Cognitive Agent Platform with Long-term Memory[102]	2026	Ep+Se	Tr+Ch	Graph + Vector	GT+DANN
73	REMem: Reasoning with Episodic Memory in Language Agents[103]	2026	Ep	Tr	Graph + Vector	GT+DANN
74	SYNAPSE: Empowering LLM Agents with Episodic-Semantic Memory via Spreadi. . . [41]	2026	Ep+Se	Tr	Graph	GT
75	Self-Reflective Memory Consolidation in Agentic Architectures[104]	2026	Ep	Ch	Vector	DANN
76	SimpleMem: Efficient Lifelong Memory for LLM Agents[105]	2026	Ep	Su	Vector	DANN
77	Skill-Pro: Learning Reusable Skills from Experience via Non-Parametric P. . . [28]	2026	Pr	Co	Document	DANN+Met
78	Structured Episodic Event Memory[106]	2026	Ep+Se	Tr+Ch	Graph + Vector	DANN+GT
79	SwiftMem: Fast Agentic Memory via Query-aware Indexing[49]	2026	Ep	Ch	Vector	DANN+Met
80	TiMem: Temporal-Hierarchical Memory Consolidation for Long-Horizon Conve. . . [62]	2026	Ep	Ch	Document + Vector	Hier+DANN
81	World Model Driven Episodic Memory for LLMs[107]	2026	Ep	Tr	Vector	DANN

A.2 Table A2: Per-Operation Governance Policy

Governance policy applied to each lifecycle operation for all 81 surveyed systems. Abbreviations — M=Model-driven, H=Heuristic-based, L=Learned/Adaptive, R=Rule-based, Hyb=Hybrid, —=operation not implemented or governance not determinable. Operations: Write (W), Update (U), Forget (F), Retrieve (R), Assemble (A), Consolidate (C).

#	System	Write	Update	Forget	Retrieve	Assemble	Consolidate
1	Self-evolving Agents with reflective and memory-augmented abilities[65]	M	—	—	M	—	M
2	Augmenting Language Models with Long-Term Memory[32]	R	—	—	L	—	—
3	Generative Agents: Interactive Simulacra of Human Behavior[2]	R	—	—	H	R	M
4	MemoChat: Tuning LLMs to Use Memos for Consistent Long-Range Open-Domain. . . [33]	L	—	—	L	—	—
5	RECOMP: Improving Retrieval-Augmented LMs with Compression and Selective. . . [36]	—	—	—	L	L	—
6	Voyager: An Open-Ended Embodied Agent with Large Language Models[27]	M	M	—	M	—	—

7	My agent understands me better: Integrating Dynamic Human-like Memory Re. . . [66]	M	—	—	M	—	M
8	A Human-Inspired Reading Agent with Gist Memory of Very Long Contexts[30]	M	—	—	M	R	—
9	ExpeL: LLM Agents Are Experiential Learners[24]	M	—	—	M	—	M
10	GraphReader: Building Graph-based Agent to Enhance Long-Context Abilitie. . . [31]	—	—	—	M	M	—
11	MemGPT: Towards LLMs as Operating Systems[1]	M	M	M	M	M	—
12	MemoryBank: Enhancing Large Language Models with Long-Term Memory[42]	M	H	H	H	—	M
13	SeCom: On Memory Construction and Retrieval for Personalized Conversatio. . . [67]	M	—	—	M	—	M
14	A-MEM: Agentic Memory for LLM Agents[34]	M	M	—	M	—	—
15	Agent KB: Leveraging Cross-Domain Experience for Agentic Problem Solving[39]	M	—	—	M	—	—
16	AriGraph: Learning Knowledge Graph World Models with Episodic Memory[47]	M	M	—	M	—	—
17	Collaborative Memory: Multi-User Memory Sharing in LLM Agents[68]	R	—	—	R + H	—	—
18	Cost-Efficient Serving of LLM Agents via Test-Time Plan Caching[69]	M	—	—	H	M	—
19	D-SMART: Enhancing LLM Dialogue Consistency via Dynamic Structured Memor. . . [35]	M	M	—	M	—	—
20	Hello Again! LLM-powered Personalized Agent for Long-term Dialogue[70]	M	—	—	M	—	M
21	Hierarchical Memory for High-Efficiency Long-Term Reasoning in LLM Agent. . . [71]	M	—	—	M	—	M
22	In Prospect and Retrospect: Reflective Memory Management for Long-term P. . . [72]	M	—	—	M	—	M
23	KG-Agent: An Efficient Autonomous Agent Framework for Complex Reasoning . . . [52]	—	—	—	M	M	—
24	LightMem: Lightweight and Efficient Memory-Augmented Generation[73]	M	—	—	M	—	M
25	MIRIX: Multi-Agent Memory System for LLM-Based Agents[74]	M	M	—	M	M	—
26	Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory[43]	M	M	H	H	—	—
27	MemInsight: Autonomous Memory Augmentation for LLM Agents[75]	M	—	—	M	—	M
28	MemOS: A Memory OS for AI System[57]	M	M	H	M	M	M
29	Memento: Fine-tuning LLM Agents without Fine-tuning LLMs[76]	L	L	—	L	—	—
30	ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory[25]	M	M + H	—	M	—	—
31	Structured Memory Mechanisms for Stable Context Representation[77]	R	—	—	R	R	—
32	Zep: A Temporal Knowledge Graph Architecture for Agent Memory[44]	M	H	H	M + R	—	—
33	A Dynamic Retrieval-Augmented Generation System with Selective Memory[78]	R	H	H	H	—	—
34	AMA: Adaptive Memory via Multi-Agent Collaboration[79]	M	M	—	M	—	—
35	Aeon: High-Performance Neuro-Symbolic Memory Management for Long-Horizon. . . [80]	M	—	—	M	R	—
36	Agentic Context Engineering: Evolving Contexts for Self-Improving Langua. . . [53]	M	M	—	—	M	—

37	Agentic Memory: Learning Unified Long-Term and Short-Term Memory Managem. . . [59]	L	L	—	L	L	—
38	Agentic Plan Caching: Test-Time Memory for Fast and Cost-Efficient LLM A. . . [26]	M	—	—	H	M	—
39	Amory: Building Coherent Narrative-Driven Agent Memory through Agentic R. . . [58]	M	—	—	M	—	M
40	AtomMem: Learnable Dynamic Agentic Memory with Atomic Memory Operation[60]	L	L	L	L	—	—
41	BMAM: Brain-inspired Multi-Agent Memory Framework[81]	M	—	—	M	—	M
42	Beyond Dialogue Time: Temporal Semantic Memory for Personalized LLM Agen. . . [82]	M	M	—	M + R	—	—
43	Beyond Static Summarization: Proactive Memory Extraction for LLM Agents[83]	M	—	—	M	—	—
44	Bi-Mem: Bidirectional Construction of Hierarchical Memory for Personaliz. . . [84]	M	—	—	M	—	M
45	Chain-of-Memory: Lightweight Memory Construction with Dynamic Evolution[46]	H	—	—	H	M	—
46	Dep-Search: Learning Dependency-Aware Reasoning Traces with Persistent M. . . [85]	L	—	—	L	—	—
47	ES-Mem: Event Segmentation-Based Memory for Long-Term Dialogue Agents[86]	M	—	—	M	—	M
48	EchoKG: A Dynamic User Preference Knowledge Graph In-vehicle Dialogue Sy. . . [38]	M	M	H	H + R	—	—
49	Embedded AI Companion System on Edge Devices[87]	H	—	—	H	—	—
50	EverMemOS: A Self-Organizing Memory Operating System for Structured Long. . . [48]	M	M	H	M	—	M
51	FOREVER: Forgetting Curve-Inspired Memory Replay for Language Model Cont. . . [56]	R	—	H	H	—	—
52	FadeMem: Biologically-Inspired Forgetting for Efficient Agent Memory[54]	R	—	H	H	—	M
53	Fine-Mem: Fine-Grained Feedback Alignment for Long-Horizon Memory Manage. . . [88]	L	L	—	L	—	—
54	GLOVE: Global Verifier for LLM Memory-Environment Realignment[89]	R	M	—	H	—	—
55	GraphMem: Self-Evolving Graph-Based Memory for Production AI Agents[63]	M	M + R	H	M	—	—
56	Grounding Agent Memory in Contextual Intent[90]	M	—	—	M + H	—	—
57	HARMONI: Multimodal Personalization of Multi-User Human-Robot Interactio. . . [91]	M	M	—	M + H	—	—
58	HiMeS: Hippocampus-inspired Memory System for Personalized AI Assistants[92]	M	—	—	M	—	M
59	HiMem: Hierarchical Long-Term Memory for LLM Long-Horizon Agents[93]	M	M	—	M	—	M
60	Inside Out: Evolving User-Centric Core Memory Trees for Long-Term Person. . . [37]	L	L	L	M	—	M
61	Learning How to Remember: A Meta-Cognitive Management Method for Structu. . . [94]	L	L	—	L	—	L
62	MAGMA: A Multi-Graph based Agentic Memory Architecture for AI Agents[45]	M	—	—	M	—	—
63	Me-Agent: A Personalized Mobile Agent with Two-Level User Habit Learning[95]	M	M	—	M	—	—
64	MemBuilder: Reinforcing LLMs for Long-Term Memory Construction via Attri. . . [96]	L	L	—	L	—	—
65	MemCtrl: Using MLLMs as Active Memory Controllers on Embodied Agents[97]	M	M	—	M	—	—

66	MemRec: Collaborative Memory-Augmented Agentic Recommender System[98]	M	M	—	M	—	—
67	MemWeaver: Weaving Hybrid Memories for Traceable Long-Horizon Agentic Re. . . [99]	M	—	—	M	M	M
68	Membox: Weaving Topic Continuity into Long-Range Memory for LLM Agents[100]	H	—	—	H	—	H
69	MemoBrain: Executive Memory as an Agentic Brain for Reasoning[29]	M	—	—	M	M	—
70	Memory-R1: Enhancing Large Language Model Agents to Manage and Utilize M. . . [61]	L	L	L	L	—	—
71	Memp: Exploring Agent Procedural Memory[101]	M	M	H	M	—	—
72	PromptX: A Cognitive Agent Platform with Long-term Memory[102]	M	—	—	M	M	—
73	REMem: Reasoning with Episodic Memory in Language Agents[103]	M	—	—	M	—	—
74	SYNAPSE: Empowering LLM Agents with Episodic-Semantic Memory via Spreadi. . . [41]	M	—	—	M	—	M
75	Self-Reflective Memory Consolidation in Agentic Architectures[104]	M	—	—	M	—	M + L
76	SimpleMem: Efficient Lifelong Memory for LLM Agents[105]	M	—	—	—	—	M
77	Skill-Pro: Learning Reusable Skills from Experience via Non-Parametric P. . . [28]	L	L	L	M	—	—
78	Structured Episodic Event Memory[106]	M	—	—	M	—	M
79	SwiftMem: Fast Agentic Memory via Query-aware Indexing[49]	H	—	—	H	—	—
80	TiMem: Temporal-Hierarchical Memory Consolidation for Long-Horizon Conve. . . [62]	R	M	—	H + R	—	M
81	World Model Driven Episodic Memory for LLMs[107]	M	—	—	M	—	—

Legend

M — Model-driven: Governance delegated to an LLM for semantic judgment.

H — Heuristic-based: Hand-crafted scoring functions (recency, similarity, importance).

L — Learned/Adaptive: Policy optimized via reinforcement learning or training.

R — Rule-based: Fixed deterministic conditions applied without model judgment.

Hyb — Hybrid: Explicit combination of two or more policy types within one operation.

— : Operation not implemented, or governance type not determinable from source paper.

Declaration of Generative AI and AI-Assisted Technologies in the Writing Process

During the preparation of this manuscript, the author used OpenAI ChatGPT/Codex and Anthropic Claude as AI-assisted tools to support manuscript development. These tools were used for language editing, improving clarity and organization, checking consistency across sections, identifying possible logical or numerical inconsistencies, assisting with table and figure formatting, and generating or refining Python code for reproducible figure preparation from author-provided data. The tools were also used to help compare wording against common publisher expectations for transparent AI-use disclosure.

The AI tools were not used as authors, were not used to make final scholarly judgments, and were not used to generate or fabricate research data, citations, or empirical findings. All classifications, numerical counts, references, interpretations, and conclusions were reviewed, verified, revised, and approved by the author. The author takes full responsibility for the accuracy, integrity, originality, and final content of the manuscript.

References

[1] C. Packer et al., "MemGPT: Towards LLMs as Operating Systems," Feb. 12, 2024, arXiv: arXiv:2310.08560. doi: 10.48550/arXiv.2310.08560.

[2] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative Agents: Interactive Simulacra of Human Behavior," in Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, San Francisco CA USA: ACM, Oct. 2023, pp. 1–22. doi: 10.1145/3586183.3606763.

[3] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems, Curran Associates, Inc., 2020, pp. 9459–9474. Accessed: Feb. 24, 2026. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[4] D. Tsichritzis and A. Klug, "The ANSI/X3/SPARC DBMS framework report of the study group on database management systems," Inf. Syst., vol. 3, no. 3, pp. 173–191, 1978.

[5] D. Marr, Vision: A computational investigation into the human representation and processing of visual information. MIT press, 2010. Accessed: May 05, 2026. [Online]. Available: https://books.google.com/books?hl=zh-CN&lr=&id=D8XxCwAAQBAJ&oi=fnd&pg=PR7&dq=Vision:+A+Computational+Investigation+into+the+Human+Representation+and+Processing+of+Visual+Information.+W.+H.+Free&ots=KJQJJ5W5fD&sig=KgspgBpmY5J62DeyMemQX2vagqU

[6] T. Sumers, S. Yao, K. R. Narasimhan, and T. L. Griffiths, "Cognitive Architectures for Language Agents," Trans. Mach. Learn. Res., Oct. 2023, Accessed: Apr. 30, 2026. [Online]. Available: https://openreview.net/forum?id=1i6ZCvflQJ

[7] R. Zeng, J. Fang, S. Liu, and Z. Meng, "On the Structural Memory of LLM Agents," Dec. 17, 2024, arXiv: arXiv:2412.15266. doi: 10.48550/arXiv.2412.15266.

[8] R. Cattell, "Scalable SQL and NoSQL data stores," ACM SIGMOD Rec., vol. 39, no. 4, pp. 12–27, May 2011, doi: 10.1145/1978915.1978919.

[9] J. Lin, R. Nogueira, and A. Yates, Pretrained Transformers for Text Ranking: BERT and Beyond. Springer Nature, 2022.

[10] Z. Zhang et al., "A survey on the memory mechanism of large language model-based agents," ACM Trans. Inf. Syst., vol. 43, no. 6, pp. 1–47, 2025.

[11] Y. Du et al., "Rethinking Memory in LLM based Agents: Representations, Operations, and Emerging Topics," Dec. 24, 2025, arXiv: arXiv:2505.00675. doi: 10.48550/arXiv.2505.00675.

[12] C. Sun, X. Chen, J. Luo, D. Zhang, and X. Li, "Beyond Heuristics: A Decision-Theoretic Framework for Agent Memory Management," Dec. 25, 2025, arXiv: arXiv:2512.21567. doi: 10.48550/arXiv.2512.21567.

[13] C. Lam, J. Li, L. Zhang, and K. Zhao, "Governing Evolving Memory in LLM Agents: Risks, Mechanisms, and the Stability and Safety Governed Memory (SSGM) Framework," Mar. 12, 2026, arXiv: arXiv:2603.11768. doi: 10.48550/arXiv.2603.11768.

[14] G. Zhang et al., "Adaptive Memory Admission Control for LLM Agents," Mar. 04, 2026, arXiv: arXiv:2603.04549. doi: 10.48550/arXiv.2603.04549.

[15] O. Weller, M. Boratko, I. Naim, and J. Lee, "On the Theoretical Limitations of Embedding-Based Retrieval," arXiv.org. Accessed: Feb. 24, 2026. [Online]. Available: https://arxiv.org/abs/2508.21038v1

[16] Y. Wu et al., "From Human Memory to AI Memory: A Survey on Memory Mechanisms in the Era of LLMs," Apr. 23, 2025, arXiv: arXiv:2504.15965. doi: 10.48550/arXiv.2504.15965.

[17] Z. He et al., "Human-inspired Perspectives: A Survey on AI Long-term Memory," arXiv.org. Accessed: Apr. 30, 2026. [Online]. Available: https://arxiv.org/abs/2411.00489v2

[18] P. Du, "Memory for Autonomous LLM Agents:Mechanisms, Evaluation, and Emerging Frontiers," Mar. 08, 2026, arXiv: arXiv:2603.07670. doi: 10.48550/arXiv.2603.07670.

[19] C. Yang et al., "Graph-based Agent Memory: Taxonomy, Techniques, and Applications," Feb. 05, 2026, arXiv: arXiv:2602.05665. doi: 10.48550/arXiv.2602.05665.

[20] S. Wu and K. Shu, "Memory in llm-based multi-agent systems: Mechanisms, challenges, and collective intelligence," Authorea Prepr., 2025, Accessed: May 10, 2026. [Online]. Available: https://www.techrxiv.org/doi/full/10.36227/techrxiv.176539617.79044553

[21] D. Jiang et al., "Anatomy of agentic memory: Taxonomy and empirical analysis of evaluation and system limitations," ArXiv Prepr. ArXiv260219320, 2026, Accessed: May 05, 2026. [Online]. Available: https://arxiv.org/abs/2602.19320

[22] E. W. Dijkstra, "On the Role of Scientific Thought," in Selected Writings on Computing: A personal Perspective, New York, NY: Springer New York, 1982, pp. 60–66. doi: 10.1007/978-1-4612-5695-3_12.

[23] R. Elmasri, S. B. Navathe, R. Elmasri, and S. B. Navathe, "Fundamentals of Database Systems," in Advances in Databases and Information Systems: 24th European Conference, ADBIS 2020, Lyon, France, August 25–27, 2020, Proceedings, Springer Nature, 2020, p. 139. Accessed: May 05, 2026. [Online]. Available: https://books.google.com/books?hl=zh-CN&lr=&id=Me_3DwAAQBAJ&oi=fnd&pg=PA137&dq=Fundamentals+of+Database+System&ots=TjglnNCxn5&sig=cGKrqoP06-4QAEB-FBTTqQdYifc

[24] A. Zhao, D. Huang, Q. Xu, M. Lin, Y.-J. Liu, and G. Huang, "ExpeL: LLM Agents Are Experiential Learners," Proc. AAAI Conf. Artif. Intell., vol. 38, no. 17, pp. 19632–19642, Mar. 2024, doi: 10.1609/aaai.v38i17.29936.

[25] S. Ouyang et al., "ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory," Sep. 29, 2025, arXiv: arXiv:2509.25140. doi: 10.48550/arXiv.2509.25140.

[26] Q. Zhang, M. Wornow, G. Wan, and K. Olukotun, "Agentic Plan Caching: Test-Time Memory for Fast and Cost-Efficient LLM Agents," Jan. 26, 2026, arXiv: arXiv:2506.14852. doi: 10.48550/arXiv.2506.14852.

[27] G. Wang et al., "Voyager: An Open-Ended Embodied Agent with Large Language Models," Oct. 19, 2023, arXiv: arXiv:2305.16291. doi: 10.48550/arXiv.2305.16291.

[28] Q. Mi et al., "Skill-Pro: Learning Reusable Skills from Experience via Non-Parametric PPO for LLM Agents," Apr. 16, 2026, arXiv: arXiv:2602.01869. doi: 10.48550/arXiv.2602.01869.

[29] H. Qian, Z. Cao, and Z. Liu, "MemoBrain: Executive Memory as an Agentic Brain for Reasoning," Jan. 12, 2026, arXiv: arXiv:2601.08079. doi: 10.48550/arXiv.2601.08079.

[30] K.-H. Lee, X. Chen, H. Furuta, J. Canny, and I. Fischer, "A Human-Inspired Reading Agent with Gist Memory of Very Long Contexts," Jul. 22, 2024, arXiv: arXiv:2402.09727. doi: 10.48550/arXiv.2402.09727.

[31] S. Li et al., "GraphReader: Building Graph-based Agent to Enhance Long-Context Abilities of Large Language Models," in Findings of the Association for Computational Linguistics: EMNLP 2024, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 12758–12786. doi: 10.18653/v1/2024.findings-emnlp.746.

[32] W. Wang et al., "Augmenting Language Models with Long-Term Memory," Adv. Neural Inf. Process. Syst., vol. 36, pp. 74530–74543, Dec. 2023.

[33] J. Lu et al., "MemoChat: Tuning LLMs to Use Memos for Consistent Long-Range Open-Domain Conversation," Aug. 23, 2023, arXiv: arXiv:2308.08239. doi: 10.48550/arXiv.2308.08239.

[34] W. Xu, Z. Liang, K. Mei, H. Gao, J. Tan, and Y. Zhang, "A-MEM: Agentic Memory for LLM Agents," Oct. 08, 2025, arXiv: arXiv:2502.12110. doi: 10.48550/arXiv.2502.12110.

[35] X. Lei, Q. Li, M. Zhang, and M. Zhang, "D-SMART: Enhancing LLM Dialogue Consistency via Dynamic Structured Memory And Reasoning Tree," Oct. 15, 2025, arXiv: arXiv:2510.13363. doi: 10.48550/arXiv.2510.13363.

[36] F. Xu, W. Shi, and E. Choi, "RECOMP: Improving Retrieval-Augmented LMs with Compression and Selective Augmentation," Oct. 06, 2023, arXiv: arXiv:2310.04408. doi: 10.48550/arXiv.2310.04408.

[37] J. Zhao et al., "Inside Out: Evolving User-Centric Core Memory Trees for Long-Term Personalized Dialogue Systems," Jan. 25, 2026, arXiv: arXiv:2601.05171. doi: 10.48550/arXiv.2601.05171.

[38] Y. Liang, "EchoKG: A Dynamic user Preference Knowledge Graph In-vehicle Dialogue System Based on Ebbinghaus Forgetting Curve," Int. J. Adv. AI Appl., vol. 2, no. 2, pp. 15–23, 2026.

[39] X. Tang et al., "Agent KB: Leveraging Cross-Domain Experience for Agentic Problem Solving," Oct. 27, 2025, arXiv: arXiv:2507.06229. doi: 10.48550/arXiv.2507.06229.

[40] B. J. Gutiérrez, Y. Shu, Y. Gu, M. Yasunaga, and Y. Su, "Hipporag: Neurobiologically inspired long-term memory for large language models," Adv. Neural Inf. Process. Syst., vol. 37, pp. 59532–59569, 2024.

[41] H. Jiang et al., "SYNAPSE: Empowering LLM Agents with Episodic-Semantic Memory via Spreading Activation," Jan. 21, 2026, arXiv: arXiv:2601.02744. doi: 10.48550/arXiv.2601.02744.

[42] W. Zhong, L. Guo, Q. Gao, H. Ye, and Y. Wang, "MemoryBank: Enhancing Large Language Models with Long-Term Memory," Proc. AAAI Conf. Artif. Intell., vol. 38, no. 17, pp. 19724–19731, Mar. 2024, doi: 10.1609/aaai.v38i17.29946.

[43] P. Chhikara, D. Khant, S. Aryan, T. Singh, and D. Yadav, "Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory," Apr. 28, 2025, arXiv: arXiv:2504.19413. doi: 10.48550/arXiv.2504.19413.

[44] P. Rasmussen, P. Paliychuk, T. Beauvais, J. Ryan, and D. Chalef, "Zep: A Temporal Knowledge Graph Architecture for Agent Memory," Jan. 20, 2025, arXiv: arXiv:2501.13956. doi: 10.48550/arXiv.2501.13956.

[45] D. Jiang, Y. Li, G. Li, and B. Li, "MAGMA: A Multi-Graph based Agentic Memory Architecture for AI Agents," Jan. 06, 2026, arXiv: arXiv:2601.03236. doi: 10.48550/arXiv.2601.03236.

[46] X. Xu et al., "Chain-of-Memory: Lightweight Memory Construction with Dynamic Evolution for LLM Agents," Jan. 14, 2026, arXiv: arXiv:2601.14287. doi: 10.48550/arXiv.2601.14287.

[47] P. Anokhin et al., "AriGraph: Learning Knowledge Graph World Models with Episodic Memory for LLM Agents," May 15, 2025, arXiv: arXiv:2407.04363. doi: 10.48550/arXiv.2407.04363.

[48] C. Hu et al., "EverMemOS: A Self-Organizing Memory Operating System for Structured Long-Horizon Reasoning," Jan. 09, 2026, arXiv: arXiv:2601.02163. doi: 10.48550/arXiv.2601.02163.

[49] A. Tian et al., "SwiftMem: Fast Agentic Memory via Query-aware Indexing," Jan. 13, 2026, arXiv: arXiv:2601.08160. doi: 10.48550/arXiv.2601.08160.

[50] D. Edge et al., "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," Feb. 19, 2025, arXiv: arXiv:2404.16130. doi: 10.48550/arXiv.2404.16130.

[51] I. Kohar and A. Krishnan, "A Benchmark for Procedural Memory Retrieval in Language Agents," Nov. 21, 2025, arXiv: arXiv:2511.21730. doi: 10.48550/arXiv.2511.21730.

[52] J. Jiang et al., "KG-Agent: An Efficient Autonomous Agent Framework for Complex Reasoning over Knowledge Graph," in Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds., Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 9505–9523. doi: 10.18653/v1/2025.acl-long.468.

[53] Q. Zhang et al., "Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models," Jan. 29, 2026, arXiv: arXiv:2510.04618. doi: 10.48550/arXiv.2510.04618.

[54] L. Wei, X. Dong, X. Peng, N. Xie, and B. Wang, "FadeMem: Biologically-Inspired Forgetting for Efficient Agent Memory," Jan. 26, 2026, arXiv: arXiv:2601.18642. doi: 10.48550/arXiv.2601.18642.

[55] L. S. Kumar, Y. Ba, and R. Pan, "MemArchitect: A Policy Driven Memory Governance Layer," Mar. 18, 2026, arXiv: arXiv:2603.18330. doi: 10.48550/arXiv.2603.18330.

[56] Y. Feng et al., "FOREVER: Forgetting Curve-Inspired Memory Replay for Language Model Continual Learning," Jan. 07, 2026, arXiv: arXiv:2601.03938. doi: 10.48550/arXiv.2601.03938.

[57] Z. Li et al., "MemOS: A Memory OS for AI System," Dec. 03, 2025, arXiv: arXiv:2507.03724. doi: 10.48550/arXiv.2507.03724.

[58] Y. Zhou, X. Guo, B. Bayar, and S. H. Sengamedu, "Amory: Building Coherent Narrative-Driven Agent Memory through Agentic Reasoning," Jan. 09, 2026, arXiv: arXiv:2601.06282. doi: 10.48550/arXiv.2601.06282.

[59] Y. Yu et al., "Agentic Memory: Learning Unified Long-Term and Short-Term Memory Management for Large Language Model Agents," Jan. 05, 2026, arXiv: arXiv:2601.01885. doi: 10.48550/arXiv.2601.01885.

[60] Y. Huo, Y. Lu, Z. Zhang, H. Chen, and Y. Lin, "AtomMem : Learnable Dynamic Agentic Memory with Atomic Memory Operation," Jan. 27, 2026, arXiv: arXiv:2601.08323. doi: 10.48550/arXiv.2601.08323.

[61] S. Yan et al., "Memory-R1: Enhancing Large Language Model Agents to Manage and Utilize Memories via Reinforcement Learning," Jan. 14, 2026, arXiv: arXiv:2508.19828. doi: 10.48550/arXiv.2508.19828.

[62] K. Li et al., "TiMem: Temporal-Hierarchical Memory Consolidation for Long-Horizon Conversational Agents," Jan. 06, 2026, arXiv: arXiv:2601.02845. doi: 10.48550/arXiv.2601.02845.

[63] A.-A. Ibrahim, V. Vadlamani, and A. Patil, "GraphMem: Self-Evolving Graph-Based Memory for Production AI Agents," 2026, Accessed: Jan. 31, 2026. [Online]. Available: https://www.researchgate.net/profile/Alamin-Ibrahim-2/publication/398203328_GraphMem_Self-Evolving_Graph-Based_Memory_for_Production_AI_Agents/links/6957dd5827359023a012ca37/GraphMem-Self-Evolving-Graph-Based-Memory-for-Production-AI-Agents.pdf

[64] Z. Lin, C. Li, and K. Chen, "A Survey on the Security of Long-Term Memory in LLM Agents: Toward Mnemonic Sovereignty," Apr. 17, 2026, arXiv: arXiv:2604.16548. doi: 10.48550/arXiv.2604.16548.

[65] Xuechen Liang, "Self-evolving Agents with reflective and memory-augmented abilities," in LLM-based Multi-Agent Systems: Towards Responsible, Reliable, and Scalable Agentic Systems, Accessed: Jan. 31, 2026. [Online]. Available: https://openreview.net/forum?id=6Mw2fO3ejN

[66] Y. Hou, H. Tamoto, and H. Miyashita, "'My agent understands me better': Integrating Dynamic Human-like Memory Recall and Consolidation in LLM-Based Agents," in Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, in CHI EA '24. New York, NY, USA: Association for Computing Machinery, May 2024, pp. 1–7. doi: 10.1145/3613905.3650839.

[67] Z. Pan et al., "SeCom: On Memory Construction and Retrieval for Personalized Conversational Agents," presented at the The Thirteenth International Conference on Learning Representations, Oct. 2024. Accessed: Feb. 10, 2026. [Online]. Available: https://openreview.net/forum?id=xKDZAW0He3

[68] A. Rezazadeh, Z. Li, A. Lou, Y. Zhao, W. Wei, and Y. Bao, "Collaborative Memory: Multi-User Memory Sharing in LLM Agents with Dynamic Access Control," May 23, 2025, arXiv: arXiv:2505.18279. doi: 10.48550/arXiv.2505.18279.

[69] Q. Zhang, M. Wornow, and K. Olukotun, "Cost-Efficient Serving of LLM Agents via Test-Time Plan Caching," presented at the ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models, Jul. 2025. Accessed: Feb. 18, 2026. [Online]. Available: https://openreview.net/forum?id=LEl6eu1g1T&noteId=LEl6eu1g1T

[70] H. Li, C. Yang, A. Zhang, Y. Deng, X. Wang, and T.-S. Chua, "Hello Again! LLM-powered Personalized Agent for Long-term Dialogue," in Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), L. Chiruzzo, A. Ritter, and L. Wang, Eds., Albuquerque, New Mexico: Association for Computational Linguistics, Apr. 2025, pp. 5259–5276. doi: 10.18653/v1/2025.naacl-long.272.

[71] H. Sun and S. Zeng, "Hierarchical Memory for High-Efficiency Long-Term Reasoning in LLM Agents," Jul. 23, 2025, arXiv: arXiv:2507.22925. doi: 10.48550/arXiv.2507.22925.

[72] Z. Tan et al., "In Prospect and Retrospect: Reflective Memory Management for Long-term Personalized Dialogue Agents," in Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds., Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 8416–8439. doi: 10.18653/v1/2025.acl-long.413.

[73] J. Fang et al., "LightMem: Lightweight and Efficient Memory-Augmented Generation," Nov. 26, 2025, arXiv: arXiv:2510.18866. doi: 10.48550/arXiv.2510.18866.

[74] Y. Wang and X. Chen, "MIRIX: Multi-Agent Memory System for LLM-Based Agents," Jul. 10, 2025, arXiv: arXiv:2507.07957. doi: 10.48550/arXiv.2507.07957.

[75] R. Salama et al., "MemInsight: Autonomous Memory Augmentation for LLM Agents," in Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng, Eds., Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 33136–33152. doi: 10.18653/v1/2025.emnlp-main.1683.

[76] H. Zhou et al., "Memento: Fine-tuning LLM Agents without Fine-tuning LLMs," Aug. 25, 2025, arXiv: arXiv:2508.16153. doi: 10.48550/arXiv.2508.16153.

[77] Y. Xing, T. Yang, Y. Qi, M. Wei, Y. Cheng, and H. Xin, "Structured Memory Mechanisms for Stable Context Representation in Large Language Models," May 28, 2025, arXiv: arXiv:2505.22921. doi: 10.48550/arXiv.2505.22921.

[78] O. Bursa, "A Dynamic Retrieval-Augmented Generation System with Selective Memory and Remembrance," Jan. 04, 2026, arXiv: arXiv:2601.02428. doi: 10.48550/arXiv.2601.02428.

[79] W. Huang et al., "AMA: Adaptive Memory via Multi-Agent Collaboration," Jan. 28, 2026, arXiv: arXiv:2601.20352. doi: 10.48550/arXiv.2601.20352.

[80] M. Arslan, "Aeon: High-Performance Neuro-Symbolic Memory Management for Long-Horizon LLM Agents," Jan. 14, 2026, arXiv: arXiv:2601.15311. doi: 10.48550/arXiv.2601.15311.

[81] Y. Li, J. Liu, Y. Wang, Y. Wu, and M. Xu, "BMAM: Brain-inspired Multi-Agent Memory Framework," Jan. 28, 2026, arXiv: arXiv:2601.20465. doi: 10.48550/arXiv.2601.20465.

[82] M. Su et al., "Beyond Dialogue Time: Temporal Semantic Memory for Personalized LLM Agents," Jan. 12, 2026, arXiv: arXiv:2601.07468. doi: 10.48550/arXiv.2601.07468.

[83] C. Yang, Z. Sun, W. Wei, and W. Hu, "Beyond Static Summarization: Proactive Memory Extraction for LLM Agents," Jan. 08, 2026, arXiv: arXiv:2601.04463. doi: 10.48550/arXiv.2601.04463.

[84] W. Mao et al., "Bi-Mem: Bidirectional Construction of Hierarchical Memory for Personalized LLMs via Inductive-Reflective Agents," Jan. 10, 2026, arXiv: arXiv:2601.06490. doi: 10.48550/arXiv.2601.06490.

[85] Y. Liu et al., "Dep-Search: Learning Dependency-Aware Reasoning Traces with Persistent Memory," Jan. 26, 2026, arXiv: arXiv:2601.18771. doi: 10.48550/arXiv.2601.18771.

[86] H. Zou et al., "ES-Mem: Event Segmentation-Based Memory for Long-Term Dialogue Agents," Jan. 13, 2026, arXiv: arXiv:2601.07582. doi: 10.48550/arXiv.2601.07582.

[87] R. Gupta and S. D. H. Hsu, "Embedded AI Companion System on Edge Devices," Jan. 13, 2026, arXiv: arXiv:2601.08128. doi: 10.48550/arXiv.2601.08128.

[88] W. Ma et al., "Fine-Mem: Fine-Grained Feedback Alignment for Long-Horizon Memory Management," Jan. 13, 2026, arXiv: arXiv:2601.08435. doi: 10.48550/arXiv.2601.08435.

[89] X. Yin and H. Du, "GLOVE: Global Verifier for LLM Memory-Environment Realignment," Jan. 27, 2026, arXiv: arXiv:2601.19249. doi: 10.48550/arXiv.2601.19249.

[90] R. Yang, Y. Jiang, Y. Jiang, P. Kargupta, Y. Zhang, and J. Han, "Grounding Agent Memory in Contextual Intent," Jan. 15, 2026, arXiv: arXiv:2601.10702. doi: 10.48550/arXiv.2601.10702.

[91] J. Malécot et al., "HARMONI: Multimodal Personalization of Multi-User Human-Robot Interactions with LLMs," Jan. 27, 2026, arXiv: arXiv:2601.19839. doi: 10.48550/arXiv.2601.19839.

[92] H. Li, F. Li, W. Que, and X. Fan, "HiMeS: Hippocampus-inspired Memory System for Personalized AI Assistants," Jan. 06, 2026, arXiv: arXiv:2601.06152. doi: 10.48550/arXiv.2601.06152.

[93] N. Zhang, X. Yang, Z. Tan, W. Deng, and W. Wang, "HiMem: Hierarchical Long-Term Memory for LLM Long-Horizon Agents," Jan. 10, 2026, arXiv: arXiv:2601.06377. doi: 10.48550/arXiv.2601.06377.

[94] S. Liang et al., "Learning How to Remember: A Meta-Cognitive Management Method for Structured and Transferable Agent Memory," Jan. 12, 2026, arXiv: arXiv:2601.07470. doi: 10.48550/arXiv.2601.07470.

[95] S. Wang et al., "Me-Agent: A Personalized Mobile Agent with Two-Level User Habit Learning for Enhanced Interaction," Jan. 28, 2026, arXiv: arXiv:2601.20162. doi: 10.48550/arXiv.2601.20162.

[96] Z. Shen, Z. Wu, F. Lai, S. Lian, and Y. Rao, "MemBuilder: Reinforcing LLMs for Long-Term Memory Construction via Attributed Dense Rewards," Jan. 09, 2026, arXiv: arXiv:2601.05488. doi: 10.48550/arXiv.2601.05488.

[97] V. S. Dorbala and D. Manocha, "MemCtrl: Using MLLMs as Active Memory Controllers on Embodied Agents," Jan. 28, 2026, arXiv: arXiv:2601.20831. doi: 10.48550/arXiv.2601.20831.

[98] W. Chen et al., "MemRec: Collaborative Memory-Augmented Agentic Recommender System," Jan. 17, 2026, arXiv: arXiv:2601.08816. doi: 10.48550/arXiv.2601.08816.

[99] J. Ye et al., "MemWeaver: Weaving Hybrid Memories for Traceable Long-Horizon Agentic Reasoning," Jan. 26, 2026, arXiv: arXiv:2601.18204. doi: 10.48550/arXiv.2601.18204.

[100] D. Tao, G. Ma, Y. Huang, and M. Jiang, "Membox: Weaving Topic Continuity into Long-Range Memory for LLM Agents," Jan. 20, 2026, arXiv: arXiv:2601.03785. doi: 10.48550/arXiv.2601.03785.

[101] R. Fang et al., "Memp: Exploring Agent Procedural Memory," Apr. 15, 2026, arXiv: arXiv:2508.06433. doi: 10.48550/arXiv.2508.06433.

[102] B. Wang et al., "PromptX: A Cognitive Agent Platform with Long-term Memory," 2026, Accessed: Jan. 31, 2026. [Online]. Available: `https://research.deepractice.ai/PromptX_WWW26_Paper.pdf`

[103] Y. Shu et al., "REMem: Reasoning with Episodic Memory in Language Agent," Feb. 28, 2026, arXiv: arXiv:2602.13530. doi: 10.48550/arXiv.2602.13530.

[104] V. Putta, K. Areti, A. Guyyala, and P. Ratna, "Self-Reflective Memory Consolidation in Agentic Architectures," Int. J. Comput. Appl., vol. 187, p. 1, Jan. 2026, doi: 10.5120/ijca2026926236.

[105] J. Liu et al., "SimpleMem: Efficient Lifelong Memory for LLM Agents," Jan. 29, 2026, arXiv: arXiv:2601.02553. doi: 10.48550/arXiv.2601.02553.

[106] Z. Lu, D. Li, Y. Shi, B. Wang, L. Wang, and B. Hu, "Structured Episodic Event Memory," Jan. 10, 2026, arXiv: arXiv:2601.06411. doi: 10.48550/arXiv.2601.06411.

[107] S. Rajesh, P. S. Holur, C. Duan, and D. Chong, "World Model Driven Episodic Memory for LLMs," in NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning, 2026. Accessed: Jan. 31, 2026. [Online]. Available: `https://openreview.net/forum?id=zxYTR2UBG2`