Your Agents Are Aging Too: Agent Lifespan Engineering for Deployed Systems

Jianing Zhu[1]* **Yeonju Ro**[1]* **John T. Robertson**[1] **Kevin Wang**[1] **Junbo Li**[1]
Haris Vikalo[1] **Aditya Akella**[1] **Zhangyang Wang**[1]

[1]The University of Texas at Austin

Abstract

Long-lived AI agents are increasingly deployed as persistent operational systems, yet they are still mostly evaluated like freshly initialized models. Day-one benchmarks miss a basic systems question: *how does an agent's reliability change after deployment*? Even when model weights are frozen, an agent's effective state keeps changing as it compresses interaction history, retrieves from a growing memory store, revises facts after updates, and undergoes routine maintenance. Reliability therefore becomes a lifespan property of the full agent harness, not only a snapshot property of the base model. We introduce AGINGBENCH, a longitudinal reliability benchmark for **agent lifespan engineering**: measuring not only whether deployed agents degrade, but what form the degradation takes and where repair should target. AGINGBENCH organizes agent *aging* into four mechanisms: *compression aging*, where writing and rewriting drops future-relevant details; *interference aging*, where accumulated similar memories crowd out the target fact; *revision aging*, where changed or derived state is not updated correctly; and *maintenance aging*, where lifecycle events such as model swapping trigger regressions. To diagnose these failures, AGINGBENCH uses temporal dependency graphs and counterfactual probes that produce diagnostic profiles for the write, retrieval, and utilization stages of the memory pipeline. Across 6 scenarios, 10 memory systems and 6 models from two families, we find that **agent aging is not one-dimensional** over 400 runs spanning 50–200 sessions: a well-formed reply can hide a value the memory no longer supplies; derived state collapses as updates accumulate; and the same wrong answer can require different repairs depending on what the diagnostic profile points to. These results suggest that reliable agent deployment requires lifespan evaluation, mechanism-level diagnosis, and stage-targeted repair, not only stronger day-one models. Our project is publicly available at: `https://agingbench.github.io`.

1 Introduction

The reliability of long-running systems degrades over time: databases accumulate stale indices [3]; software accrues technical debt [32]; human memory fades, distorts, and conflates experiences with age [9]. Memory-enabled AI agents [7, 27, 11], systems that persist state across sessions, are subject to the same pressure, which we term *aging*. Even with frozen weights, their effective system state [42] keeps changing as coding agents accumulate workspace context across sessions [5, 28] and enterprise agents grow project context over months [44]. These systems also undergo routine operational events: model updates, memory compaction, or prompt revisions. The decline can be gradual and partly hidden: an agent may keep answering fluently while the exact value that matters has disappeared, a

*Equal Contribution

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

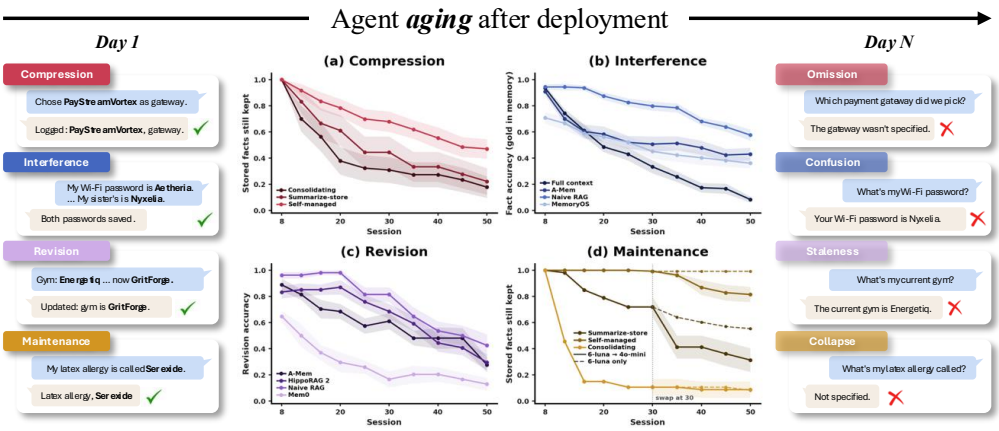


Figure 1: Four aging mechanisms after deployment. (a) *Compression*: fact retention erodes over sessions as memory rewrites itself. (b) *Interference*: cross-domain recall collapses as accumulated state crowds out the target entries. (c) *Revision*: derived-value error compounds across probes when a missed update contaminates downstream answers. (d) *Maintenance*: a routine operational event (e.g., replacing the agent internal model) produces divergent recoveries with varied shock types.

similar entry has taken its place, or an obsolete fact remains active. Yet unlike databases, which have external analysis tools [34], and software, which has regression test suites [14], long-lived agents lack an established way to measure and diagnose how their reliability degrades after deployment.

Recent memory benchmarks [15, 46, 18, 6, 30, 26] have begun to study long-context and multi-session memory and show that agent performance can degrade as context grows. Most of them, however, still report reliability as an end-to-end score, with little control over how facts depend on, supersede, or compete with facts from earlier sessions, and without localizing where in the agent a failure arises (§2). Evaluating long-lived agents as *evolving* systems needs more than a score. We refer to this problem space as **agent lifespan engineering** (ALE): measuring, diagnosing, and repairing degradation in long-running agent systems. ALE asks three questions:

> *what kind* of degradation is agent aging ***over time***? *how* do routine operational events shape it? And *where* in the agent's memory pipeline did it originate?

We introduce AGINGBENCH, a longitudinal reliability benchmark that moves beyond measuring *whether* agents degrade to diagnosing *how*. We organize agent aging into four mechanisms (refer to Figure 1): *compression aging* (information lost as memory is written and rewritten), *interference aging* (relevant facts buried behind similar entries as state grows), *revision aging* (failure to track changing or derived state), and *maintenance aging* (silent regression after operational events, such as model swapping). To make aging measurable, we introduce a *temporal dependency DAG* (Directed Acyclic Graph) that encodes the cross-session structure: competing content accumulates, values are revised over time, and answers require chaining facts written sessions apart. Mechanism-specific metrics are then computed based on probes asked at checkpoints. To simulate longitudinal deployment, all scenarios are backed by *programmatic generators* that scale to multiple sessions, providing a controlled pressure surface for isolating failures that are hard to disentangle in production traces.

A deployed agent is not merely a model but a *harness*: a language model coupled with a memory pipeline [7, 38, 13, 20, 24, 10] for writing, storing, retrieving, and utilizing information. When the agent fails, "the memory is getting worse" is not actionable; knowing *which component* is responsible is. We therefore build counterfactual conditions into the evaluation: replacing retrieval with an oracle over the agent's own store, and replacing both write and retrieval with the evidence from the history itself. The resulting profiles point to the stage a repair should target (write, retrieval, or utilization), rather than providing a unique causal decomposition for every architecture.

Across 6 scenarios, 10 memory systems, and 6 models from two families, our experiments (§6) show that agent aging is not one-dimensional. A well-formed reply can hide a value the memory has lost; derived state collapses as updates accumulate, even when the full history is in context; replacing the model that maintains a memory makes it lose facts it had kept, while retrieval-based and agentic stores are left nearly untouched; and a stronger model repairs only the stages it performs. The same

surface symptom, "the agent is wrong," can therefore point to different failing stages that call for different repairs. Our contributions can be summarized as follows:

- **A four-mechanism aging taxonomy** that maps compression, interference, revision, and maintenance to the lifecycle of agent deployment, each with dedicated metrics for auditing (§3).
- **AGINGBENCH**, a benchmark suite of diverse practical scenarios with programmatic generation and temporal dependency, which support both controlled and autonomous agent evaluation (§4).
- **Component-level attribution** via a configurable evaluation harness that supports counterfactual diagnosis, narrowing "the agent forgot" to a specific memory pipeline component: write-time omission, retrieval failure, or utilization failure, identifying the underlying reason of aging (§5).
- **Empirical findings** across all four mechanisms showing that agent aging is not one-dimensional: it can hide behind well-formed replies, is sharp for derived state, is triggered by lifecycle events, and depends on the model interaction with its memory architecture (§6).

2 Related Work

Existing work increasingly studies multi-session memory and long-horizon capabilities of AI agents; AGINGBENCH differs by treating *longitudinal degradation itself* as the evaluation object, instrumented through aging curves, a temporal dependency DAG, lifecycle event injection, and component-aware diagnostic profiles. We expand this comparison with detailed discussion in Appendix A.

Aging in deployed agents. In practice, long-lived agents face pressures that no snapshot benchmark captures. A coding agent that compresses months of project context into a fixed-size summary inevitably loses low-frequency details like specific API versions or configuration values [12]. An enterprise assistant managing multiple clients can retrieve the wrong client's budget when similar entries accumulate in its memory store [51]. A personal planner that once tracked a user's dietary restriction fails to update when the user lifts it, continuing to enforce an obsolete constraint [40]. And a production agent that behaves reliably for weeks silently regresses after a memory recompaction [39]. Complementary to other benchmarks that evolve the external target (e.g., codebase evolution [8]), our work measures degradation of the agent's internal memory state, with component attribution. On the memory systems side, some works [50, 43] characterize compression as a bottleneck but do not measure how it degrades agent reliability, nor do they track the full range of deployment pressures.

Lifecycle events and attribution for system harnessing. Few existing benchmarks (we summarized in Table 4) treat operational events as controlled experimental conditions, and generally assumes a static evaluation environment; the agent memory does not evolve during the benchmark run. Yet deployed agents routinely undergo such events like memory compacting or flush [17], and their impact on reliability is rarely measured. Similarly, failure attribution remains largely unaddressed: existing benchmarks report end-to-end scores without localizing whether the failure originated at write time, during retrieval, or at utilization. TierMem [50] partially addresses this by distinguishing summary-caused omissions from reasoning failures, but does not provide a general counterfactual framework. Our approach adapts counterfactual analysis to inspect the failure of long-lived agents.

3 Aging Taxonomy in Agent Lifelong Deployment

We organize agent aging into four mechanisms (Figure 1). Conceptually, they fall into two families. *Accumulation-driven aging* (compression, interference) worsens as the agent's state grows over sessions; it is the cost of operating over time, though discrete spikes (e.g., a file overwrite) can punctuate the trend. *Event-driven aging* (revision, maintenance) is triggered by discrete changes in the environment or agent itself; it is the cost of operating in a world that does not stand still.

- **Compression aging** arises from the *write-before-query barrier*: memory systems must decide what to preserve at write time, but which facts matter depends on future queries that have not yet arrived [50, 43, 46]. Summarization, consolidation, fact extraction, and self-managed notes can drop or blur specific values and every further rewrite as the history grows can lose more.
- **Interference aging** arises even when no information is lost *and no facts have changed*: as state grows, similar content accumulates (other owners' values, same-topic rounds, look-alike chains, near-miss items) and crowds the evidence out of what the reader sees, or leads it to answer with a competitor [25]. Interference is orthogonal to revision (freezing all facts does not prevent it).

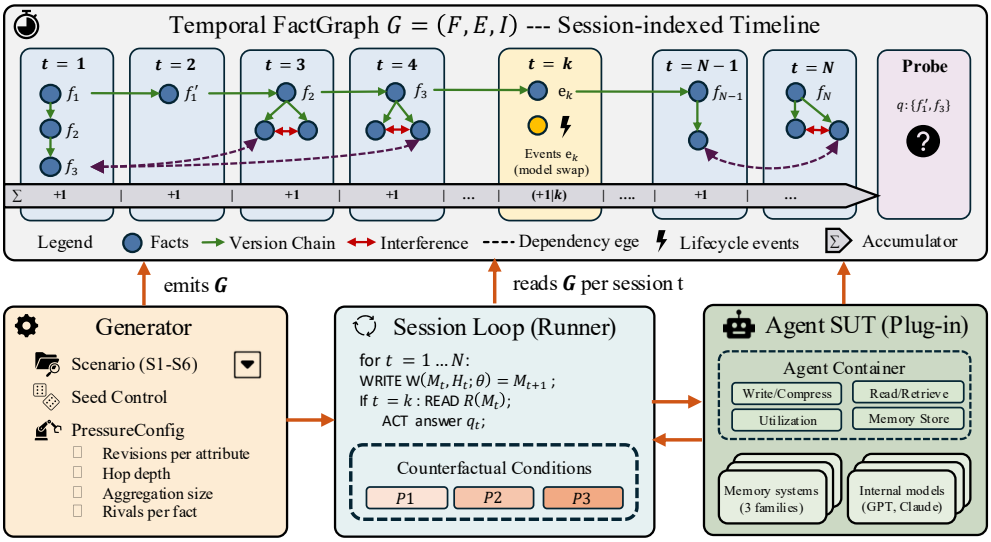


Figure 2: AgingBench evaluation pipeline. *Top*: temporal FactGraph as a session-indexed timeline, with version chains, interference rivals, dependency edges, aggregation Σ, and probes. *Bottom*: generator emits G and the probe stream; session loop writes every session and probe at checkpoints, with counterfactual conditions; the SUT plugs in a memory system and its internal model.

- **Revision aging** occurs when a value changes and the agent answers with a superseded version. A harder form is *derived state*, such as what a value was before or how many times it changed, which must be recomputed from the whole update history.
- **Maintenance aging** occurs when lifecycle events, such as replacing the model that maintains the memory or flushing and migrating stored memory [36], remove knowledge the agent already had. Unlike the other three mechanisms, it is driven by actions taken *on* the agent.

Deployment scenarios. Different agent deployments naturally encounter different subsets of these mechanisms. A research literature agent accumulates paper summaries over months; published findings rarely change, so its risk lies in losing the details of older papers and confusing similar results. A lifestyle assistant tracks a user's budgets, diet, and schedule, which change over time, so an outdated preference can look as valid as the current one. An enterprise knowledge base spans many similar projects whose decisions keep shifting, so the answer for one project can be crowded out by another project's or by an earlier decision. A production agent also goes through operational events such as routine model rotations, which can remove what it already knew regardless of its domain. Full archetype mapping are discussed in Appendix C.1. All four mechanisms can co-occur over an agent's operational lifetime, and the four-way split matters as the same surface symptom "the agent is wrong," needs different interpretations depending on which mechanism is binding.

4 AGINGBENCH : Longitudinal Reliability Benchmark

Making the four aging mechanisms from §3 measurable requires an evaluation framework that can simulate multi-session deployment, encode cross-session dependencies, and scale to long operational lifetimes. We describe the generation framework that produces cross-session timeline structure at varied scale (§4.1) and the evaluation procedure with findings preview (§4.2), in our AGINGBENCH.

4.1 Timeline Generation with Temporal Structure

In real deployment, facts accumulate across sessions, supersede each other, and compete for retrieval. Capturing this structure is essential for making aging measurable: without it, the evaluation cannot distinguish a lost fact from a crowded-out or a stale one.

Temporal dependency DAG. To encode this cross-session structure, the generator produces a DAG $G = (\mathcal{F}, \mathcal{E}, \mathcal{I})$ for each timeline, in which every planted round carries the session that writes it and the answer key of each checkpoint is computed by graph queries. *Facts* $\mathcal{F}$ are (subject, relation, value) triples, and *version chains* track supersession within $\mathcal{F}$: when a value is

revised, $f_i^{(v)} \rightarrow f_i^{(v+1)}$ extends a chain, and the gold at a checkpoint is the latest version written before it. *Dependency edges* $\mathcal{E}$ link a probe to the facts its answer requires, either a multi-hop chain of depth d or the members of an aggregation. *Interference rounds* $\mathcal{I}$ attach competitors to a probe: other owners' values for the same attribute, same-topic rounds without a value, look-alike chains, and near-miss items. These structures generate four probe families as listed in Table 1. *Lookup* asks for one fact in $\mathcal{F}$, with its competitors written later. *Compose* follows a multi-hop chain in $\mathcal{E}$ to its second or later hop, with a look-alike chain from another root. *Update* asks for the current value of a version chain, alongside never-updated controls of the same question form. *Count* aggregates over members in $\mathcal{E}$, with near-miss items and same-topic rounds. Figure 3 illustrates these structures and the pressure they apply.

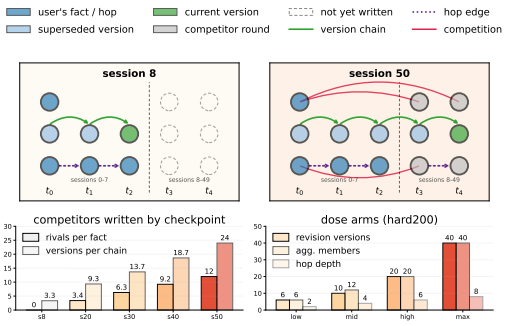


Figure 3: Temporal DAG and pressure control. Top: one timeline at sessions 8 and 50, with the user's facts, version chains, and competitor rounds. Bottom left: competitors written before each checkpoint in the 50-session builds. Bottom right: the dose arms of the 200-session builds.

Scalable programmatic generation. Measuring aging over long operational lifetimes requires timelines that scale without manual authoring. AGINGBENCH separates authoring from generation. An offline, LLM-assisted step builds a frozen library of content units for each scenario in Table 1 and repairs it against an item-level audit (Appendix C.5). Generation is deterministic then: it binds units to timelines without sharing a unit across timelines, schedules every edge, and inserts each one as a round into scenario-themed base conversations, so a configuration reproduces its build byte for byte. Timelines scale to about 240 sessions with the current pools, and pressure knobs (timeline length, competitors per fact, versions per attribute, hop depth, aggregation members) can be varied one at a time, binding the same units across arms where possible. More details can be found in the appendix: generator and pressure configuration (Appendix F.2), memory systems and prompts (Appendix F.3).

Table 1: Scenario design and probe profile. All scenarios share one planted dose and differ in domain content. Symbols rank each probe family's accuracy drop across scenarios (● high, ◐ medium, ○ low). Concrete task examples are provided in Appendix C; the scenario summary is in Appendix C.3.

Scenario	Domain	Lookup	Compose	Update	Count	Words / session
S1 Research	Papers + experiments	◐	◐	○	●	1972
S2 Lifestyle	Budget, diet + schedule	◐	○	●	◐	1702
S3 Knowledge base	Project decisions	●	●	●	○	1805
S4 Software	Code planning	●	●	◐	●	2030
S5 Notebook	Personal + project notes	○	◐	○	◐	1801
S6 Multi-domain	Travel, shopping + email	○	○	◐	○	1695

4.2 Evaluation Procedure and Aging Preview

We formalize agent aging evaluation as a *session loop* over the N sessions of a generated timeline (Figure 2), with each memory system plugged in as its own U. At each checkpoint session t, the agent reads its memory M_t without changing it, answers the held-out probes q_t, and receives a rule-based accuracy score s_t. Every session's dialogue H_t is written into the next state:

$$M_{t+1} = U(M_t, H_t; \theta) \tag{1}$$

where U is the memory system's write function ($\mathcal{W}$ in §5.1) and θ its parameters (internal model, prompts, and, where used, a word budget). At a designated maintenance session $t = k$, the runner injects a lifecycle event e_k that changes θ by replacing the internal model; a paired run without the event serves as the control, enabling controlled measurement. The accuracy of a fixed probe set at each checkpoint forms the *aging curve* of a probe family. A key design principle is *gold-grounded scoring*: every probe is scored by rule against its answer key, with no LLM judge, and each loss is tied to one mechanism. Compression is the share of facts no longer in memory at the last checkpoint; interference is the accuracy drop on facts whose gold is still in memory; revision is the share of revised attributes answered with a superseded version; maintenance is the additional loss of stored facts caused by a lifecycle event, relative to the paired control. The full definitions are in Appendix B.

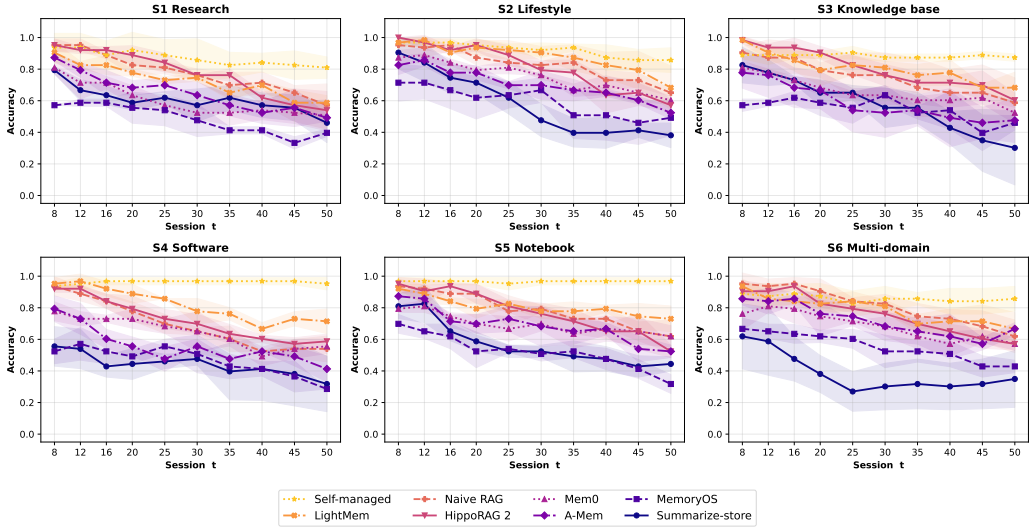


Figure 4: Overall aging curves measured on the mechanism probes within each scenario.

Aging Curve Preview. Figure 4 shows aging trajectories across all scenarios and memory systems: accuracy shows an overall downward trend over the horizon, with the rate and shape varying across memory systems. We leave more analysis to §6.2 for mechanism-level observations.

5 Component-Level Attribution

The aging taxonomy (§3) classifies *what kind* of degradation occurred; the benchmark (§4) measures *how much*. Attribution answers the question that the deployment engineer needs: *where in the memory pipeline* did the failure originate? This section develops a framework for attribution of aging. §5.1 defines a decomposition of the memory pipeline into explicit components that serve as the attribution targets, and §5.2 introduces a set of counterfactual interventions and diagnostic tools to detect and attribute aging to those components.

5.1 Memory Pipeline Decomposition and Failure Loci

To localize aging, we represent the deployed agent as a dataflow over a memory store and decompose it into explicit components (Figure 5).

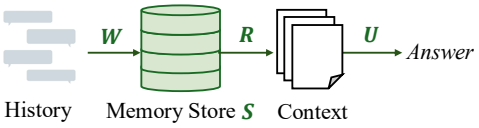


Figure 5: Memory Pipeline. Data flows sequentially as History $\xrightarrow{\mathcal{W}}$ $\mathcal{S}$ $\xrightarrow{\mathcal{R}}$ Context $\xrightarrow{\mathcal{U}}$ Answer. We attribute aging into highlighted components: **W, S, R,** and **U**

Write/Compress ($\mathcal{W}$) transforms each session into what the memory system stores. It is governed by the system's design and internal model, and can be lossy [21, 1, 47] (e.g., summarization, consolidation, fact extraction).

Read/Retrieval ($\mathcal{R}$) returns the working context for the current question: the query the reader issues and the system's retrieval algorithm (e.g., Last-k by recency [35, 23] or top-k cosine [52]). Injected memory in the prompt have no retrieval step.

Utilization Logic ($\mathcal{U}$) is the LLM model's reasoning over the context it receives: it tells the answer from its competitors and synthesizes the context into the response.

Memory Store ($\mathcal{S}$) is the persistent artifact written by $\mathcal{W}$ and read by $\mathcal{R}$; lifecycle events act on it. Each aging mechanism can have a primary stage in the pipeline: compression primarily stresses write ($\mathcal{W}$), interference stresses retrieval ($\mathcal{R}$), revision stresses utilization ($\mathcal{U}$), and maintenance stresses the store/lifecycle ($\mathcal{S}$)[2]. Localizing the error site provides guidance for mitigating aging.

[2]Maintenance (e.g., model swapping or flush) stems from external operations that directly modify the memory management.

5.2 Counterfactual Interventions and Diagnosis

After decomposing the memory pipeline into write, read, and utilization components, we use oracle-based counterfactual conditions to point to the failing stage.

Interventions. We perform component-level attribution using three counterfactual conditions on the same probes, timelines, and checkpoints as the main evaluation, summarized in Table 2. The conditions form an ablation ladder over the memory pipeline: each replaces selected upstream components with oracle implementations, and the resulting accuracy gaps identify the first non-oracle component that can still explain the failure. Let Acc_{Pi} denote the accuracy under condition Pi. $P1$ is the baseline execution condition: the agent uses its own write policy, retrieval procedure, and utilization logic, yielding Acc_{P1}. $P2$ replaces the agent's retrieval procedure with an oracle retriever while keeping the agent-written memory store fixed. Whatever the reader searches for, the oracle returns the store items that hold the probe's target or competitor values, yielding Acc_{P2}. Thus, $P2$ removes retrieval failures but still exposes any information that was omitted, corrupted, or underspecified by the write process. $P3$ replaces both write and retrieval with oracle context: the reader receives the rounds of the history that state the target and competitor values, yielding Acc_{P3}. Both oracles return the answer with its competitors, without indicating which is which, as an ideal retriever would; any remaining error under $P3$ is attributed to utilization, since the model is given the information needed to answer. Two further conditions that refine utilization are in Appendix D.1.

Diagnosis. From these conditions we derive a diagnostic profile over $\mathcal{R}$, $\mathcal{W}$, and $\mathcal{U}$. *Utilization error* is $1 - \text{Acc}_{P3}$: the evidence is in view but the answer is wrong. *Write error* is $\text{Acc}_{P3} - \text{Acc}_{P2}$: the answer is recoverable from the history but not from what the system stored, because it is missing or can no longer be used. *Read error* is $\text{Acc}_{P2} - \text{Acc}_{P1}$: the stored items suffice but do not reach the reader. Memories without a retrieval step do not run $P2$; there, $\text{Acc}_{P3} - \text{Acc}_{P1}$ is split per probe by whether the gold is in the memory text at $P1$: absent counts as a write error, present but missed as a read error. We read these profiles as counterfactual diagnosis under paired controls that points to a candidate stage, not as an exact causal decomposition.

Table 2: Diagnostic conditions. Each replaces more of the memory pipeline with an oracle.

	Write ($\mathcal{W}$)	**Read** ($\mathcal{R}$)	**Utilize** ($\mathcal{U}$)
P1 (baseline)	Agent	Agent	Agent
P2 (oracle retrieval)	Agent	Oracle	Agent
P3 (oracle context)	Oracle	Oracle	Agent

Maintenance Aging. A fact removed by a lifecycle event looks like a write error, so events are measured separately from the ladder: we compare a run with the event against a paired run without it, on the share of stored facts still in memory at the last checkpoint (Appendix B.2).

6 Results

In this section, we present main experimental results chaining into one structural thesis: *deployment-time memory aging is a property of the agent's interaction with its memory architecture*.

6.1 Experimental Setup

Agent Evaluated. We evaluate ten memory systems in three families: RAG (Naive RAG, HippoRAG 2 [13]), in-context memory (Self-managed, Summarize, Consolidating), and agentic memory (A-Mem [38], MemoryOS [20], LightMem [10], SimpleMem [24], Mem0 [7]), with full context as the reference. A system under test pairs a memory system with an internal model, which also reads the memory and answers the probes through a ReAct loop [42]. We mainly consider models from two families, GPT (gpt-4o-mini, gpt-5-nano, gpt-5.6-luna, gpt-6-luna) and Claude (claude-haiku-5.5).

Design Details. The main experiments use 50-session timelines (about 122K tokens) with ten checkpoints from session 8 to 50, on three timelines per scenario (18 per cell). For maintenance, the internal model is replaced at session 30 in our main experiments and compared against a paired run without the replacement. Stage attribution uses gpt-6-luna, and the pressure-knob experiments use 200-session timelines (about 444K tokens). Full setup details are in Appendix E.1.

Probe Form. The main experiments ask each probe as a direct question answered with a test value. This keeps the difficulty of a probe in what the memory supplies rather than in carrying out a task, as a strong reader given the full history answers nearly all of them (Appendix C.5), and lets every answer be scored by rule against the answer key without an LLM judge. A direct question, however, names

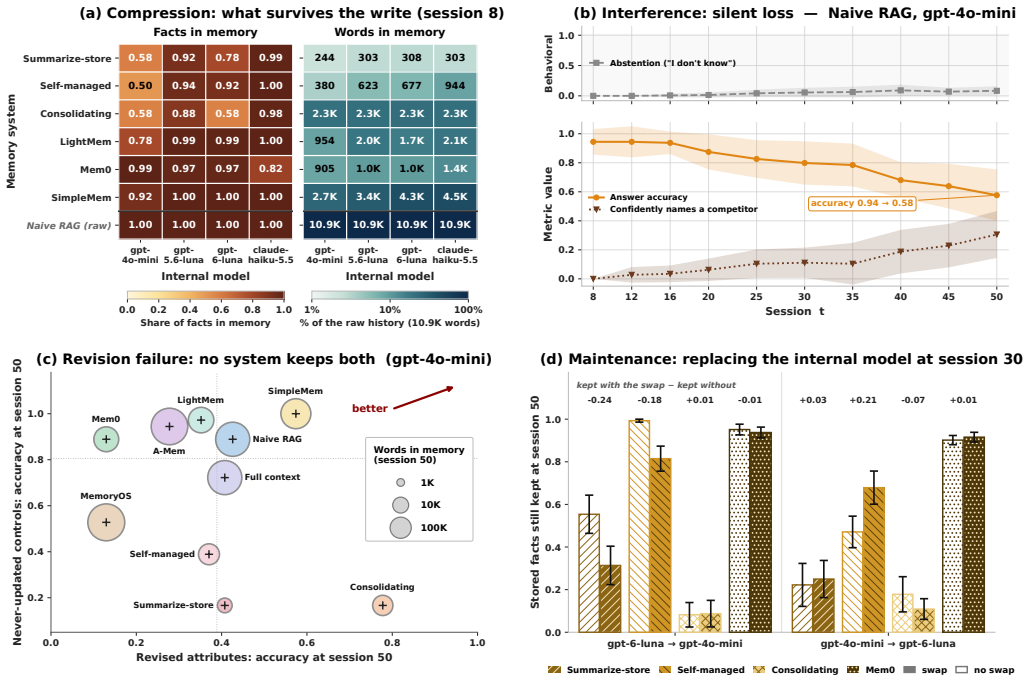


Figure 6: Mechanism-level findings. (a) *Compression*: facts in memory and words stored at session 8, the write-time part of compression. (b) *Interference*: Naive RAG keeps the raw history and fact values never change; the reader rarely abstains and increasingly names a competitor (gpt-4o-mini). (c) *Revision*: accuracy at session 50 on revised attributes against never-updated controls (gpt-4o-mini). (d) *Maintenance*: stored facts still kept at session 50 when the internal model is replaced at session 30, in both directions; the number above each pair is the change caused by the replacement.

what is being asked for, and a value-only reply discourages abstention, so the resulting accuracy is likely optimistic and the measured utilization error is a lower bound. We therefore also ask the probes as task requests, scored by whether the reply names the gold (Finding V; Appendix C.4).

6.2 Main Results

In this section, we discuss the main observations obtained from AGINGBENCH. Figure 6 shows one finding per mechanism, and Table 3 summarizes the aging profile by memory system and models.

***Finding* I: Aging is multi-dimensional; no memory system dominates across mechanisms.** Read as a whole, Table 3 shows no row that consistently dominates across mechanisms. A memory system that leads under one mechanism is often average or worst under another, and these rank reversals recur throughout the table rather than arising from isolated comparisons. The reversals appear to reflect how much a memory keeps. Stores that retain every round or extracted item lose few facts but tend to answer with a superseded version, while memories that rewrite themselves within a budget lose facts that never changed and keep losing them as the history grows; extraction-based memories, in contrast, lose theirs mostly at write time. Consequently, the choice of a memory system depends on which failure mechanism is most relevant to the target setting, rather than on a single notion of being "better." Aggregate scores [37, 29, 15, 46] may therefore obscure deployment-relevant behavior.

***Finding* II: Interference can degrade accuracy without a visible signal.** Naive RAG stores the raw history and its fact values never change, so its decline on fact probes is interference by construction (Figure 6b). As competing content accumulates, fact accuracy falls, yet the reader rarely abstains and increasingly names another owner's value. The agent continues to produce well-formed answers even though the evidence remains in its store, because the gold often no longer reaches the reader. In this regime, aging is difficult to detect: failures appear as wrong values rather than explicit refusals, so the replies alone may not reveal the degradation. The degree of silence varies with the model, and our value-only prompt discourages abstention, so this pattern should be read as indicative. Detecting it requires mechanism-level probes that test what the agent should still know.

***Finding* III: Revision aging depends on the architecture, and derived state fails even with the full history.** Revision aging does not appear to share one cause across memory architectures. In memories

Table 3: Mechanism-level aging profile of ten memory systems in three families. Each column group is one aging mechanism (§3), scored as a loss from session 8 to session 50 on S1–S6 (lower is better; each column is the system's internal model, which also answers the probes (top, GPT; bottom, Claude)). *RAG* retrieves raw passages from an index; *in-context memory* places the whole memory in the prompt; *agentic memory* has an LLM turn the dialogue into memory units at write time and retrieves those. **Bold**, the largest loss per column. No system has the lowest loss on all four mechanisms, and the maintenance loss of the rewriting in-context memories.

		Compression ↓		Interference ↓		Revision ↓		Maintenance ↓	
GPT	Memory system	4o-mini	6-luna	4o-mini	6-luna	4o-mini	6-luna	4o-mini	6-luna
RAG	Naive RAG	0.00	0.00	0.37	**0.37**	0.57	0.54	0.00	0.00
	HippoRAG 2	0.00	0.00	0.30	0.27	0.70	0.59	0.00	0.00
In-context memory	Self-managed	0.76	0.09	0.17	0.03	0.50	0.06	-0.21	0.18
	Summarize	0.88	0.56	0.15	0.21	0.56	0.13	-0.03	**0.24**
	Consolidating	**0.88**	**0.97**	0.16	0.17	0.19	0.04	**0.07**	-0.01
Agentic memory	A-Mem	0.00	0.00	**0.48**	0.36	0.72	0.74	0.00	0.00
	MemoryOS	0.00	0.00	0.35	0.22	**0.87**	**0.85**	0.00	0.00
	LightMem	0.22	0.01	0.31	0.23	0.65	0.52	0.00	0.00
	SimpleMem	0.08	0.00	0.29	0.11	0.43	0.09	0.00	0.00
	Mem0	0.10	0.08	0.29	0.18	**0.87**	0.35	-0.01	0.01
Claude		haiku-5.5	haiku-4.5	haiku-5.5	haiku-4.5	haiku-5.5	haiku-4.5	haiku-5.5	haiku-4.5
RAG	Naive RAG	0.00	0.00	**0.29**	0.35	0.56	0.69	0.00	0.00
	HippoRAG 2	0.00	0.00	0.20	0.25	0.52	0.56	0.00	0.00
In-context memory	Self-managed	0.01	0.17	0.03	0.09	0.04	0.08	0.00	-0.01
	Summarize	0.60	0.22	0.24	0.47	0.35	0.36	-0.05	0.22
	Consolidating	**0.88**	**0.30**	0.23	**0.59**	0.13	0.25	-0.04	**0.64**
Agentic memory	A-Mem	0.00	0.00	0.24	0.28	0.48	0.50	0.00	0.01
	MemoryOS	0.00	0.00	0.22	0.25	**0.81**	**0.78**	0.00	0.00
	LightMem	0.00	0.00	0.26	0.25	0.61	0.50	0.00	0.00
	SimpleMem	0.00	0.04	0.04	0.13	0.06	0.17	0.00	0.00
	Mem0	0.24	0.14	0.27	0.31	0.22	0.28	**0.03**	0.02

that keep every version, the stale rate in Table 3 stays high under every model and the current version often does not reach the reader, which places the failure at retrieval. Where the model itself writes the memory, as in self-managed memory and SimpleMem, the stale rate can fall considerably with a stronger model. Memories that rewrite themselves trade one mechanism for another: Consolidating tracks the latest value but loses the attributes that never change (Figure 6c). Derived state is harder still. Such state must be recomputed from the whole update history, which neither retrieval nor more context provides on its own. Reliable tracking of derived values may therefore require explicit state maintained at write time, rather than relying on larger models or better compression alone.

***Finding* IV: A stronger model only repairs the stages it performs.** Read across the internal models in Table 3 and the stage profiles of gpt-6-luna in Figure 7, the benefit of a stronger model depends on what the model does inside the harness. When it only reads, as in Naive RAG, the aging curve moves much less with the reader than under full context: the errors sit at retrieval, where the index decides which passages surface among a growing set of competitors, and a stronger reader cannot use what it is never shown. When the same model also writes and rewrites the memory, as in self-managed memory, it determines most of the aging, because its capacity now enters at write and maintenance time. Model replacement isolates this effect (Table 3 and Figure 6d). With the reader unchanged, swapping only the model that writes moves most rewriting memories toward the level the new model reaches on its own, while the RAG and agentic stores barely change. The shift follows the new model rather than its overall strength: a model that is stronger overall can lose more facts when

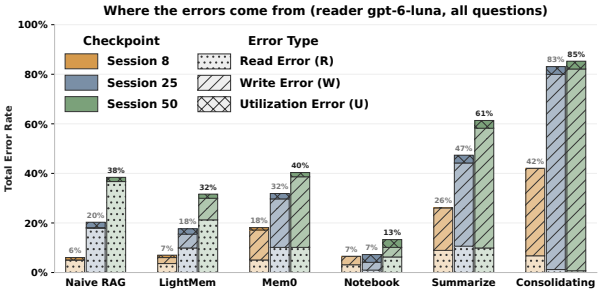


Figure 7: Aging attribution across different memory systems.

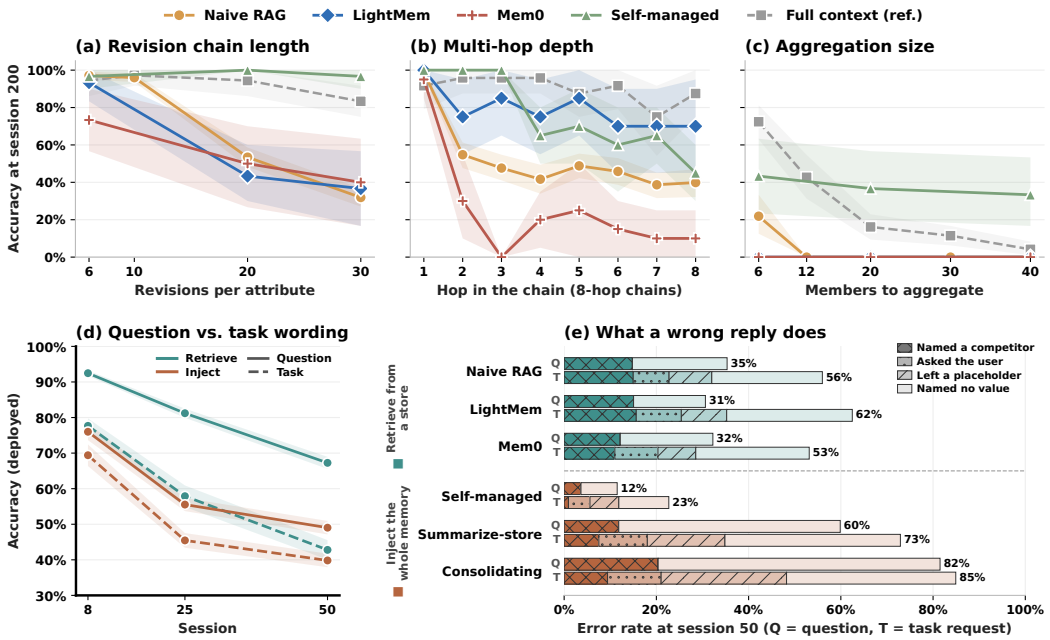


Figure 8: Aging under controlled pressure and task use. (a) *Revisions*: Naive RAG falls once the versions outnumber what a search returns, while self-managed memory holds. (b) *Hops*: Naive RAG drops at the second hop and then levels off. (c) *Aggregation*: retrieval-based memories barely count at any size, and even full context falls as the members grow. (d) *Question vs. task*: asking the same probes as a request that depends on the remembered value costs accuracy from the earliest sessions. (e) *Wrong replies*: most failed task replies still look complete, and only a minority asks the user.

it takes over a memory that rewrites itself, and a memory held to a tight budget ages under every model we tested. Aging therefore appears to follow the model's *role* in the harness.

***Finding* V: Question-style probes understate aging once memory must serve a task.** Memory benchmarks typically probe recall with a direct question that names what is being asked for. In deployment, the user more often asks the agent to *do* something that silently depends on a remembered value, e.g., drafting an email to "my gym." Asking the same probes in this task form lowers accuracy across the memory systems and scenarios we tested (Figure 8d), while the requests stay answerable once the evidence is given, so most of the gap does not come from the wording alone. The extra loss lands on different stages for different architectures. Retrieval-based memories lose it at retrieval: an indirect request leaves the reader to reconstruct the query, and an underspecified query appears easily crowded out by competing entries, so the cost is present from the earliest sessions and, in most systems, grows with the history. In-context memories more often still hold the value but miss it while the reader is occupied with the task. These failures are also hard to see: the agent usually returns a well-formed draft, leaving a placeholder, omitting the value, or using a competitor, and only a minority of wrong replies asks the user (Figure 8e). Question-form scores are therefore best read as a rough upper bound on deployed reliability, and lifespan evaluation should probe memory through the tasks it is meant to serve.

7 Conclusion

Long-lived agents can degrade quietly after deployment, even when their model weights are frozen: as memory state drifts *across sessions*, reliability becomes a property of the full agent harness rather than of the model alone. AGINGBENCH measures this phenomenon by organizing agent aging into four mechanisms (compression, interference, revision, and maintenance), measured on generated timelines with temporal structure and diagnosed with counterfactual conditions that point to a candidate stage of the memory pipeline. Our results show that aging is not one-dimensional: it can hide behind well-formed replies, is sharp for derived state, and depends on the role the model plays in the memory pipeline. AGINGBENCH provides the shared vocabulary and diagnostic insights to help the community build agents that age gracefully across their full operational lifetime.

Acknowledgement

This work of Jianing Zhu and Zhangyang Wang was supported in part by the Schmidt Sciences Trustworthy AI Award. This research has been supported by computing support on the Vista GPU Cluster through the Center for Generative AI (CGAI), the Texas Advanced Computing Center (TACC), InfraAI Center at The University of Texas at Austin and National Science Foundation under Grant No. 2326576. The authors declare no competing interests.

References

[1] Anthropic. Claude code. https://claude.ai, 2026. Accessed: 2026-04.

[2] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, 2025.

[3] Marcia J Bates. Indexing and access for digital libraries and the internet: Human, database, and domain factors. *Journal of the American Society for information science*, 49(13):1185–1205, 1998.

[4] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[6] Yuhao Chen, Yi Xu, Xinyun Ding, Xiang Fang, Shuochen Liu, Luxi Lin, Qingyu Zhang, Ya Li, Quan Liu, and Tong Xu. Vehiclemembench: An executable benchmark for multi-user long-term memory in in-vehicle agents. *arXiv preprint arXiv:2603.23840*, 2026.

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

[8] Gangda Deng, Zhaoling Chen, Zhongming Yu, Haoyang Fan, Yuhong Liu, Yuxin Yang, Dhruv Parikh, Rajgopal Kannan, Le Cong, Mengdi Wang, et al. Evoclaw: Evaluating ai agents on continuous software evolution. *arXiv preprint arXiv:2603.13428*, 2026.

[9] Ahmed Disouky, Mark A Sanborn, KR Sabitha, Mostafa M Mostafa, Ivan Alejandro Ayala, David A Bennett, Yisha Lu, Yi Zhou, C Dirk Keene, Sandra Weintraub, et al. Human hippocampal neurogenesis in adulthood, ageing and alzheimer's disease. *Nature*, 2026.

[10] Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. Lightmem: Lightweight and efficient memory-augmented generation. In *International Conference on Learning Representations*, volume 2026, pages 98706–98729, 2026.

[11] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

[12] Yuyao Ge, Lingrui Mei, Zenghao Duan, Tianhao Li, Yujia Zheng, Yiwei Wang, Lexin Wang, Jiayu Yao, Tianyu Liu, Yujun Cai, et al. A survey of vibe coding with large language models. *arXiv preprint arXiv:2510.12399*, 2025.

[13] Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*, 2025.

[14] Mary Jean Harrold, James A Jones, Tongyu Li, Donglin Liang, Alessandro Orso, Maikel Pennings, Saurabh Sinha, S Alexander Spoon, and Ashish Gujarathi. Regression test selection for java software. *ACM Sigplan Notices*, 36(11):312–326, 2001.

[15] Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, et al. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026.

[16] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[17] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025.

[18] Cheng Jiayang, Dongyu Ru, Lin Qiu, Yiyang Li, Xuezhi Cao, Yangqiu Song, and Xunliang Cai. Amemgym: Interactive memory benchmarking for assistants in long-horizon conversations. *arXiv preprint arXiv:2603.01966*, 2026.

[19] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[20] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 25972–25981, 2025.

[21] Lingavasan Suresh Kumar, Yang Ba, and Rong Pan. Memarchitect: A policy driven memory governance layer. *arXiv preprint arXiv:2603.18330*, 2026.

[22] Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *The Fourteenth International Conference on Learning Representations*, 2026.

[23] Xuechen Liang, Meiling Tao, Yinghui Xia, Jianhui Wang, Kun Li, Yijin Wang, Yangfan He, Jingsong Yang, Tianyu Shi, Yuantao Wang, et al. Sage: Self-evolving agents with reflective and memory-augmented abilities. *Neurocomputing*, 647:130470, 2025.

[24] Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*, 2026.

[25] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

[26] Shuochen Liu, Junyi Zhu, Long Shu, Junda Lin, Yuhao Chen, Haotian Zhang, Chao Zhang, Derong Xu, Jia Li, Bo Tang, et al. Perma: Benchmarking personalized memory agents via event-driven preference and realistic task environments. *arXiv preprint arXiv:2603.23231*, 2026.

[27] Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*, 2025.

[28] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

[29] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, 2024.

[30] Praveen Kumar Myakala, Manan Agrawal, and Rahul Manche. Beliefshift: Benchmarking temporal belief consistency and opinion drift in llm agents. *arXiv preprint arXiv:2603.23848*, 2026.

[31] Ziheng Qin, Yaxin Lu, Zhangyang Atlas Wang, and Kai Wang. Statem: Reaching 95.3% raw accuracy, or a $15 frontier run, on terminal-bench 2.1 via harness scaling. *arXiv preprint arXiv:2608.15089*, 2026.

[32] Narayan Ramasubbu and Chris F Kemerer. Technical debt and the reliability of enterprise software systems: A competing risks analysis. *Management Science*, 62(5):1487–1510, 2016.

[33] Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D Nowak. The long-horizon task mirage? diagnosing where and why agentic systems break. *arXiv preprint arXiv:2604.11978*, 2026.

[34] Yifan Wang and Daisy Zhe Wang. Extensible database simulator for fast prototyping in-database algorithms. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 5029–5033, 2022.

[35] John T Wixted and Ebbe B Ebbesen. On the form of forgetting. *Psychological science*, 2(6):409–415, 1991.

[36] Chengwen Wu, Guangyan Zhang, and Keqin Li. Rethinking computer architectures and software systems for phase-change memory. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 12(4):1–40, 2016.

[37] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

[38] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

[39] Cheng Yang, Xuemeng Yang, Licheng Wen, Daocheng Fu, Jianbiao Mei, Rong Wu, Pinlong Cai, Yufan Shen, Nianchen Deng, Botian Shi, et al. Learning on the job: An experience-driven self-evolving agent for long-horizon tasks. *arXiv preprint arXiv:2510.08002*, 2025.

[40] Ke Yang, Zixi Chen, Xuan He, Jize Jiang, Michel Galley, Chenglong Wang, Jianfeng Gao, Jiawei Han, and ChengXiang Zhai. Plugmem: A task-agnostic plugin memory module for llm agents. *arXiv preprint arXiv:2603.03296*, 2026.

[41] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

[42] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

[43] Zhixing You, Jiachen Yuan, and Jason Cai. D-mem: A dual-process memory system for llm agents. *arXiv preprint arXiv:2603.18631*, 2026.

[44] Guancheng Zeng, Xueyi Chen, Jiawang Hu, Shaohua Qi, Yaxuan Mao, Zhantao Wang, Yifan Nie, Shuang Li, Qiuyang Feng, Pengxu Qiu, et al. Routine: A structural planning framework for llm agent system in enterprise. *arXiv preprint arXiv:2507.14447*, 2025.

[45] Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025.

[46] Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, et al. Ama-bench: Evaluating long-horizon memory for agentic applications. *arXiv preprint arXiv:2602.22769*, 2026.

[47] Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, et al. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026.

[48] Chenyang Zhu, Spencer Hong, Jingyu Wu, Kushal Chawla, Yuhui Tang, Youbing Yin, Nathan Wolfe, Erin Babinsky, and Daben Liu. Raffles: Reasoning-based attribution of faults for llm systems. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7659–7688, 2026.

[49] Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025.

[50] Qiming Zhu, Shunian Chen, Rui Yu, Zhehao Wu, and Benyou Wang. From lossy to verified: A provenance-aware tiered memory for agents. *arXiv preprint arXiv:2602.17913*, 2026.

[51] Shitong Zhu, Chenhao Fang, Derek Larson, Neel Reddy Pochareddy, Rajeev Rao, Sophie Zeng, Yanqing Peng, Wendy Summer, Alex Goncalves, Arya Pudota, et al. Compliance brain assistant: Conversational agentic ai for assisting compliance tasks in enterprise environments. *arXiv preprint arXiv:2507.17289*, 2025.

[52] Shiwei Zhu, Junjie Wu, Hui Xiong, and Guoping Xia. Scaling up top-k cosine similarity search. *Data & Knowledge Engineering*, 70(1):60–83, 2011.

Appendix

A Extended Related Work **16**

B Metric Definitions and Scoring **19**

B.1 Probe Scoring 19

B.2 Mechanism Losses 19

C Scenario Details and Task Illustrations **21**

C.1 Scenario Curation and Generator Rationale 21

C.2 Session Anatomy: Timeline and Reader Input 22

C.3 Per-Scenario Examples and Summary 22

C.4 Task Form 24

C.5 Quality Checks of the Generated Items 24

D Component-Aware Diagnosis: Conditions and Memory Architectures **25**

D.1 Diagnostic Conditions 25

D.2 Stage Errors 25

D.3 Memory Architectures and Calibration 25

E Additional Experimental Results **26**

E.1 Experimental Setup 26

E.2 Variation over Timelines 26

E.3 Supplementary Evidence 26

F Implementation Details **27**

F.1 Running an Experiment 27

F.2 Generator and Pressure Configuration 28

F.3 Memory Systems and Prompts 29

F.4 Cost of Experiments 31

G Case Studies **32**

G.1 Same Wrong Answer, Different Repairs 32

G.2 A Stored Fact Lost by an In-Context Memory 33

H Evaluation Card **34**

I Broader Discussion **35**

Appendix

This appendix supplements detailed material about our exploration on long-lived AI agent aging (as illustrated in Figure 9). Appendix A extends the related-work discussion. Appendix B defines how probes are scored and how the aging curves and mechanism losses are computed. Appendix C describes the scenarios, the anatomy of a timeline and of the reader's input, examples of every probe family, the task form, and the quality checks of the generated items. Appendix D details the diagnostic conditions and how they adapt to each memory architecture. Appendix E gives the experimental setup and additional results. Appendix F covers running an experiment, the generator's configuration, the memory systems and their prompts, and the cost. Appendix G presents two case studies. Appendix H provides an evaluation card summarizing what AGINGBENCH measures, the scope of its attribution claims, and intended use. Appendix I discusses broader implications and limitations.

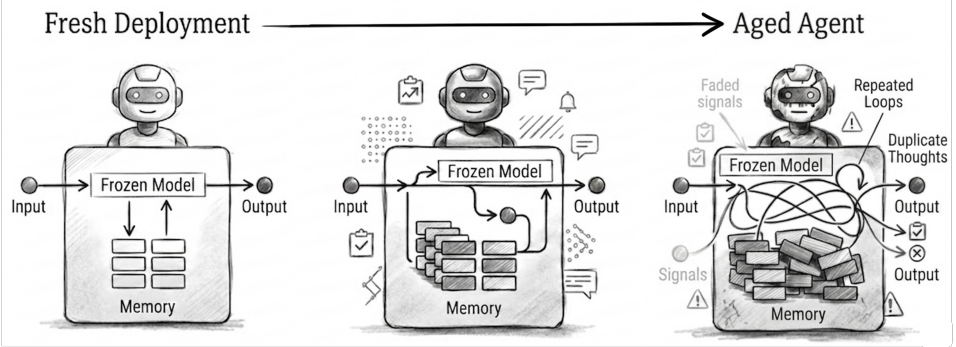


Figure 9: A conceptual illustration of long-lived agent aging after deployment.

A Extended Related Work

This section extends the related-work discussion of §2 along five dimensions that characterize longitudinal-aging evaluation: multi-session evaluation, cross-session dependencies, lifecycle events, measurable aging, and component-aware diagnosis.

Memory capability versus longitudinal reliability. Existing agent memory benchmarks predominantly evaluate *capability* at a given evaluation point: how well an agent's memory supports a task at one snapshot in time. These studies have surfaced a range of memory-degradation patterns (e.g., performance dropping as context grows). We pursue a complementary question: how memory-supported behavior changes across the agent's operational lifetime. The five dimensions below characterize this longitudinal-reliability axis; the full benchmark landscape is consolidated in Table 4.

From long-context to multi-session evaluation. Long-context benchmarks (RULER [16], LongBench [2]) evaluate a model's ability to attend over a single growing context window. Increasing the context window often absorbs the difficulty, with degradation largely characterized along a single "long-context" axis. A related line of work [22] reports performance loss *within* a single underspecified multi-turn conversation; we focus on a different axis, where the conversation ends per session and the memory state evolves across many subsequent sessions. Harness scaling [31] also works within a single long-running task, improving the execution system around a fixed model. Multi-session evaluation *through a memory system* introduces a qualitatively different challenge: the agent does not see the raw history; it sees only what the memory system wrote and what it retrieves for the current query. The bottleneck shifts from attention span to the write→store→read pipeline of the memory system, which is what makes the four aging mechanisms distinguishable as failure modes. However, most existing evaluations treat sessions independently: each session's score is computed without reference to what the agent remembered or forgot from prior sessions.

From independent sessions to cross-session dependencies. To diagnose *why* an agent fails, the evaluation must encode how facts relate across sessions: which facts supersede which, which probes require multi-session synthesis, and which entities are confusable. MemoryArena [15] partially addresses this by requiring later subtasks to depend on earlier ones; LoCoMo [29] and PERMA [26] test temporal reasoning through questions that reference prior conversations; LongMemEval [37]

Table 4: Evaluation benchmark landscape. Design parameters shape what each benchmark can measure; capability columns mark which of the five longitudinal-aging dimensions are supported. *Aging curves*: performance on the same knowledge tracked as the interaction grows (sessions, periods or subtask depth), not a sweep over independent inputs of different length. *Lifecycle events*: operations injected on the agent or its memory during the run. TierMem is a memory system, listed for its failure attribution. ✓, *Partial*, and ✗ reflect emphasis rather than formal capability exclusion.

Category	Benchmark	Avg. length (tokens)	Sessions/ turns	Scalable	Multi-session	Cross-sess. dep.	Lifecycle events	Aging curves	Attribution
Long-context	RULER [16]	4K–128K	1	✓	✗	✗	✗	✗	✗
	LongBench v2 [2]	8K–2M words	1	✗	✗	✗	✗	✗	✗
Agent	τ-bench [41]	varies	1 (multi-turn)	✗	✗	✗	✗	✗	✗
Multi-session	LongMemEval [37]	∼115K	∼40–50 sess.	Partial	✓	Partial	✗	✗	✗
	LoCoMo [29]	∼9K	up to 35 sess.	✗	✓	Partial	✗	✗	✗
	MemoryArena [15]	14K–122K	2–16 subtasks	✗	✓	✓	✗	Partial	✗
	AMA-Bench [46]	∼57K	∼73 turns	✓	✗	✗	✗	✗	Partial
	AMemGym [18]	60K–140K	11–21 periods	✓	✓	Partial	✗	✓	Probe-based
	PERMA [26]	∼32K	∼80 events	Partial	✓	Partial	✗	Partial	✗
	BeliefShift [30]	∼57K	10–50 sess.	Partial	✓	✓	✗	✗	✗
	VehicleMemBench [6]	∼93K	30 event chains	Partial	✓	Partial	✗	✗	✗
Memory system	TierMem [50]	varies	N/A	✗	✗	✗	✗	✗	Partial
Longitudinal	**AgingBench (ours)**	**∼120K–450K**	**50–200 sess.**	✓	✓	✓	✓	✓	**Counter-factual**

includes knowledge-update and multi-session question types. However, to our knowledge no surveyed benchmark jointly encodes version chains (a value updated repeatedly, with the current version tracked as ground truth and a non-updated control chain), multi-hop chains (each hop written in a different session, with a look-alike distractor chain), and competing content (other owners' values for the same attribute and same-topic rounds without a value), each with a controllable amount. Without this structure, the scorer cannot distinguish whether a failure reflects information loss (compression), competition from similar content (interference), or a stale version (revision). AgingBench's temporal DAG (§4.1) addresses this gap by generating these structures programmatically, with the answer key of every checkpoint computed from the graph.

From static environments to lifecycle events. All benchmarks listed above assume a static evaluation environment: the agent's memory architecture does not change during the run. Yet deployed agents routinely undergo lifecycle events: the model behind the memory is upgraded or replaced, and stored memory is flushed or migrated. These events act on memory the agent has already built, so their cost shows up on knowledge written *before* the event. To our knowledge, no surveyed benchmark injects such events as controlled experimental conditions. AgingBench supports different maintenance event types (§3) injected at designated sessions, enabling controlled measurement of maintenance aging.

What existing benchmarks lack for measurable aging. Few benchmarks that evaluate across multiple sessions report *aging curves*: how performance on the same knowledge changes as the history behind it grows. LongMemEval [37] evaluates with a single Q&A over multi-session conversational history; the snapshot remains a single evaluation point. AMemGym [18] is the closest: it re-asks its evaluation questions after every interaction period, reports a decline as the interaction grows, and classifies each failure as a write, read, or utilization failure by querying the assistant about the user state it should have stored. TierMem [50] distinguishes summary omissions from reasoning failures, but does not track these signals over the agent's operational lifetime or provide counterfactual conditions for localizing the responsible component. The five capabilities discussed above (multi-session evaluation, cross-session dependencies, lifecycle event injection, aging curves, and component-aware diagnostic profiles) are each present in part of the prior literature, but to our knowledge, none of the benchmarks we surveyed integrates the joint combination within a single longitudinal evaluation framework; AGINGBENCH is designed to address this combination.

Trajectory-based attribution and judge methods. A complementary line of work attributes agent failures by analyzing rollout trajectories, typically with LLM-as-a-Judge pipelines: automated attribution over multi-agent traces [45], failure taxonomies validated by judges against human annotators [4]; AgentErrorTaxonomy [49]), reasoning-based attributors [48], and long-horizon trajectory diagnosis [33]. These methods scale cheaply and apply to any failure mode visible in the rollout, but fine-grained step-level attribution is empirically hard [45]. Our diagnostic framework is epistemically complementary: where judges *infer* the responsible step from a trajectory, our

conditions *intervene* on the memory pipeline, replacing retrieval or the stored content with an oracle, and produce a diagnostic profile. For memory-aging failures specifically, several failure modes are observationally *ambiguous* in a trajectory log without gold knowledge of what the agent *should* have written, motivating intervention as a stage-localizing probe. A principled composition would use trajectory attribution to localize memory-bound probes, and our counterfactuals to produce component-aware diagnostic profiles for the candidate stages.

Summary. Table 4 captures the landscape along two complementary axes: design parameters (context length, session count, scalability of the task generator) on the left, and capability presence on the five aging-evaluation dimensions on the right. Read together, the two halves substantiate our position: prior work covers each capability individually, but the joint combination under controlled longitudinal pressure is, to our knowledge, not present in the benchmarks we surveyed.

B Metric Definitions and Scoring

This section defines how each probe is scored, the probe families and the mechanism each one reads, the aging curves, and the four mechanism-level losses reported in Table 3. Every score is rule-based and gold-grounded: each probe carries its answer key from the temporal DAG, and no LLM judge is used at our framework. New metrics can be added by defining them over the same per-probe records.

B.1 Probe Scoring

A timeline τ is evaluated at checkpoints $c_1 < \cdots < c_K$ (in sessions). At checkpoint c, the memory holds the sessions written before c, and each probe p is answered against it with gold $g_p(c)$, read from the answer key of the build. The gold of most probes is the same at every checkpoint; for a revision probe it is the version current at c, and for a count probe it is the number of members written before c. Each timeline asks a fixed sample of eight fact probes (the same at every checkpoint) and all probes of the other families.

- **Correct** $x_p(c) \in \{0, 1\}$. Answers are value-only (a name or a number). Answer and gold are normalized (lower case, punctuation and digit separators removed, a leading "the" dropped); the answer is correct if it equals the gold, or contains it with at most two extra words (a title or a unit). For a count probe, the number stated in the answer must equal the gold.
- **Competitor named**. The answer is not correct and names a competitor that the DAG records for the probe: a rival's value for a fact, another version for a revision probe, or, for a count probe, a count above the gold (near-miss items counted as members).
- **Unknown**. The answer declines ("unknown", "not found", and similar).
- **Gold in memory** $\mathrm{in}_p(c) \in \{0, 1\}$. Whether the gold value occurs in the memory at c (for a count probe: every member). For memories placed in the prompt (Self-managed, Summarize, Consolidating) this is the memory text given to the reader; for the other memory systems, a scan of every item of the archived store at c; Naive RAG keeps the raw history, so $\mathrm{in}_p(c) = 1$.

Probe families. Each probe family is anchored to one structure of the temporal DAG, so a failure is localized to that structure rather than collapsed into a single recall number. Fact probes, whose competitors are other owners' values and same-topic rounds, read interference, as do the compositional families: multi-hop chains with a look-alike distractor chain, and counts with near-miss items. Revision probes, paired with a never-updated control of the same question form, read revision, as do derived-state probes that must be recomputed from the update history (the previous version, the number of versions, and counts that grow with the history).

B.2 Mechanism Losses

Table 3 reports one loss per mechanism (lower is better), computed per timeline on the sampled facts F_τ or the revised attributes V_τ and averaged over timelines.

Loss	Mechanism	Definition
missing at c_K	Compression	$1 - \frac{1}{\lvert F_\tau \rvert} \sum_{f \in F_\tau} \mathrm{in}_f(c_K)$: share of facts no longer in memory at the last checkpoint. It sums the facts lost when they are written (not in memory at c_1) and those lost later as the memory rewrites its contents.
crowding drop	Interference	$\mathrm{Acc}(c_1) - \mathrm{Acc}(c_K)$, where $\mathrm{Acc}(c)$ is the accuracy over $\{f \in F_\tau : \mathrm{in}_f(c) = 1\}$: the accuracy drop on facts whose gold is still in memory.
stale rate	Revision	$\frac{1}{\lvert V_\tau \rvert} \sum_{v \in V_\tau} \mathbb{1}[\text{competitor named}(v, c_K)]$: share of revised attributes answered at c_K with another version than the current one.
event loss	Maintenance	$\frac{1}{\lvert F_\tau^1 \rvert} \sum_{f \in F_\tau^1} \left(\mathrm{in}_f^{\mathrm{ctrl}}(c_K) - \mathrm{in}_f^{\mathrm{event}}(c_K)\right)$ over $F_\tau^1 = \{f \in F_\tau : \mathrm{in}_f(c_1) = 1\}$: of the facts in memory at the end of their writing phase, the share still in memory at c_K in the paired run without the event minus the share in the run with it (the event takes place at c_e, between c_1 and c_K).

Lifecycle event. The event is a rotation of the memory system's internal model: the store built by model A up to c_e is carried over unchanged, and model B performs every later write and reorganization; the reader stays A. The control run is the same timeline with A throughout. Systems without an internal model that writes the store (Naive RAG, full context) have no event.

The four losses read different quantities. Compression counts every fact the memory has lost by c_K, whether at write time or through its own later rewriting; maintenance counts only the additional loss attributable to the event, since the paired control already contains the compression; interference is measured only on facts whose gold is in memory; and revision counts only answers that name another version, so an attribute lost from memory is not counted as stale.

C Scenario Details and Task Illustrations

This section describes how the scenarios are curated (§C.1), what a timeline and the reader's input look like (§C.2), examples of every probe family on one S2 timeline with a summary of other scenarios (§C.3), the task form of the probes (§C.4), and the quality checks of generated items (§C.5).

C.1 Scenario Curation and Generator Rationale

Sources of scenario design. Each scenario (S1–S6) mirrors a long-lived assistant deployment archetype (Table 5). A scenario is defined by a persona, the range of content the user shares, and sixteen everyday subtopics; it sets the theme of the planted content and of the base conversations. The planted structure (facts with competitors, revision chains with controls, multi-hop chains with distractor chains, and aggregation units) is the same in every scenario, so scenarios differ in domain, not in which mechanisms they exercise. The base conversations are transposed from real LongMemEval-S sessions [37] into the scenario's subtopics, keeping each session's number and length of turns, and are rewritten so that the user states nothing about their own things that a probe could ask.

Table 5: Scenario design and supporting references. Each archetype is grounded in published evidence of deployed agent use cases or memory-system challenges; the planted structure is the same.

Scenario	Real-world archetype	Supporting references
S1 Research	An ML researcher's assistant tracking papers, experiments, collaborators and conferences	Memory-enabled agents [7, 27, 11]; compression as bottleneck [12]
S2 Lifestyle	A personal assistant tracking budgets, diet, preferred stores and the weekly schedule	Persona-state benchmarks [26, 30]; preference revision in deployment [40]
S3 Knowledge base	A project manager's record of meeting decisions: vendors, components, milestones, owners	Enterprise/routine agents [44]; compliance-driven memory [51]
S4 Software	A developer planning changes to an evolving Python codebase: modules, tools, data models, tests	Code-agent capability [19, 5, 28]; codebase evolution [8]
S5 Notebook	An assistant keeping a notebook of the user's personal (health, providers), project and code facts	Agent memory systems [7]
S6 Multi-domain	An assistant for travel, online shopping, calendar and email	Long-form conversational memory [29, 37]; multi-domain settings [6]

Generation beyond a fixed curated dataset. A generator-based approach supports four aging-measurement requirements simultaneously. (i) *Arbitrary session counts*: aging curves require evaluation at deployment-horizon scale; the same configuration can be built at any length up to the size of the scenario's base-conversation pool, and longer pools can be authored. (ii) *Reproducible builds*: a configuration reproduces its build byte for byte, and each build holds many timelines with disjoint content units, so statistics are computed over independent timelines. (iii) *Controlled pressure sweeps*: each pressure knob (Appendix F.2) changes one quantity, such as the number of rivals, versions, hops or aggregation members, while the other content stays fixed, and dose arms can bind the same units so that probes are paired across doses. (iv) *Gold ground truth for mechanism-specific scoring*: each probe carries its answer key from the temporal DAG, so scoring is rule-based.

The generator produces histories that exercise the four aging mechanisms under controlled conditions; it does not claim to capture the full distribution of real-world user behavior. This trade-off (*controlled longitudinal pressure* over *naturalistic distribution*) is by design: aging is plausibly hard to measure in noisy production traces, since the longitudinal signal is entangled with everything else, so a synthetic-yet-mechanism-faithful generator gives us a controlled measurement surface. Combining AGINGBENCH with production telemetry to verify that the same mechanisms compound at real timescales is an open frontier (§I).

Extending the scenario set. A new scenario is one specification file (persona, content range and subtopics); one authoring command builds its content library and base-conversation pool and repairs them against the checks of §C.5.

C.2 Session Anatomy: Timeline and Reader Input

A timeline is a sequence of dated sessions. Each session is a base conversation of the scenario into which the planted rounds of that session are inserted at turn boundaries; a planted round is one user turn with an assistant reply, either a short acknowledgement or a reply written with the unit. The memory system receives the sessions in order, without any marking of which rounds are planted. The excerpt below is session 1 of an S2 timeline.

S2 timeline, session 1 (2023/01/02 14:00), excerpt; planted rounds marked here only

user: I need help creating a multi-week meal plan that fits within a budget. I have some dietary restrictions to consider, and I want to make sure it includes easy-to-prepare meals. [. . .]
assistant: Absolutely! Please share any dietary restrictions, store preferences, and a general budget. [. . .]
user: Hey, do you know anything about Fintrac? *[hop 2, intro]*
assistant: That's helpful to know.
user: I heard they're developed by Zenthix. *[hop 2, value]*
assistant: That's helpful to know.
user: I'm trying to reserve a study room at the library for my study group this weekend, and I believe they need my library membership number, which is 46912. Could you help me with the process? *[fact]*
assistant: To reserve a study room at the library, you'll usually need to log into the library's online portal using your library membership number, which you mentioned is 46912. [. . .]
user: Harvest Delights was a great choice as a vendor for the Urban Harvest Festival. *[aggregation member]*
assistant: Yes, Harvest Delights is noted as a vendor for the Urban Harvest Festival.
[. . .]
user: That sounds great! Can you also help me with portion sizes and any alternatives I might need for certain ingredients based on any allergies? [. . .]

At each checkpoint the reader receives the probe as `Current date: <checkpoint date>` followed by `Question: <probe>`. For memory systems with a search step, the reader is the ReAct loop below, whose only tool returns the system's own retrieval results (its top 10 where the system takes a result count); Naive RAG returns rounds in reading order with their dates.

Reader, Thought/Action format

You are a personal assistant. You can look things up in the memory of your past conversations with the user, using a search tool that returns the most relevant past conversation rounds with their dates.

Work step by step. At each step write:
Thought: <your reasoning>
Action: search[<query>] or Action: answer[<final answer>]

After a search you will receive an Observation. You may search at most 5 times. Some questions need several lookups: use what one search finds to decide the next query. If information was updated, use the most recent version. Answer concisely.

The final answer must be ONLY the requested name or number (for a 'how many' question, only the number): no sentence, no explanation, no list of candidates.

Models called through an OpenAI-compatible router receive the same instructions with a one-line `Action:` format instead of *Thought* + *Action*. A reader that answers before any search is told once to search first. Memories placed in the prompt (Self-managed, Summarize, Consolidating) and the full-context baseline answer in one call under the same answer rules; these prompts and the memory systems' own prompts are given in Appendix F.3.

C.3 Per-Scenario Examples and Summary

S2: one timeline, every probe family. The examples below come from the same S2 timeline as the excerpt above (200 sessions). Each probe lists the rounds it depends on (session number) and its competitors; "gold @s" is the answer key for a reader holding the history up to session s.

Fact probe (interference)

Q: Can you remind me of the number for my library account? **gold:** 46912 at every checkpoint

Evidence: [1] I'm trying to reserve a study room at the library [. . .] my library membership number, which is 46912.

Rivals (12, sessions 14–196), e.g.: [14] My uncle's library membership number is 98723 [. . .]; [61] My cousin Alex's library membership number is 55834 [. . .]

Same-topic rounds without a value, e.g.: [32] I heard the library is hosting a book fair soon. Do you know the details?

Revision probe and its control (revision)

Q: What is my current coworking space? **gold:** @8 ZenCubicle; @100 WhimsyHaven; @200 OrbitHaven

Versions (40, evenly spread), e.g.: [2] I've switched to a new coworking space called QuirkHub. [7] Now I'm working from ZenCubicle as my coworking space. [. . .] [197] OrbitHaven is my new coworking space destination.

Control Q: Who is my current nutritionist? **gold:** Zorbina at every checkpoint

Only version: [5] My nutritionist has changed to Zorbina recently.

Multi-hop probe, depth 3 (interference, compositional)

Q: Where is the headquarters of the company that developed my budget tracking app? **gold:** Noveria City

User's chain: [0] My budget tracking app is called Fintrac. [1] Hey, do you know anything about Fintrac? / I heard they're developed by Zenthix. [2] Hey, do you know where Zenthix is based? / I've heard it's located in Noveria City.

Distractor chain (another owner): [80] Sophia's investment management tool is called WealthMaster. [125] [. . .] It was developed by a company named Finovatech. [197] [. . .] Yeah, they're located in Riverton City.

Aggregation probe (interference, compositional)

Q: How many vendors did we partner with for the Urban Harvest Festival? **gold:** 6

Members (6, sessions 0–6), e.g.: [0] GreenLeaf Grocers is one of the vendors we partnered with for the Urban Harvest Festival.

Near misses (6), e.g.: [80] Organic Eden was considered as a vendor, but we didn't end up partnering with them for the Urban Harvest Festival. [127] We partnered with Sunset Produce for the Coastal Wellness Expo.

Same-topic rounds naming no member (24), e.g.: [15] How are the Urban Harvest Festival vendors handling the payment processes?

Scenario summary. Table 6 lists the six scenarios with one fact probe each. Every timeline of every scenario carries the same planted structure and the same pressure configuration.

Table 6: The six scenarios. Timelines: number of timelines per build (each build of a scenario holds the same number); every timeline of a build uses content units no other timeline uses.

Scenario	Persona	Example fact probe (gold)	Timelines
S1 Research	an ML researcher keeping track of papers, experiments, collaborators and conferences	Could you remind me which special interest group I'm involved with? (QuantumNeuron Nexus)	17
S2 Lifestyle	budgets, diet, preferred stores and weekly schedule	Can you remind me which local restaurant I favor the most? (The Green Orchid)	19
S3 Knowledge base	a project manager recording decisions from team meetings	Could you remind me of the numerical identifier for our main software component? (46329)	20
S4 Software	a developer planning changes to an evolving Python codebase	What do we call our logging module in the codebase? (LogChamp)	20
S5 Notebook	personal (health, providers), project and code facts kept in a notebook	Which hospital do I prefer for my treatments? (Meadowfield Medical Center)	20
S6 Multi-domain	travel, online shopping, calendar and email	What's the name of the car rental company I typically use when traveling? (SwiftWheel Rentals)	20

C.4 Task Form

Besides the question form, a probe can be asked as a request that needs its value, for example:

Task form of the revision control above

Question form: Who is my current nutritionist?
Task form: Please draft an email to my current nutritionist to discuss adjusting my meal plan for the upcoming month, focusing on incorporating more allergen-free snacks.

The history and the answer key are unchanged; only the wording of the probe changes. The reader completes the request and its whole reply is scored by rule: it is correct if it names the gold (for an aggregation, every member), and a placeholder such as "[name]" does not count; a reply that names a competitor is recorded as such. Tasks are written per scenario and kept only if they pass rule checks and an oracle check (given every relevant planted round, a reader must name the gold); probes without a task are asked only in the question form.

C.5 Quality Checks of the Generated Items

A scalable generator needs checks on the items it produces. Every build is audited at generation time: each gold value occurs only in its own planted rounds, units that could answer each other's probes never share a timeline, and the questions of different units stay below a similarity bound. The content libraries were repaired during authoring against an item-level audit (e.g., a rival that is in fact the user's own value, near-synonym attributes in one history, evidence too weak to support the question); the repairs are listed in the released documentation.

Three further checks estimate what remains (Table 7). *Full-context ceiling*: a strong reader (gpt-6-sol) given the whole history answers the probes of every timeline (eight sampled facts per timeline, all other probes) at every checkpoint. *Item audit*: for each probe of the evaluated timelines, a judge (gpt-6-sol) reads every round of the timeline that could bear on the question and decides whether a careful reader would give exactly the gold; flagged probes are ambiguous, rest on weak evidence, are malformed, or have a wrong gold. *Scoring check*: all answers of the main evaluation are re-checked; the rules route each one, and the non-trivial cases (a scored-wrong answer that shares words with the gold, a scored-correct answer that is not an exact match, a count answer with several numbers) are judged by gpt-4o. At this scale a small residue of flawed items and scoring errors is hard to avoid; the checks bound it rather than remove it.

Table 7: Quality checks per scenario. Probes: all probes of the 50-session build. Gold leaks: gold values found outside their own planted rounds (generation audit). Ceiling: accuracy of a full-context gpt-6-sol reader over all timelines and checkpoints. Item audit: probes flagged among those asked in the evaluation (eight sampled facts and all other probes of every timeline). Scoring check: answers scored wrong that are correct (missed) and answers scored correct that are wrong (accepted), among all answers of the main evaluation.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**All**
Timelines / probes	17 / 458	19 / 513	20 / 540	20 / 540	20 / 540	20 / 540	116 / 3,131
Gold leaks	0	0	0	0	0	0	0
Full-context ceiling	0.983	0.990	0.995	0.992	0.994	0.983	0.990
Item audit: flagged	26 / 390	24 / 437	9 / 460	16 / 460	10 / 460	15 / 460	100 / 2,667 (3.7%)
Scoring: missed	0.30%	0.20%	0.14%	0.09%	0.40%	0.44%	0.26%
Scoring: accepted	0.02%	0.02%	0.08%	0.08%	0.02%	0.03%	0.04%

D Component-Aware Diagnosis: Conditions and Memory Architectures

This section details how the counterfactual conditions of §5 are built (§D.1), how stage errors are read from them (§D.2), and how the design adapts to each memory architecture (§D.3).

D.1 Diagnostic Conditions

Why a paired ladder. The conditions form a ladder rather than a fully factorial isolation: each rung replaces one stage of the memory pipeline with an oracle and keeps the others as deployed, so the accuracy gain from one rung to the next is attributed to the replaced stage. All conditions answer the same probes on the timelines at the same checkpoints, and are compared probe by probe (Table 8).

Table 8: Diagnostic conditions. Each rung replaces one stage with an oracle. "Tell apart": who distinguishes the answer from its competitors when both are in view.

Condition	Write	Retrieval	Tell apart	What the reader receives
P1	system	system	reader	The system as deployed.
P2	system	oracle	reader	The items of the system's own store at the checkpoint that hold the probe's target or competitor values, whatever the reader searches for.
P3	oracle	oracle	reader	The planted rounds of the history that state the target or competitor values, written before the checkpoint.
P3gold	oracle	oracle	oracle	The planted rounds that state the target values only.
P3ctx	oracle	oracle	reader	The rounds of P3 mixed into what the reader actually had in front of it at P1 (the items its searches returned, or the memory text).

Relevance oracles. P2 and P3 return everything relevant to the probe, the answer and what competes with it, without indicating which is which, as an ideal retriever would. The targets and competitors come from the temporal DAG at the checkpoint: a fact's value and its rivals' values; the current version and the other versions written so far; the hops of the user's chain and of the distractor chain up to the asked depth; the members and near-miss items of a count. Same-topic rounds carry no value to match on and are left out of both. P2 finds the store items by matching these values as spans.

Delivery through the system's own channel. Each condition hands its material over the way the system under test delivers memory, so that only one stage changes per rung: for a system with a search step, the reader keeps its ReAct loop and its search tool returns the oracle items whatever the query; for a system that places its memory in the prompt, the oracle items become that memory text. Both probe forms (question and task, Appendix C.4) are run under every condition; only the probe wording changes.

D.2 Stage Errors

With Acc_c the accuracy under condition c on the paired probes, the stage errors are

$$R = \mathrm{Acc}_{P2} - \mathrm{Acc}_{P1}, \qquad W = \mathrm{Acc}_{P3} - \mathrm{Acc}_{P2}, \qquad U = 1 - \mathrm{Acc}_{P3} = D + U_0,$$

with $D = \mathrm{Acc}_{P3gold} - \mathrm{Acc}_{P3}$ (telling the answer from its competitors when both are in view) and $U_0 = 1 - \mathrm{Acc}_{P3gold}$ (using the answer alone), so that $R + W + D + U_0 = 1 - \mathrm{Acc}_{P1}$. P3 and P3gold hand over clean material, so U is a lower bound on the use error under real conditions; P3ctx measures use in the reader's actual context, $U_{\mathrm{ctx}} = 1 - \mathrm{Acc}_{P3ctx} = N + D + U_0$, where $N = \mathrm{Acc}_{P3} - \mathrm{Acc}_{P3ctx}$ is the loss caused by the rest of that context. Each stage is averaged per timeline and reported with a 95% bootstrap interval over timelines. The ladder assumes that each rung does not lower accuracy; where a stage's interval lies wholly below zero, the cell is reported as a *diagnostic anomaly* rather than as a stage magnitude.

D.3 Memory Architectures and Calibration

Systems without a retrieval step. Memories placed whole in the prompt (Self-managed, Summarize, Consolidating) have no retrieval step, so P2 is not run and write and retrieval are reported jointly as

$WR = \mathrm{Acc}_{P3} - \mathrm{Acc}_{P1}$. For the per-system profile of §5, WR is split per probe by whether the gold value is in the memory text at P1: a probe whose gold is absent counts as a write error, one whose gold is present but missed counts as a read error.

Calibration. Naive RAG stores the raw rounds, so its P2 already holds every round of P3 and its write error should be zero; it serves as a positive control of the ladder. The full-context baseline has no memory system and runs only under P1, as a capacity reference. Neither enters the attribution of other systems.

Scope. We read these conditions as counterfactual diagnosis under paired controls, not as an exact causal decomposition: a stage profile points to a candidate stage for repair.

E Additional Experimental Results

This section gives the experimental setup (§E.1), the variation of the mechanism losses over timelines (§E.2), and supplementary results (§E.3).

E.1 Experimental Setup

Systems under test. A system under test is a memory system together with its internal model; in every run the same model also reads the memory and answers the probes. The memory systems fall into three families: RAG (Naive RAG, HippoRAG 2), in-context memory (Self-managed, Summarize, Consolidating), and agentic memory (A-Mem, MemoryOS, LightMem, SimpleMem, Mem0); full context is the reference without a memory system. Models come from two families, GPT (gpt-4o-mini, gpt-5-nano, gpt-5.6-luna, gpt-6-luna) and Claude (claude-haiku-4.5, claude-haiku-5.5). Non-reasoning models decode at temperature 0 and reasoning models at low reasoning effort; every model call is cached, so a run can be replayed exactly.

Histories and probes. The main experiments use the 50-session builds with ten checkpoints between sessions 8 and 50, on three timelines per scenario (two for claude-haiku-4.5), i.e., 18 timelines per cell. All facts are written in the first eight sessions, and each timeline asks a fixed sample of eight facts together with all other probes at every checkpoint.

Maintenance event. At session 30, the store built so far is carried over and the memory system's internal model is replaced by the other model of the same family (gpt-4o-mini $\leftrightarrow$ gpt-6-luna); the new model performs every later write, the reader is unchanged, and the fact probes are asked at the checkpoints of sessions 35–50. Naive RAG has no internal model that writes the store.

Other experiments. Stage attribution (§5) runs every condition with gpt-6-luna as the reader on the stores of the gpt-6-luna systems, at sessions 8 and 50, in both the question and the task form. The comparison caps the Naive RAG store of all 116 timelines at β times its size at session 8 and keeps it within the cap by eviction or by LLM summarization, reading retention without a reader. The pressure-knob experiments use 200-session builds whose arms change one knob (revision versions, hop depth, aggregation), with Naive RAG and full context read by gpt-6-luna.

E.2 Variation over Timelines

Each build holds many timelines with disjoint content, so the variation of a result is measured over timelines rather than over seeds. Table 9 reports the losses of Table 3 as mean $\pm$ standard deviation.

E.3 Supplementary Evidence

Who ages. Table 10 gives fact accuracy by memory system and model, including the models and the full-context reference that Table 3 does not show.

Derived state of revised attributes. Table 11 asks three questions about the same revised attributes as the number of versions grows: the current version, the version before it, and how many versions there have been. With Naive RAG the current and previous versions fall once the versions outnumber what one search returns, and the count is already low with six versions. With full context the current version stays high, while the count falls to near zero: state that must be recomputed from the whole update history ages even when that history is entirely in view.

Table 9: The losses of Table 3 as mean ± standard deviation over timelines (GPT models; S1–S6, three timelines each). Rows are grouped by model, which is both the memory system's internal model and the reader. For maintenance, the model is the one that built the store, whose internal model is replaced at session 30 by the other GPT model; negative values mean that fewer stored facts were lost with the replacement. Lower is better.

Model	Memory system	Compression ↓	Interference ↓	Revision ↓	Maintenance ↓
gpt-4o-mini	Naive RAG	0.00 ± 0.00	0.37 ± 0.19	0.57 ± 0.36	0.00 ± 0.00
	HippoRAG 2	0.00 ± 0.00	0.30 ± 0.16	0.70 ± 0.25	0.00 ± 0.00
	Self-managed	0.76 ± 0.20	0.17 ± 0.30	0.50 ± 0.35	-0.21 ± 0.30
	Summarize	0.88 ± 0.25	0.15 ± 0.17	0.56 ± 0.34	-0.03 ± 0.41
	Consolidating	0.88 ± 0.24	0.16 ± 0.38	0.19 ± 0.21	0.07 ± 0.28
	A-Mem	0.00 ± 0.00	0.48 ± 0.17	0.72 ± 0.24	0.00 ± 0.00
	MemoryOS	0.00 ± 0.00	0.35 ± 0.16	0.87 ± 0.17	0.00 ± 0.00
	LightMem	0.22 ± 0.15	0.31 ± 0.20	0.65 ± 0.24	0.00 ± 0.00
	SimpleMem	0.08 ± 0.09	0.29 ± 0.19	0.43 ± 0.30	0.00 ± 0.00
	Mem0	0.10 ± 0.10	0.29 ± 0.21	0.87 ± 0.20	-0.01 ± 0.04
gpt-6-luna	Naive RAG	0.00 ± 0.00	0.37 ± 0.15	0.54 ± 0.36	0.00 ± 0.00
	HippoRAG 2	0.00 ± 0.00	0.27 ± 0.15	0.59 ± 0.24	0.00 ± 0.00
	Self-managed	0.09 ± 0.08	0.03 ± 0.07	0.06 ± 0.13	0.18 ± 0.25
	Summarize	0.56 ± 0.30	0.21 ± 0.25	0.13 ± 0.23	0.24 ± 0.50
	Consolidating	0.97 ± 0.07	0.17 ± 0.29	0.04 ± 0.11	-0.01 ± 0.13
	A-Mem	0.00 ± 0.00	0.36 ± 0.13	0.74 ± 0.31	0.00 ± 0.00
	MemoryOS	0.00 ± 0.00	0.22 ± 0.20	0.85 ± 0.21	0.00 ± 0.00
	LightMem	0.01 ± 0.04	0.23 ± 0.17	0.52 ± 0.23	0.00 ± 0.00
	SimpleMem	0.00 ± 0.00	0.11 ± 0.09	0.09 ± 0.15	0.00 ± 0.00
	Mem0	0.08 ± 0.11	0.18 ± 0.17	0.35 ± 0.21	0.01 ± 0.04

Table 10: Who ages: fact accuracy at session 50 (drop from session 8 in parentheses) by memory system and model. A stronger model removes the aging of full context and of self-managed memory, not that of retrieval or of rewrite-based consolidation.

System	gpt-4o-mini	gpt-5-nano	gpt-5.6-luna	gpt-6-luna	claude-haiku-5.5
Full context	0.08 (-0.85)	0.32 (-0.65)	0.79 (-0.20)	0.94 (-0.06)	0.97 (-0.00)
Naive RAG	0.58 (-0.37)	0.44 (-0.52)	0.62 (-0.36)	0.62 (-0.37)	0.69 (-0.29)
Self-managed	0.19 (-0.29)	0.18 (-0.27)	0.88 (-0.05)	0.87 (-0.03)	0.92 (-0.05)
Consolidating	0.07 (-0.45)	0.04 (-0.35)	0.06 (-0.72)	0.02 (-0.47)	0.08 (-0.83)

Use in the reader's actual context. The use error U of Appendix D.2 is measured on clean evidence. Table 12 compares it with U_{ctx}, the use error when the same evidence is placed in what the reader actually saw at P1. With clean evidence the reader almost never fails at either checkpoint. In its actual context, the use error grows by session 50 for the systems that extract or rewrite memory, but not for Naive RAG, whose context consists of raw rounds.

F Implementation Details

This section documents the benchmark's components for reproducibility. Metric definitions are in Appendix B; this section covers running an experiment, the generator's configuration, the memory systems and their prompts, and the cost.

F.1 Running an Experiment

A run has three steps. (1) *Build*: the generator writes, for every timeline of a configuration, the history, the answer key of every checkpoint, the temporal DAG, and the position of every planted round; it calls no model, and a configuration reproduces its build byte for byte. (2) *Evaluate*: a system under test writes the history session by session, is frozen at each checkpoint, and answers that checkpoint's

Table 11: Derived state of revised attributes: accuracy at session 200 by the number of versions per revised attribute (200-session builds, reader gpt-6-luna; Naive RAG on all 116 timelines, full context on two timelines per scenario). *Current*: the latest version; *Previous*: the version before it; *Count*: how many versions so far. Never-updated controls stay at 0.96–1.00 in every arm.

	Naive RAG			Full context		
Versions	Current	Previous	Count	Current	Previous	Count
6	0.97	0.93	0.69	0.94	0.92	0.92
10	0.96	0.82	0.46	0.97	0.89	0.61
20	0.53	0.21	0.00	0.94	0.78	0.17
30	0.32	0.08	0.00	0.83	0.64	0.06

Table 12: Use error with clean evidence ($U = 1 - \mathrm{Acc}_{P3}$) and with the evidence placed in the reader's actual context ($U_{\mathrm{ctx}} = 1 - \mathrm{Acc}_{P3ctx}$), at sessions 8 and 50.

	U		U_{ctx}	
Memory system	s8	s50	s8	s50
Naive RAG	0.01	0.02	0.02	0.02
LightMem	0.01	0.02	0.02	0.12
Mem0	0.01	0.02	0.04	0.20
Self-managed	0.00	0.03	0.02	0.08
Summarize	0.00	0.03	0.02	0.13
Consolidating	0.00	0.03	0.03	0.14

probes; the same entry point runs the diagnostic conditions (Appendix D) and the lifecycle event. (3) *Report*: aging curves and losses are computed from the per-probe records. Every model call is cached on its full request, so a run can be replayed exactly, and the store of every checkpoint is archived, so another reader can be evaluated on the same stores without rebuilding them.

F.2 Generator and Pressure Configuration

Each probe family is generated by one motif of the temporal DAG, and each motif exposes the knobs below. All knobs are fields of a configuration file and can be overridden at build time.

Knob	**Effect**
`n_sessions`, `checkpoints`	Length of the history; sessions at which the probes are asked. Adding or removing checkpoints after the first does not change the history.
`facts`, `rivals`, `topical`	Facts per timeline; other owners' values for the same attribute per fact; same-topic rounds without a value per fact.
`revisions`, `rev_len`, `rev_controls`	Revised attributes; versions per attribute, evenly spread over the history; never-updated controls.
`rev_probes`	Extra questions per revised attribute: the previous version, the number of versions.
`hops`, `depth`, `split_hops`, `hop_place`	Chains per timeline; hops per chain; each later hop split into an introduction round and a value round; chains written early, or spread so that hop h is written at the session of version h.
`aggregations`, `agg_members`, `agg_near`, `agg_topical`, `agg_place`	Count units; members per unit; near-miss items; rounds naming the anchor but no member; members written early, or accumulating over the history so that the count grows.
`scale_ref`	Scales the competing content with `n_sessions`, keeping its density per session constant.
`pool_ref`	Makes the arms of a dose experiment bind the same content units, so that probes pair across arms.

Three families of configurations are released (Table 13). The 50- and 200-session configurations hold 116 timelines over S1–S6; dose arms that need scarce units (many members, deep chains) hold fewer.

Table 13: Released configurations. All share 12 facts per timeline with 12 rivals and 6 same-topic rounds each, 3 revised attributes with 2 controls, 3 chains of depth 3 with split hops, and one aggregation unit with 6 members, 6 near-miss items and 24 anchor rounds, except where noted.

Configuration	Sessions	Checkpoints	Differences
50-session	50	8–50 (5 or 10)	24 versions per revised attribute: the competing content of 200 sessions written within 50.
200-session	200	8–200 (10)	40 versions per revised attribute.
200-session dose arms	200	8–200 (10)	As 200-session, plus three accumulating count units and the previous-version and count questions; each arm changes one knob: versions (6–30), members (6–40), or hop depth (2–8, spread).

F.3 Memory Systems and Prompts

Every memory system writes the history in time order through its own pipeline, with its LLM calls routed to the internal model of the system under test (Table 14). At a checkpoint the store is frozen; systems whose retrieval changes their state (MemoryOS) are probed on a copy.

Table 14: Memory systems. Unit: what one write receives. The reader of a system with a search step uses the ReAct prompt of Appendix C.2; the in-context memories are placed in the prompt.

System	Unit	What the internal model does	Embedding
Naive RAG	round	—	text-embedding-3-small
HippoRAG 2	round	entity and triple extraction; fact filtering at retrieval	NV-Embed-v2
Self-managed	session	adds, rewrites and deletes notes in a notebook	—
Summarize	session	rewrites the memory and the new session into a new memory	—
Consolidating	session	keeps the latest session verbatim and folds older ones into a summary	—
A-Mem	turn	builds a note per turn and evolves related notes	all-MiniLM-L6-v2
MemoryOS	round	pages, topic summaries, user profile and knowledge	all-MiniLM-L6-v2
LightMem	round	extracts memories from pre-compressed turns	all-MiniLM-L6-v2
SimpleMem	window	extracts memory units; plans and reflects at retrieval	Qwen3-Embedding-0.6B
Mem0	round	extracts facts and decides to add, update or delete	text-embedding-3-small

Answer prompts. Memories placed in the prompt and the full-context baseline answer in one call:

Answering from a memory placed in the prompt (Self-managed, Summarize, Consolidating)

You are a personal assistant. Below is your memory of your past conversations with the user, written by you after each conversation.

<the memory: the notebook's notes / the summary (+ raw window)>

Answer the user's question from this memory. If information was updated, use the most recent version. The final answer must be ONLY the requested name or number (for a 'how many' question, only the number): no sentence, no explanation, no list of candidates.

Answering from the full history (full context)

You are a personal assistant. Below is the complete record of your past conversations with the user, each round with its date.

<every round written before the checkpoint: [date] user: ...
assistant: ...>

Answer the user's question from this record. If information was updated, use the most recent version. The final answer must be ONLY the requested name or number (for a 'how many' question, only the number): no sentence, no explanation, no list of candidates.

Self-managed memory. The notebook is the only memory written by the tested model itself. After every session, one call receives the current notebook and that session's conversation and returns operations, applied in order (an operation on an unknown id is skipped). The prompt is identical for every model:

Self-managed: notebook update after every session (same for every model)

You are a personal assistant who keeps a notebook about the user to help them in future conversations. The notebook is all you will remember: the conversations themselves will be gone.

NOTEBOOK (id: note):
<numbered notes, one per line: [id] note; (empty) at the first session>
Below is today's conversation. Update the notebook: add what you may need later about the user (their own things and
choices, names, numbers, plans, the people around them, and anything that changed), rewrite notes that changed, and
delete what is no longer true. Note what belongs to other people only as theirs. Keep each note short and self-contained; add the date when it matters.

CONVERSATION:
<Conversation on <date>:
user: ...
assistant: ... (the whole session)>

Return JSON {"ops": [{"op": "add", "text": "..."}, {"op": "update", "id": <id>, "text": "..."}, {"op": "delete", "id": <id>}]}

Summarize and Consolidating. Both compact the history with the prompt below, led by the scenario of the build: its persona and content range are filled verbatim from the scenario definition, without reference to the probes. Summarize rewrites the memory and the new session into a new memory after every session; Consolidating keeps the latest session verbatim and folds older ones into the summary. The word budget N is 2% of the words of the history the summary stands for, within $[50, 4000]$; an output over $1.1N$ words is rewritten once with the same prompt, then cut. For S2, the persona is *someone whose lifestyle assistant tracks their budgets, diet, preferred stores and weekly schedule*, and the content hints are personal budgets and spending limits by category; diet: allergies, restrictions and favourite foods; preferred stores, brands and where the user buys things; schedule: classes, appointments, recurring activities and their venues.

Summarize and Consolidating: compaction prompt

You are a personal assistant's memory manager. The user is <persona of the scenario>. Below are your notes and conversations with the user.
Rewrite them as a concise memory. You MUST preserve ALL of the following verbatim:
<one line per content hint: - Everything about <hint>>
- Every other named person, place, organisation, product or thing the user mentions, and what it is to the user
- Every number, amount and date, with what it refers to
- Every item of a list the user keeps (each one separately)
- Every update: when something changed, the most recent value with its date
Write only what the text states; do not add any detail that is not in it. Be concise but complete.
Write at most <N> words.

DOCUMENT:
<previous memory, then the new session(s)>

SUMMARY:

Third-party systems. Mem0 (1.0.11), LightMem (8449d57), SimpleMem (db80b6a), HippoRAG 2 (2.0.0a4), MemoryOS (587ed77) and A-Mem (b31a1dd) run with their own prompts and default settings. Changes needed to run them:

- Mem0: one conversation round per `add()`; the date its prompts give as today is fixed, so that cached calls stay valid.
- LightMem: the configuration of its LongMemEval script (LLMLingua-2 pre-compression, user-only extraction); eager attention in the segmenter; a failed pre-compression raises an error instead of silently keeping the raw text.
- HippoRAG 2: one passage per round (whole sessions overflow its entity-extraction prompt); an entity answer given as a bare JSON list is also parsed.
- SimpleMem: code defaults, with parallel retrieval off for a deterministic order.
- MemoryOS and A-Mem: code defaults; A-Mem writes one note per turn, as in its LoCoMo driver.

F.4 Cost of Experiments

Table 15 reports the API cost of one main-experiment cell (S1–S6, three timelines each, ten checkpoints). Calls answered from the shared cache are not charged, so the figures are lower bounds; local models (embeddings, LightMem's pre-compression) are not included.

Table 15: API cost (USD) of one cell of the main experiments on 18 timelines.

Model	Naive RAG	HippoRAG-2	Self-managed	Summarize	Consolidating
gpt-4o-mini	$2.0	$4.4	$1.6	$1.4	$2.4
gpt-6-luna	$1.3	$5.2	$3.3	$3.9	$4.8
Model	A-Mem	MemoryOS	LightMem	SimpleMem	Mem0
gpt-4o-mini	$13.6	$13.7	$1.0	$14.5	$4.2
gpt-6-luna	$18.0	$5.8	$1.1	$8.5	$6.0

G Case Studies

This section walks through two probes of S2 in detail. §G.1 follows one wrong answer given by six memory systems through the diagnostic conditions and into what each system kept; §G.2 follows one stored fact through ten checkpoints of two in-context memories. All excerpts are verbatim; numbers in brackets are session indices.

G.1 Same Wrong Answer, Different Repairs

Multi-hop probe, depth 2 (S2, asked at session 50)

Q: What banking service does my budget app integrate with? **gold:** ZephyrBank

User's chain: [6] My budget app is named PennyTrack. [7] Hey, I just found out about this cool service called PennyTrack that helps manage expenses. / I heard they integrate with a banking service known as ZephyrBank.

Distractor chain (another owner): [34] Alex's finance tracker is named CoinWatch. [40] I'm looking into this service called CoinWatch. / It integrates with a banking service known as StarlingBank.

At session 50, six of the seven memory systems in the attribution experiment (gpt-6-luna as internal model and reader) answer *StarlingBank*, the value of the other owner's chain. The diagnostic conditions of Appendix D locate the failure at three different stages (Table 16).

Table 16: One wrong answer, three stages. Gold seen: the gold value is among the items the reader retrieved (systems with a search step) or in the memory text (in-context memories) at P1. Recovered at: the first condition under which the answer is correct.

System	P1 answer	Gold seen at P1	Recovered at	Stage
Naive RAG	StarlingBank	yes	P2	retrieval
LightMem	StarlingBank	yes	P2	retrieval
MemoryOS	StarlingBank	no	P2	retrieval
Mem0	StarlingBank	no	P3	write
Summarize	StarlingBank	no	P3	write
Consolidating	StarlingBank	no	P3	write
Self-managed	ZephyrBank	yes	—	—

What each system had in front of it explains the three stages.

Retrieval stage: Naive RAG at P1 (one search)

Search: budget app banking service integrates with

Top results, in rank order (3 of 10 shown):
[40] user: It integrates with a banking service known as StarlingBank.
[7] user: I heard they integrate with a banking service known as ZephyrBank.
[6] user: My budget app is named PennyTrack.

Not retrieved: [34] Alex's finance tracker is named CoinWatch. [40] I'm looking into this service called CoinWatch.

Answer: StarlingBank

The store holds every round, and the search returns both values, but not the rounds that say whose service CoinWatch is; the StarlingBank round, whose subject is only "it", reads as if it continued the user's own chain. Handing over the rounds of both chains (P2) recovers the answer: the repair is in retrieval.

Write stage: what Mem0 stored (4 of 373 items)

The user's budget app is named PennyTrack, a service that helps manage expenses.
Asked where ZephyrBank is based.
Alex's finance tracker is named CoinWatch.
It integrates with a banking service known as StarlingBank.

Mem0 kept both apps, but no item ties ZephyrBank to PennyTrack, while the StarlingBank item survives without its subject. Even when these items are handed over (P2), the reader answers StarlingBank; only the raw rounds (P3) recover the answer: the repair is in what Mem0 writes.

Write stage: what the in-context memories kept at session 50

Summarize (1,769 words): PennyTrack tracks expenses. [...] Alex's finance tracker: CoinWatch; user looking into CoinWatch, which integrates with StarlingBank, headquartered in Lumenville.

Consolidating (summary of 1,713 words): CoinWatch is Alex's finance tracker and integrates with StarlingBank. *(No mention of PennyTrack or ZephyrBank.)*

Self-managed (answers correctly): ZephyrBank, which integrates with PennyTrack, is headquartered in Novaville. [...] Alex uses a finance tracker called CoinWatch. User is looking into CoinWatch; StarlingBank integrates with CoinWatch and is headquartered in Lumenville.

Summarize kept the user's app but dropped ZephyrBank, and recorded the distractor's integration as the user's own interest; Consolidating kept only the distractor chain. In both, the gold is no longer in memory, so no reader can recover it: the repair is in what is kept. The notebook kept both chains with their owners. The symptom is the same for six systems; the repairs are a better retrieval for three, and a better write for the other three.

G.2 A Stored Fact Lost by an In-Context Memory

The fact below is stated once, in session 4, and never mentioned again by the user. Other people's cosmetics brands follow over the history, for example:

Fact probe (S2) and some of its rivals

Q: Could you remind me which cosmetics brand I have a particular fondness for? **gold:** VelvetEssence

Evidence: [4] I'm putting together a shopping list for skincare products, and since I'm loyal to VelvetEssence, I need to make sure I have all their essentials. [...]

Rivals (12), e.g.: [9] I heard my neighbor Sarah got hooked on MoonAura cosmetics after trying their foundation. [...] [26] Nathan, my college roommate, is a fan of LushLuxe because of their exotic scents. He's convinced me to try some when I can. [31] My colleague, Emma, uses SparkSilhouette for her cosmetics because she loves their radiant finish. [...]

Table 17 follows the fact through two in-context memories: Summarize with gpt-6-luna and Self-managed with gpt-4o-mini.

Table 17: One fact at every checkpoint. Size: words of the memory text (Summarize) or number of notes (Self-managed). In memory: the gold value occurs in the memory. Answer: the reader's answer (same model as the internal model).

	Summarize (gpt-6-luna)			**Self-managed** (gpt-4o-mini)		
Session	**Size**	**In memory**	**Answer**	**Size**	**In memory**	**Answer**
8	340	yes	VelvetEssence	24	yes	VelvetEssence
12	491	yes	VelvetEssence	44	yes	MoonAura
16	717	yes	VelvetEssence	59	yes	MoonAura
20	893	yes	VelvetEssence	71	yes	MoonAura
25	1,057	yes	VelvetEssence	5	no	I'm sorry, but I don't have any information [...]
30	1,255	yes	VelvetEssence	28	no	You haven't mentioned a specific cosmetics brand [...]
35	1,470	no	Not specified	57	no	SparkSilhouette
40	1,683	no	Not specified	74	no	LushLuxe
45	1,776	no	PrismAura	100	no	LushLuxe
50	1,943	no	I don't have that information.	126	no	LushLuxe

Summarize (gpt-6-luna): the fact drifts to the end of a growing summary

Session 8 (340 words, the line at word 136): **Stores/brands:** [...] Preferred clothing boutique: SilkenThread; exclusive line: VelvetFrost; flagship jacket: Midnight Whisper. Loyal to skincare brand VelvetEssence. Wants to check essentials and sales at preferred stores. [...]

Session 30 (1,255 words, the line at word 1,164): [...] Preferred boutique SilkenThread; exclusive line VelvetFrost; flagship jacket Midnight Whisper. Loyal skincare brand VelvetEssence; wants to check essentials/sales at preferred stores. [...]

Sessions 31–35: every rewrite exceeds the word budget even after the second pass and is cut from the end, by 159 to 630 words.

Session 50 (1,943 words): the only skincare line left is about another person: Niece Olivia drinks 2825 milliliters daily for skincare and believes it makes a noticeable difference.

As the summary grows, the line about the brand moves from 40% to 93% of its length by session 30, toward the end that the budget cuts. It disappears in the rewrites between sessions 30 and 35, all of which were cut. A fact the summary still holds becomes fragile once it sits near the end of a text that grows faster than its budget.

Self-managed (gpt-4o-mini): the notebook rewrites the fact, then deletes itself

Session 8 (24 notes): [19] User is putting together a shopping list for skincare products from their preferred store, VelvetEssence.

Sessions 12–20: note 19 is still present, next to [44] User's neighbor Sarah got hooked on MoonAura cosmetics after trying their foundation. The reader answers MoonAura.

Update after session 24: one update deletes 86 notes; 5 notes remain.

Session 50 (126 notes), e.g.: [109] User's college roommate is named Nathan, who is a fan of LushLuxe because of their exotic scents and has convinced the user to try some. [130] User's colleague Emma uses SparkSilhouette for her cosmetics because she loves their radiant finish.

The notebook loses the fact in two steps. At write time, the brand becomes a "store", so that from session 12 the only note that mentions *cosmetics* is a rival's and the reader answers with it although the gold is still in memory. After session 24, a single update deletes almost the whole notebook. The notes written afterwards come from later sessions, which hold only rivals; one of them is phrased as something the user was persuaded to try, and the reader answers with it.

Both memories held the fact when its writing phase ended, and both lost it while rewriting their own contents. Summarize loses it silently and the reader declines to answer; the notebook loses it to a rival and the reader answers wrongly with confidence. that rival.

H Evaluation Card

What AGINGBENCH reports. For each system under test (a memory system with its internal model), AGINGBENCH reports aging curves of a fixed probe set at every checkpoint of a timeline, one loss per aging mechanism (compression, interference, revision, maintenance), and a component-aware diagnosis (§5) that points to the write, retrieval, or utilization stage of the memory pipeline. Histories are generated deterministically from frozen content libraries (Appendix C.1), so every build is reproducible from its configuration, and every probe is scored by rule against its answer key.

Scope of attribution claims. We treat the component-aware diagnosis as *counterfactual diagnosis under paired controls*, not as exact additive causal decomposition. The conditions (P1, P2, P3, P3gold, P3ctx) yield response profiles that we interpret as diagnostic profiles pointing to a candidate stage; we do not claim unique causal identification. The oracles of P2 and P3 are relevance oracles: they hand over the answer together with its competitors, without indicating which is which. Where the additive accounting requires the ladder to be monotone and a cell violates it, we report the cell as a *diagnostic anomaly* rather than as a stage magnitude. For memories without a retrieval step (memories placed whole in the prompt), P2 is not run and write and retrieval errors are reported jointly.

Intended use. Two regimes: (i) *evaluating a memory system or a model*: a new system under test is registered once and run on the released builds, which yields its aging curves and mechanism losses next to the systems evaluated here; and (ii) *diagnostic analysis* (§5) that identifies which mechanism binds a given system and which stage a repair should target.

I Broader Discussion

Implications for long-lived agent system design. Three design observations follow from our results. First, memory design and model capability interact: a stronger model removes part of the aging of some memories, such as full context or a self-managed notebook, but not that of retrieval or of rewrite-based consolidation, so investment in a stronger model should be paired with an assessment of which component binds. Second, monitoring answers alone can miss aging: a memory that loses a fact may lead the agent to decline, or to answer confidently with a competitor, and only probes of what the agent should still know expose the difference. Third, memory *curation* is an active operation with its own failure modes: rewriting, consolidating and deleting stored content can remove knowledge the agent already had, so the operations that keep a memory within its budget deserve the same scrutiny as the operations that write it.

Connection to task horizons. If the *task horizon* an agent can reliably handle is a north-star metric for agent progress, AGINGBENCH supplies the longitudinal component that one-shot evaluations cannot. Its aging curves describe how reliability erodes as the history grows and as competing content accumulates, and its pressure knobs vary these axes one at a time. In the configurations we tested, a stronger model closes part of this erosion but not all of it. Extending the usable task horizon therefore requires intervening on the specific mechanism that binds the regime in question, whether compression of what is written, interference as similar content accumulates, revision as user state evolves, or maintenance around lifecycle events; single-axis improvements are unlikely to yield deployment-stable agents.

Aging as a runtime control problem. A long-lived agent makes continuous decisions about what to write, what to compress, what to retrieve, and when to reorganize its memory, mirroring the deliberate practices humans use to maintain cognitive function with age. Read this way, each memory system in our evaluation is a point in a control-policy space, and its per-mechanism aging curves are the closed-loop evaluation of that policy. The per-checkpoint signals AGINGBENCH emits could in principle drive such decisions at run time, which makes it a natural testbed for runtime-control studies within agent lifespan engineering. State that must be recomputed from a whole update history, such as how many times a value changed or what it was before, ages even when the full history is in view. This suggests that text memory may be the wrong abstraction for such state, and that keeping it in an explicit, structured form updated at write time is a natural direction for memory design.

LLM Declaration. LLMs are central to this work as the research objective under evaluation. They appear methodologically as the experimental subject (multiple LLMs across families) and as components of the evaluated pipeline (the memory management; the tool-calling loop in ReAct loop for agent). Scoring does not use LLM-as-judge: metrics are computed by deterministic procedures over gold ground truth carried through the temporal dependency DAG. The authors additionally used LLMs for language refinement editing (e.g., grammar, spelling, word choice), code implementation, and experiment conducting; this did not influence methodology design, claims, or originality.

Limitations and the open frontier. AGINGBENCH contributes a mechanism-level vocabulary for agent lifespan engineering, covering four mechanisms (compression, interference, revision, maintenance), along with a reproducible protocol that measures them over controlled history lengths and diagnoses aging at specific pipeline stages rather than against aggregate model quality. Its scope has limits. The histories are synthetic: facts are planted into scenario-themed conversations, and base conversations recur across the timelines of a build, so timelines are not fully independent. Most probes ask for a name or a number; the task form covers part of the probes. The content libraries were authored and checked with particular model families, which may favor some readers. A small share of items and scores remains flawed despite the checks. Lifecycle events can be extended in future work to more types of operations on stored memory, such as flushes and migrations. The open frontier is anchoring this vocabulary to production deployment telemetry to verify at real-user timescales that the mechanisms characterized here compound over weeks-long deployments; we hope the community can extend this diagnostic lens to deployed systems.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have summarized the position and key contributions of the paper in the abstract and introduction parts.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are discussed in Appendix I.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: There are no theoretical results in this paper.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We introduced the experiment and implementation details in Appendices E.1, C, F, and E.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code can be accessed by our project page at https://agingbench.github.io, with details introduced in Appendix F.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental setup and configurations are documented in Appendices E.1 and F.3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The multi-seed validation results are reported in Appendix E.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The compute resources are summarized in Appendix F.4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The broader impacts are discussed in Appendix I.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not release new models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All assets used in the paper are cited in Appendix E.1 and Appendix A.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The released benchmark is documented in Appendices C and F, with intended use described in the Evaluation Card (Appendix H).

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper is not about crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper is not about research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The LLM usage is described in the LLM Declaration paragraph of Appendix I.

Guidelines:

- The answer [N/A] means that the core method development in this research does not invade LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.