FILESYSTEM-BASED MEMORY FOR LLM AGENTS: ORGANIZATION, EVOLUTION, AND SUSTAINABILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Deployed large language model (LLM) agents increasingly keep their long-term memory as a filesystem: a directory tree of markdown files that the agent itself reads, writes, and reorganizes through generic file tools. Yet research has largely passed over this medium: prior systems design bespoke memory representations and study retrieval over them, leaving the default's two working assumptions untested: that an agent can keep a growing memory organized as memories accumulate, conflict, and go stale, and that this organization pays. We present, to our knowledge, the first systematic exploration of filesystem-based memory for LLM agents. We formalize the setting as three roles around one memory filesystem: a management agent integrates and organizes incoming content, a search agent answers queries with cited sources, and an execution agent supplies task trajectories that are distilled into skills, unifying declarative memory and skills in the same filesystem. Across long-conversation benchmarks and embodied tasks, we vary the memory variant, the stream's scale, the harness, and the backbones of the management and search agents, tracking answer quality, cost, and the health of the memory as it grows. Organization reliably buys search economy, but not always answer quality: reorganized memories more than halve per-query search cost on the two PersonaMem tiers, where the memorized content is largest, while no memory organization leads answer quality everywhere. As a memory grows it stays useful and its structure holds, but its fit to the content erodes in the well-organized memories that keep adding files. Capability matters in the agent that reads on conversation and, as a step at the top tier, in the agent that writes in the embodied setting, and two other model families also show no universally best organization while changing its form and its benefits. The harness alone reshapes the memory as strongly as the model. The study turns the filesystem default from an assumption into a design space for agent memory.

1 INTRODUCTION

LLM agents increasingly work over horizons no single context window can span, maintaining codebases across sessions and assisting the same user for months, and, in the field's ambition, learning continually from their own experience. Both demand a memory that persists across episodes and evolves: absorbing new information, reconciling it with what is stored, and staying organized enough to remain searchable and trustworthy as it grows. The context window is ephemeral and degrades long before it is full (Liu et al., 2024), so persistent external memory has become a first-order component of agent design (Zhang et al., 2025). We use *memory* in the inclusive sense (Sumers et al., 2024): declarative content (facts, events, preferences) and procedural content, the reusable *skills* an agent distills from experience (Wang et al., 2023; Anthropic, 2025b), held in one place.

Research has given this memory many forms, from paged context (Packer et al., 2023) and extracted fact memories (Chhikara et al., 2025) to knowledge graphs (Rasmussen et al., 2025), note networks (Xu et al., 2026), summary banks (Zhong et al., 2023), discourse-unit stores (Pan et al., 2025; Zhou & Han, 2025) and embedding-organized trees (Rezazadeh et al., 2025), each behind its own interface. Much deployed practice has converged on something plainer: the *filesystem*. Anthropic's memory tool is a directory of files behind six file operations (Anthropic, 2025a), Claude Code keeps its notes in a memory folder (Anthropic, 2026a), and context files and skills ship as repository markdown at ecosystem scale (OpenAI, 2025; Anthropic, 2025b). The filesystem is inspectable, portable,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

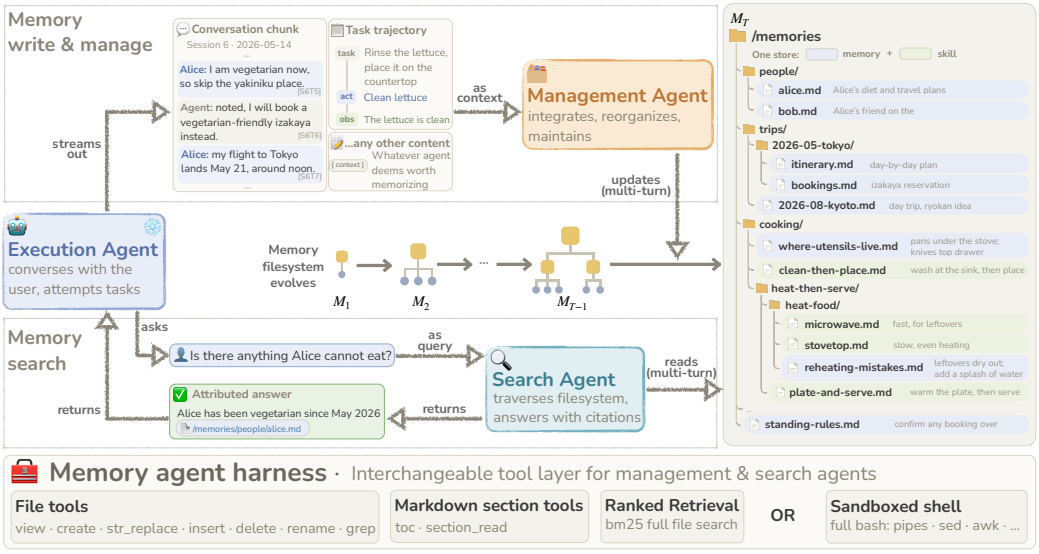


Figure 1: Filesystem-based agent memory. The management agent integrates streamed content into one memory filesystem and keeps it organized; the search agent returns cited answers or retrieved skills. Both act on the memory only through an interchangeable harness. The optional execution agent supplies trajectories, directly or through logs, and consumes what is retrieved. Declarative memories and skills share the filesystem, whose evolution over the stream is tracked.

operated with the tools agents already master, and natively hierarchical: folders form a taxonomy whose names are its labels. Yet research has largely passed over the default medium: prior work builds systems *on* filesystem memory and studies retrieval *over* files (Zhang et al., 2025), while how an agent-curated memory filesystem should be built, organized and kept healthy has received little systematic study.

The default rests on an untested assumption: that the memory stays manageable as it grows. Over long horizons memories accumulate, and with them duplicates, contradictions, stale facts and content scattered across files, so the memory must be updated, reconciled and reorganized, and doing this badly can degrade a memory below the no-memory baseline (Zhang et al., 2026). Existing mechanisms act per item (Chhikara et al., 2025) or run outside the agent: OpenAI's dreaming synthesizes memory in the background (OpenAI, 2026), and Anthropic's *Dreams* rebuilds the memory wholesale between sessions (Anthropic, 2026b), introduced precisely because the working agent's own writes stay local and incremental. The filesystem's native answer is the human one, *organizing*: grouping related content into folders, naming it so it can be found again, splitting and merging as content demands. Whether LLM agents can do the same, and whether it pays, is open in both directions: organization might be what keeps a growing memory sustainable, or a flat memory swept by strong search tools might serve just as well.

This paper presents, to our knowledge, the first systematic exploration of filesystem-based memory for LLM agents. Section 2 formalizes the setting as three roles around one memory filesystem (Figure 1): a *management agent* integrates each incoming chunk and keeps the memory organized, a *search agent* answers queries over it with cited sources, and, in the embodied setting, a fixed *execution agent* attempts tasks. We instantiate the setting on question answering with source attribution over long dialogues and on embodied task success, vary how the memory is built, the scale of the stream, the harness, and the backbones of the two agents, and measure quality and success together with cost and the health of the memory over time.

The study answers four questions, the first two in Section 4 and the other two in the appendix. **RQ1 (organization).** Management agents grow subject-based trees whose organization is set by the model and the harness: backbones change the file count by an order of magnitude, and the harness reshapes the memory as strongly as the model. **RQ2 (value).** Organization reliably buys search economy, not always answer quality; which memory serves an execution agent depends on its strength; capability matters in the search agent on conversation and, as a step at the top tier, in the management agent in the embodied setting, where replacing the harness moves success as much

as a backbone step. **RQ3 (growth, Section E.3).** Within our horizons memory keeps its advantage over no memory as it grows, raw experience gaining late and distilled guidance early, and most early files survive; the fit to the contract erodes in the well-organized memories that keep adding files, and search cost grows with the memory for the raw log while every memory shortens episodes. **RQ4 (model families, Section F).** Across two other model families no organization leads answer quality everywhere either, while citation support, search cost, the memories written, and the standing of curated skills against raw logs change.

Contributions. (1) A formalization of filesystem-based agent memory as three roles with minimal contracts around one memory filesystem, unifying declarative memory and skills. (2) A study framework spanning long conversations and embodied tasks, with controlled memory variants, harness, scale and backbone axes, and lifecycle measures of a growing memory. (3) An empirical characterization that turns the filesystem default from an assumption into a design space: when organization repays its cost, what to serve weak and strong execution agents, where backbone capability matters, and the harness as a control over organization.

2 Formalizing Filesystem-Based Agent Memory

The formalization abstracts deployed memory systems into the components we vary.

2.1 The memory filesystem

A *memory filesystem* $\mathcal{M}$ is a finite set of files organized in a rooted path hierarchy. Each file $f \in \mathcal{M}$ is a triple $f = (p_f, d_f, c_f)$: a path p_f (for example `/memories/people/alice.md`), a one-line description d_f, and text content c_f. Folders are the shared prefixes of paths and carry no content of their own. The folder structure, the file names, and the markdown headings inside files together form the memory's *taxonomy*: one labeled tree that continues below the file level into nested sections, navigated top-down, whose names and one-line descriptions, exposed by directory listings and ranked search, are all an agent sees before opening a file. We write $\varnothing$ for the empty memory. In our instantiation each file is a markdown document whose frontmatter carries its name and description (Figure 2; file anatomy in Section B.1), and nothing below depends on this choice.

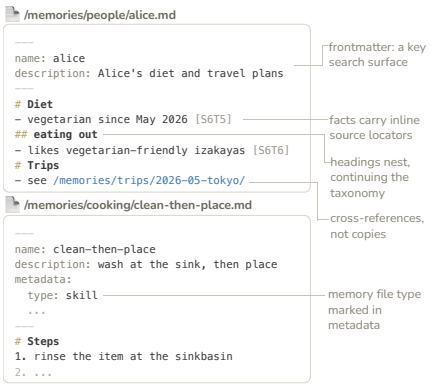


Figure 2: A declarative memory file (top) and a skill file (bottom) share one anatomy: frontmatter, headings, source locators, cross-references and metadata (Section B.1); `[S6T5]` reads session 6, turn 5.

The taxonomy contract. What should this tree look like? Both settings' management instructions state the same five properties (Sections C.1 and C.5), which we adopt as the paper's working definition of a well-organized memory, binding at every level of the tree, headings included (full statements in Section A.1, measurements in Section A.2). **P1, sibling distinction**: siblings can be told apart by name, else by name and description, without opening them. **P2, sibling relatedness**: siblings are related enough to belong together. **P3, parent-child coverage**: a parent's name covers its descendants and, as far as practical, all in-scope content lives under it. **P4, tree-wide proximity**: more related content lies nearer together. **P5, structural economy**: depth is added only where it improves routing to a fact. Section 4.1 reads the memories agents actually build against this contract.

2.2 Harness and agent roles

Harness. Agents never manipulate $\mathcal{M}$ directly; every access passes through a *harness* $\mathcal{H}$, a finite set of operations $o : (\mathcal{M}, \text{args}) \mapsto (\mathcal{M}', \omega)$ that return a text observation ω and may mutate the memory (read operations leave $\mathcal{M}' = \mathcal{M}$). We consider a sandboxed shell over the memory's directory; a function set mirroring the industry-default memory tool's six operations (view, create,

string-replacement and insertion edits, delete, rename); and search-augmented variants that add line-level regex search or ranked keyword search. The harness is an experimental axis in its own right.

Agents. An agent binds an LLM policy π to the harness: it runs a tool loop of operation calls and observations through $\mathcal{H}$ and terminates by emitting its output. Three roles share this form and differ only by contract.

Management agent. Given an instruction ι, an incoming chunk x_t, and the current memory, the management agent produces the next memory:

$$\mathcal{M}_t \;=\; \mathrm{m}_{\pi}^{\mathcal{H}}(\iota,\, x_t,\, \mathcal{M}_{t-1})\,, \qquad \mathcal{M}_0 = \varnothing, \tag{1}$$

so a stream $x_1, \ldots, x_T$ induces a memory trajectory $\mathcal{M}_1, \ldots, \mathcal{M}_T$. The mandate is integration *and* maintenance: the agent may create, rewrite, merge, split, move or delete anything in the memory, so organization is part of its job, and how well it exercises this mandate as T grows is a central object of study.

Search agent. Given an instruction, a query q, and a fixed memory, the search agent returns an answer with citations:

$$(a, \Gamma) \;=\; \mathrm{s}_{\pi'}^{\mathcal{H}_r}(\iota,\, q,\, \mathcal{M})\,, \tag{2}$$

where Γ is a set of references into $\mathcal{M}$ (file paths, optionally sections or lines) supporting a, and $\mathcal{H}_r$ is the harness available to search. Search is read-only in intent, since an answer is graded against the memory that was searched; a write-free operation set enforces this structurally, a write-capable shell only by instruction.

Execution agent. In the embodied setting a third role appears: an execution agent $\mathrm{e}(\tau, \Gamma) = (\xi, z)$ attempts a task τ with the retrieval payload Γ, retrieved skill files or synthesized guidance, in its context and returns its trajectory ξ and a success signal $z \in \{0, 1\}$. It is the probe that converts memory quality into task outcomes; the search agent serves as its retrieval subroutine and its trajectories are the chunks the management agent integrates. Without it, as in the conversational setting, the stream comes from the environment and queries are posed externally. The contracts are minimal so the roles map onto deployed harnesses, where one model may play all three (Section B.1).

2.3 Two settings, one memory filesystem

The conversational setting. Contiguous slices of a multi-session dialogue, with inline source locators on every turn, form the stream $x_1, \ldots, x_T$, and the management agent builds $(\mathcal{M}_t)$ by Equation (1). Questions are answered from the final memory and graded for correctness and citation support.

The embodied setting. A group of K tasks is attempted in sequence. The memory holds *skills*, reusable procedures, and optionally *notes*, lessons, observations and warnings distilled from attempts (Section B.5). For task τ_i, retrieval, execution and curation follow

$$\Gamma_i = \mathrm{s}_{\pi'}^{\mathcal{H}_r}(\iota_r, \tau_i, \mathcal{M}_{i-1}), \qquad (\xi_i, z_i) = \mathrm{e}(\tau_i, \Gamma_i), \qquad \mathcal{M}_i = \mathrm{m}_{\pi}^{\mathcal{H}}(\iota_c, \mathrm{render}(\xi_i), \mathcal{M}_{i-1}), \tag{3}$$

where Γ_i is the search agent's retrieval payload, selected files or task-specific guidance, ι_r and ι_c are fixed role instructions, and $\mathrm{render}(\xi_i)$ serializes the trajectory into a chunk. Each task retrieves from a memory built only from earlier attempts, so later performance measures transfer.

Both settings share the memory filesystem, the role contracts and the harnesses, so one system serves declarative and procedural memory; chunks are dialogue slices or rendered trajectories, queries are questions or tasks. The contracts are fixed and the execution agent is fixed within each cell; every measurement follows the memory trajectory $(\mathcal{M}_t)$: downstream quality, the cost of management and of deployment (search plus execution), and the organization of the memory over time (Section 3).

3 Experimental Setup

Full settings: Section B (memory variants in Table A2, prompts in Section C, metric definitions in Section A).

Benchmarks and metrics. Conversational: LoCoMo, PersonaMem at its 32k and 128k tiers, and REALTALK (Maharana et al., 2024; Jiang et al., 2025; Lee et al., 2025), judged for correctness

and attribution (whether the cited memory supports the answer). Embodied: ALFWorld and ScienceWorld (Shridhar et al., 2021; Wang et al., 2022), reporting success and, for ScienceWorld, a subgoal-based task score. Both: cost, split into management cost and deployment cost (search plus execution), and memory organization. Question counts, task families, adaptations, metric definitions and the paired sign tests are in Sections A, B.2 and B.5.

Memory variants. Six conversational variants: **Closed-book**, no memory; **Chunk retrieval**, the stream chunked and indexed for ranked keyword search (Lewis et al., 2020); **Verbatim dump**, one verbatim file per session; **Foldered sessions**, the Verbatim dump's files moved into an LLM-designed folder taxonomy; **Reorganized memory**, the Verbatim dump restructured by an LLM, in a Reorg. (condense) default version and a Reorg. (preserve) version that adds one "keep every fact" rule; and **Agent-curated memory**, built from empty by the management agent of Equation (1). Five embodied variants: **No memory**, no memory; **Episode log**, each attempt's rendered trajectory appended verbatim; **Curated skills**, attempts distilled into procedure files by the management agent; **Curated skills+mem**, curated skills plus memory notes under an outcome gate; and **Curated skills+mem (GS)**, that memory read by a search agent that writes task-specific guidance instead of returning files.

Agents and judge. The management agent and search agent run `gpt-5.4-mini` under one fixed prompt family; the execution agent is fixed within a cell at `gpt-4.1` or `gpt-4.1-mini`. A fixed `gpt-5.4-mini` judge, calibrated before any experiment ran, grades conversational answers (Zheng et al., 2023) and shares a vendor with the default family (Panickssery et al., 2024). Section F repeats the study with `gemma-4-31B-it` and `claude-sonnet-5` as the two memory agents.

4 Results and Analysis

This section answers RQ1 (what organization a management agent produces, and what determines it) and RQ2 (whether organization pays, in quality and search cost where memory answers questions and in success and deployment cost where an agent acts on it) in both settings. RQ3 (whether a memory stays useful and organized as it grows) is answered in Section E.3 and RQ4 (whether the answers hold for two other model families, under the settings of Section B.7) in Section F.

4.1 What organization do management agents produce, and why? (RQ1)

4.1.1 Organization by memory variant and scale

What does the management agent build when told to organize? Table A5 profiles every agent-curated memory and Table A6 gives the shape of its tree. Together they show that the agent-curated memories in both settings group content by subject, with the hierarchy at different levels: LoCoMo keeps it inside three files, with 116 headings nested five deep and knit by 151 cross-references; PersonaMem 128k inside two files with 210 headings nested six deep; REALTALK across 41 small files in two folders. Is this form set by the content or by the model? Holding the management agent at `gpt-5.4-mini` and moving from PersonaMem 32k to 128k, a stream five times longer, the memory consolidates: about two folders of twelve files become one folder of two files while the headings grow from 53 to 210. Holding the scale at 128k and changing only the backbone, the same conversation becomes 122 files in twelve folders under `gpt-5.4-nano`, two files under `gpt-5.4-mini`, and 105 files nested seven deep and stitched by 233 cross-references under `gpt-5.4`. **The backbone changes the file count sixtyfold on one fixed conversation, whereas the scale contrast also changes the conversation, so it cannot be read as a scale law.** Section 4.1.2 reads this signature in both settings (full reading: Section E.1.1).

Constraining that behavior shows what the model puts at risk: in Table A5 foldering preserves file contents, whereas free reorganization condenses silently, shrinking PersonaMem 128k from 612 to 20 kilobytes and REALTALK from 105 to 18, and a "keep every fact" instruction roughly preserves byte volume on LoCoMo and REALTALK (Section E.1.1). Section 4.2.1 tests whether useful information survives this compression.

Counts of files and headings say how much structure a memory has, not whether it is good structure. Against the taxonomy contract (Table A8, metrics in Section A.2), LoCoMo's three-file design has sibling cohesion no better than random (B2 0.91) and 15.7 percent of its sections lexically closer

to another group, whereas the memories organized at the file level keep siblings 2.4 to 5.0 times more similar than random with leakage under 4 percent, the most sharded 128k rebuilds highest. Few files alone do not explain these differences, since the two-file PersonaMem 128k memory is cohesive with little leakage; and cohesion does not make the sharded memories uniformly more accurate (Section 4.2).

The embodied setting asks the same question over procedures, where the management agent receives the execution agent's own trajectories. At the end of the ALFWorld single chains (Table A15) the Episode log holds 140 files (220 to 270 kilobytes), while the curated memory holds 45 files and 70 kilobytes under the weak execution agent but 137 files and 120 kilobytes under the strong one. **Curation compacts bytes at both tiers but files only under the weak execution agent: the same management agent writes 137 or 45 files depending on the trajectories it is fed.** Along the chain (Figure A7) the log adds one file per task and most curated memories keep adding files, while the gpt-5.4 management agent adds only two net files after task 35, ending with 16, and edits and expands their contents instead (Section E.3). Against the contract (Table A8), the four reported skill memories have scope leakage below 2.5 percent, sibling cohesion of 1.9 to 3.9 and B4 of .14 to .37; low leakage alone does not establish semantic coherence. Along the chain (Figure 3) this fit erodes in the well-organized memories that keep adding files, from .46 after task 40 to .30 after task 140 for the mini-curated chain and from .26 to .14 under Center+BM25, while the memory the gpt-5.4 management agent built holds at .25 to .23 (Table A28; Section E.3). The memory that tracks relatedness best at the end (.37) is the one built through the shell, the harness effect that Section 4.1.3 takes up.

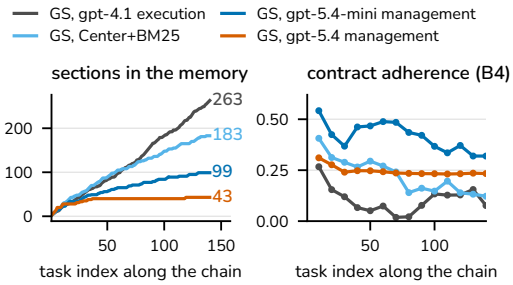


Figure 3: Organization along the ALFWorld chains: sections in the memory (left) and contract adherence, B4 at every tenth task (right). Adherence erodes in the well-organized memories that keep adding files and holds for the gpt-5.4 management agent's memory (Figure A10, Section E.3).

4.1.2 Organization by the agents' backbones

Does a stronger management agent organize better, or only differently? Table A6 and Table A8 answer for both settings. On the same 128k conversation the three backbones leave three organizations (Figure A1): `gpt-5.4-nano` shards into 122 files no deeper than five levels with 57 cross-references, `gpt-5.4-mini` consolidates into two files nested six deep, and `gpt-5.4` shards again into 105 files nested seven deep and stitched by 233 cross-references. **Capability changes the organization decisively, but the change does not run in one direction with capability**: the weakest and the strongest agents shard, the middle one consolidates. On the ALFWorld chain the change does run in one direction: Table A15 gives 114, 45 and 16 files for the nano, mini and gpt-5.4 management agents, with 290, 99 and 43 sections, 62, 56 and 10 cross-references, and a mix that shifts from 89 skills and 25 notes under nano to 5 skills and 11 warning notes under gpt-5.4. **Where writing means distilling trajectories into procedures, a stronger agent writes fewer entries with fewer cross-references; where it means arranging content that already exists, capability sets the form without ranking it.** The contract metrics of Table A8 show that adherence does not follow capability either: in both settings the sharded memories have the most cohesive siblings (cohesion lift 4.5 to 5.0 for the nano and gpt-5.4 rebuilds against 2.5 for mini's, and 3.9 against 1.9 for the skill memories), and of the three skill memories the mini-built one tracks relatedness best (B4 .29). Finally, the management agent is not the only agent that shapes the memory. With the management agent held at `gpt-5.4-mini`, Table A15 shows the chain's memory ending at 50, 45 and 62 files under a nano, mini and gpt-5.4 search agent: **a different search agent induces different trajectories, and the same management agent turns them into a different memory.** Full reading: Section E.1.2.

4.1.3 THE HARNESS AS A LEVER ON ORGANIZATION

Does the harness change what the same agent writes? Table A7 holds the memory variant and the backbone fixed and varies only the tools: Center (the file tools of Section 3), Center+BM25 (ranked keyword search for both roles and two section-level write tools for the management agent added) and Shell (a bash shell). On PersonaMem 128k the two file harnesses fold the persona into one or two files while Shell shards it into 147; on the shorter LoCoMo the three land at 3, 29 and 39. **With nothing changed but the tools, a shell agent partitions long dialogue into two orders of magnitude more files; on the short LoCoMo the spread is one a single rebuild can produce.** Two LoCoMo Center+BM25 builds that differed only in a corrected tool description produced 2 and 29 files (Section E.1): **organization is a characteristic behavior, not a constant; a harness's reliable signature is its direction on long content, not a file count.** Sharding costs bytes, not dollars (Table A7). The embodied chain runs the other way. In Table A15 the same `gpt-5.4-mini` management agent ends the ALFWorld chain at 45 files under Center, 76 under Center+BM25 and 36 under Shell; the shell memory has the highest B4 of Table A8 (.37), whereas under Center+BM25 the memory keeps adding files along the chain, 29 after task 40 to 76 after task 140, while its B4 falls from .26 to .14 (Table A28). **In the embodied setting the shell consolidates and its layout mirrors relatedness best; ranked search makes the chain keep adding files and the fit decays. Bash offers no structured write tool, yet the skill schema held.** The direction therefore belongs to the model and the content, not to the tools: the same shell drives `gpt-5.4-mini` to shard dialogue and to consolidate procedures, and drives `claude-sonnet-5` and `gemma-4-31B-it` to consolidate dialogue, 1 and 6 files on PersonaMem 128k (Section F). **The harness is a lever on organization as strong as the backbone, but it sets that the organization changes, not which way.** Whether they matter for answers and success is read in Section 4.2.3 (full reading: Section E.1.3).

4.2 DOES ORGANIZATION PAY? (RQ2)

4.2.1 ANSWER QUALITY, TASK SUCCESS AND COST BY MEMORY VARIANT AND SCALE

Does the memory's organization change what the search agent answers, and at what cost? Table 1 summarizes correctness and search cost for the conversational variants; Table A9 in Section E.2.1 adds attribution. Closed-book scores far below every memory, so the questions need the memorized conversation, and attribution is 82 to 98 percent for every filesystem memory. Correctness has no consistent leader: the nearly free Foldered sessions ties or tops LoCoMo (86.1), REALTALK (77.6) and PersonaMem 128k (76.2), while the Agent-curated memory leads only LoCoMo and is the weakest memory on PersonaMem 32k (37.5 against the Verbatim dump's 78.1), where even Chunk retrieval (71.9) beats every reorganized or curated memory.[1] **No organization leads correctness everywhere, and the least organized structured memory leads most consistently.** The 32k collapse is a failure of representation, not of coverage: in all 13 inspected cases the missed gold fact is present in the curated memory, but curation left superseded preferences standing as present traits (Section E.2.8; Section 4.2.2 reads the rebuild). **Curation removed what the benchmark rewards, the latest state of a changed preference.** Search cost is where organization repays, and only at scale (Tables A10 and A17): the reorganizers cut the Verbatim dump's per-query cost by more than half on the two PersonaMem tiers (1.4 and 1.6 cents against 4.0 and 3.8) and the curated memory by a third, by making more calls that read far fewer tokens, while on the smaller LoCoMo and REALTALK memories every variant searches at parity. Against the management cost of Table A5, a reorganizer repays itself after roughly 40 to 160 PersonaMem queries per conversation and the curated memory

Table 1: Conversational setting: correctness (%) / search cost (cents per query) per cell, shaded by correctness; question counts 158, 32, 42 and 85, left to right; PM is PersonaMem, and PM 32k costs average its three conversations. Attribution, management cost and per-query effort: Tables A5, A9, A10 and A17.

Memory	LoCoMo	PM 32k	PM 128k	REALTALK
Closed-book	18.4 / 0.4	34.4 / 0.3	47.6 / 0.3	10.6 / 0.3
Chunk retrieval	81.6 / 1.4	71.9 / 1.1	66.7 / 1.3	71.8 / 1.3
Verbatim dump	84.2 / 2.4	78.1 / 4.0	69.0 / 3.8	77.6 / 3.2
Foldered sessions	86.1 / 2.4	62.5 / 2.9	76.2 / 3.5	77.6 / 3.6
Reorg. (preserve)	82.9 / 2.4	56.2 / 1.4	59.5 / 1.6	77.6 / 3.4
Reorg. (condense)	79.1 / 2.1	68.8 / 1.4	54.8 / 1.6	41.2 / 3.2
Agent-curated memory	86.1 / 2.4	37.5 / 2.5	66.7 / 2.6	75.3 / 3.1

[1] Cells rest on 32 to 158 questions and re-judging a fixed cell moved about two questions, so gaps of a few questions are suggestive; the LoCoMo ordering survives removing four defective golds (Section E.2.9).

after 170 to 1,100, against the 10 to 12 questions asked per 32k conversation and 42 at 128k, and never on LoCoMo or REALTALK. **Organization repays search cost only where the memorized content is large, and over many more queries than one benchmark asks.** The two reorganizers' split by benchmark is read in Section E.2.1.

Table 2: ALFWorld single chains (140 tasks, Curated skills+mem (GS), gpt-4.1-mini execution agent): success (%) and total cost per chain in dollars. The shared default, `gpt-5.4-mini` as management agent and search agent under Center, scores 76.4 at $17.5; each block changes one factor; model labels are gpt-5.4 variants (full results and memory sizes: Table A15).

Management agent			Search agent			Harness		
nano	75.0	$10.3	nano	73.6	$14.4	Center+BM25	75.0	$15.0
5.4	89.3	$34.6	5.4	80.0	$22.5	Shell	82.9	$14.9

The embodied setting asks the same question of the execution agent. In the three-chain runs of Table A11 (Section E.2.2) the order inverts with the execution agent: under `gpt-4.1` the verbatim Episode log leads (87.1 against Curated skills+mem (GS)'s 82.1; net $+7$ of 140 tasks, $p \approx 0.23$), under `gpt-4.1-mini` Curated skills+mem (GS) leads by ten points (76.4 against 66.4; net $+14$, $p \approx 0.02$), and the tier drop raises the execution agent's invalid-action rate from 19 to 36 percent under the log against 12 to 22 under GS. **Raw transcripts demand an execution agent strong enough to digest them, guidance written for the task degrades gracefully, so which memory serves best depends on who consumes it.** The inversion reproduces under end-to-end accumulation (Table A15), on ScienceWorld (Table A12) and when both execution agents read the same frozen memory (Section E.2.7), so on ALFWorld it belongs to the pairing, not to whose trajectories built the memory. It is not universal: the frozen ScienceWorld comparisons differ from the full chains (Sections E.2.7 and E.2.12), and with `claude-sonnet-5` in both memory roles the curated memories beat the log at both tiers in both environments (Section F). Cost favors the same memory: Curated skills+mem (GS) is the least expensive to deploy on both tiers and costs less per solved task than no memory (5.7 against 6.1 cents at mini, 15.1 against 16.9 at `gpt-4.1`; Table A23), and building it, about ten dollars per 140 tasks, is the extra, the same trade of management cost against search cost as in conversation. **In both settings the value of organization is conditional, on the scale of the memorized content in conversation and on the execution agent in embodied tasks, and in neither does the most organized memory lead outright.** Full readings: Sections E.2.1 and E.2.2.

4.2.2 Quality, success and cost by the agents' backbones

Does a stronger backbone improve answers, and in which agent? Table 3 shows the backbone-capability grid on the PersonaMem 128k conversation (Tables A13 and A14 in Section E.2.3). With the `gpt-5.4-mini` search agent fixed, the nano, mini and gpt-5.4 rebuilds score 73.8, 66.7 and 71.4, three questions of 42 apart and not in one direction with capability, on memories of 122, 2 and 105 files; the strongest agent makes twice the build calls and leaves the memory that is the most expensive to search. **The management agent's strength shows in the organization, not in the answers.** Varying the search agent over the same three memories, mean correctness rises from about 62 to 71 to 79 across its tiers, and the strongest search agent scores best on the memory the weakest management agent built (83.3) and worst on the most elaborate one (71.4). **Capability matters in the agent that answers the question, not in the agent that builds the memory, and the most elaborate memory helped under no search agent.** The search agent's backbone moves a query's price twentyfold (under a cent at nano, 10.0 to 18.5 cents at gpt-5.4) and the memory at most twofold within one search agent, the two-file memory halving the strongest search agent's cost at close correctness (81.0 against 83.3): **the search agent's backbone is the dominant cost axis; organization still sets a factor of two inside it.** The management agent's capability does count where the build itself failed: rebuilding the collapsed PersonaMem 32k memories with a `gpt-5.4` management agent and the same search agent raises

Table 3: Backbone-capability grid on the PersonaMem 128k conversation (42 questions): rows vary the search agent, columns the management agent whose memory is searched (file counts), all gpt-5.4 variants; cells give correctness (%) / search cents per query at each search agent's own rates, shaded by correctness (full tables: Tables A13 and A14).

Search agent	nano (122 files)	mini (2)	5.4 (105)
nano	64.3 / 0.9	59.5 / 0.7	61.9 / 0.8
mini	73.8 / 3.2	66.7 / 2.6	71.4 / 3.8
5.4	83.3 / 18.5	81.0 / 10.0	71.4 / 15.3

correctness from 37.5 to 56.3 (Section E.2.8). **The search agent compensated for every arrangement in this grid but not for what a rewrite destroys, so the management agent's capability counts in the fidelity of the record more than in its layout.**

In the embodied setting the roles reverse. Varying the management agent with the search agent and execution agent fixed (Table 2; Table A15 in Section E.2.4) gives 75.0, 76.4 and 89.3: nano and mini do not differ detectably (net -2, $p \approx 0.85$) at a seventh of the management cost, and gpt-5.4 adds 13 points (net $+18$, $p \approx 0.001$), concentrated in the cool and heat families. All three memories passed the schema audit, and their procedural content nevertheless differed: nano filed twelve heating entries and scored 6 on heat, gpt-5.4 wrote two dense ones and scored 75. The gpt-5.4-curated GS chain reaches 89.3 with the weak execution agent, against 84.3 for the mini-curated GS chain with the strong one and one task above the strong execution agent's raw-log result of 88.6, a comparison that changes both the memory and the execution agent. **In the embodied setting the management agent's capability gain is a step at the top tier: every backbone meets the format, only the strongest memory distills procedures that transfer, and that memory carries the weak execution agent past the strong one's mini-curated results.** Varying the search agent instead, with the management agent fixed, gives 73.6, 76.4 and 80.0, neither step significant and net $+9$ end to end ($p \approx 0.09$). **Which agent's capability counts depends on what writing means: arranging content that already exists rewards the search agent, distilling trajectories into procedures rewards the management agent at the top tier; and in neither setting does organization alone explain value**, since conversation varies organization wildly at flat quality and the two weaker management agents of the embodied study do not differ detectably across 114 against 45 files. Full readings: Sections E.2.3 and E.2.4.

4.2.3 Quality, success and cost by the harness

Does the harness that reshapes the memory also change what comes out of it? With the memory variant and both backbones fixed (Table A16 in Section E.2.5), correctness is 86.1, 86.7 and 86.1 on LoCoMo under Center, Center+BM25 and Shell, and 66.7, 64.3 and 61.9 on PersonaMem 128k, a question or two of 42 apart, within run-to-run variation (Section E.1). **Memory organization changes a hundredfold across harnesses, from one or two files to 147, but answer correctness does not.** Search cost per query stays between 2.1 and 3.4 cents (Table A7), the search agent adapting its strategy to each layout (Section E.2.5). In the embodied setting the harness moves success (Table 2). Adding ranked search for both roles and two section-level write tools for the management agent changes the search agent's behavior and the memory (76 files instead of 45) without a detected success gain (75.0 against 76.4; net -2, $p \approx 0.85$), whereas the shell changes the memory (36 files) and success together: 82.9, ahead of Center+BM25 (net $+11$, $p \approx 0.04$) and of Center (net $+9$, $p \approx 0.06$), with heat rising from 44 to 81 percent, at the lowest total cost of the three (14.9 against 17.5 dollars). **Adding tools changes behavior and organization but not outcomes; replacing the harness changes organization and success together, by as much as a step in the management agent's model (Section 4.2.2) and at lower cost.** Which direction a family takes is its own (Section F). **The harness is therefore a lever as strong as the backbone on organization everywhere and on success where memory guides action, with no detected effect on conversational correctness, which no organization moved.** The full reading is in Section E.2.5.

5 Conclusion

We formalized filesystem-based agent memory as one filesystem operated by three agent roles and asked whether an agent can keep a growing memory organized and what organizing gains. The organization an agent produces is a signature of the model and the harness, and its value is conditional: reorganization lowers search cost on the two PersonaMem tiers but no organization is consistently most accurate, and in the embodied setting curated skills beat raw experience for every execution agent when `claude-sonnet-5` both writes and retrieves them, while the default family's curated memory loses to raw experience under the strong execution agent. Capability matters in the agent that reads on conversation and in the agent that writes in the embodied setting, and memory keeps its advantage over no memory as it grows within our horizons. Open questions are when benchmarks reward organization itself, whether that changes over longer horizons, and whether useful hierarchy can be induced where it lowers cost.

AI USE STATEMENT

We used generative AI assistants for implementing the experiment code and for drafting text, literature search, and figures and tables; we did not use them to create synthetic data or the conceptual framework, and proofs and translation are not applicable. AI assistants also gave feedback on experiment design, wrote data-processing scripts, and assisted the qualitative analysis and interpretation of results, all reviewed by the authors. We have reviewed all AI-assisted work, including the code, the reported results and the citations. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Every setting behind every reported cell is stated in Section B: the configuration of each run, the benchmark data and evaluation sets (Section B.2), the prompt families and their variants (Section C), every tool's parameters and description as sent (Section B.4), the skill-setting and ScienceWorld settings (Sections B.5 and B.6), the cross-family settings with their declared serving differences (Section B.7), and the run-integrity and provenance rules (Section B.8). The metrics, the cost accounting and the organization measures are defined in Section A.

ETHICS STATEMENT

This work uses only publicly released benchmarks (LoCoMo, PersonaMem, REALTALK, ALFWorld, ScienceWorld) under their published terms, recruits no new participants and collects no new personal data; REALTALK contains human conversations released by its authors, which we use as released. Filesystem memories of the kind studied here store what users tell an agent, so deployments carry the privacy obligations of any persistent user memory; the paper's contribution is a measurement of how such memories are organized and used, not a system for collecting them.

REFERENCES

Gemma Team Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cuarbune, Michelle Cas-bon, Mayank Chaturvedi, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, Johan Ferret, Filippo Galgani, Sertan Girgin, Jean-Bastien Grill, Maarten R. Grootendorst, Jiaxian Guo, Cassidy Hardin, Yanzhang He, Steven M. Hernandez, Omri Homburger, L'eonard Hussenot, Juyeong Ji, Armand Joulin, Aishwarya B Kamath, Parnian Kassraie, Olivier Lacombe, Preethi Lahoti, Gael Liu, Gus Martins, Luciano Martins, Tatiana Matejovicova, Ramona Merhej, Nikola Momchev, Sneha Mondal, Ryan Mullins, Sindhu Raghuram Panyam, Shreya Pathak, Sarah Perrin, André Susano Pinto, Etienne Pot, Angéline Pouget, Alexandre Ram'e, Sabela Ramos, Doug Reid, David Rim, Morgane Rivière, Karsten Roth, Louis Rouil-lard, Omar Sanseviero, Pier Giuseppe Sessa, Shane Settle, Danila Sinopalnikov, Sara Smoot, Piotr Stańczyk, Andreas Steiner, Lawrence Stewart, Ilya O. Tolstikhin, Michael Tschannen, Anton Tsitsulin, Nino Vieillard, Renjie Wu, Ping mei Xu, Hai-Tao Yang, Edouard Yvinec, Li Zhang, Joe Zou, Nicolas Aagnes, Abdelrahman Abdelhamed, Shivani Agrawal, Shubham Agrawal, Ibrahim Alabdulmohsin, Jean-Baptiste Alayrac, Uri Alon, Chandramouli N. Amarnath, Ankesh Anand, Chrysovalantis Anastasiou, Setareh Ariafar, Franccois-Xavier Aubet, Kyriakos Axiotis, Federico Barbero, Joelle Barral, Alexei Bendebury, Urs M. Bergmann, Stanley M. Bileschi, Kat Black, Mathieu Blondel, Sebastian Borgeaud, Arthur Bravzinskas, Ryan Burnell, Róbert Istvan Busa-Fekete, Mu Cai, Glenn Cameron, Char lotte Caucheteux, Garima Chadha, Jetha Chan, Aditya Chawla, Blake Jianhang Chen, Jesse Chen, Lin Chen, Xu Chen, Derek Zhiyuan Cheng, Tzu hsiang Chien, Nikolai Chinaev, Ying-Fen Chou, Zhaohui Chu, Benjamin Coleman, Pooja Consul, Sam Conway-Rahman, Scott Crowell, Dylan Cutler, Vivek Dani, Samira Daruki, Anil Das, Daniel Deutsch, Nishanth Dikkala, Linyi Ding, Qiuhan Ding, Shenil Dodhia, Konstantin Donhauser, Tulsee Doshi, Anca D. Dragan, Alex Druinsky, Sahil Dua, Zoltan Egyed, Danielle Eisenbud, Daniel Eppens, Cindy Fan, Bahare Fatemi, Yassir Fathullah, Vladimir Feinberg, Milen Ferev, Takumi Fujimoto, Isaac R. Galatzer-Levy, João Gante, Simon Geisler, Soham Ghosal, Antonious M. Girgis, Alec Go, Alhaad Gokhale, Alex Grills, Yiming Gu, Pramod Gupta, Guru Guruganesh, Raia Hadsell, Hamza Harkous, Jitendra K. Harlalka, Demis Hassabis, Anja Hauth, Joseph Heyward, Arian Hosseini, Chih-Yang Hsia, I-Hung Hsu, Xiao-Peng Huang,

Yangsibo Huang, Kevin Hui, Adrian Hutter, I Te, Fotis Iliopoulos, Advait Jain, Ganesh Jawahar, Ziwei Ji, Qilin Jin, Melvin Johnson, Kandarp Joshi, Arun Kumar Reddy Kandoor, Wang-Cheng Kang, Koray Kavukcuoglu, Mehran Kazemi, Kathleen Kenealy, Amr Khalifa, Phoebe Kirk, Suraj Kothawade, Vitaly Kovalev, Neel Kovelamudi, Adam Kraft, Ravin Kumar, Harish Kuppam, Justin Lannin, Chen-Yu Lee, Seungjin Lee, Dmitry Lepikhin, Dong dong Li, Qiujia Li, Valentin Liévin, Ethan Lin, Ziqian Lin, Casper Liu, Tianlin Liu, Tianqi Liu, Xin Liu, Mayank Lunayach, Min Ma, Gagan Madan, Andrii Maksai, Eric Malmi, Michal Matuszak, Daniel McDuff, Gaurav Menghani, Daniil Mirylenka, Karolis Misiunas, Vedant Misra, Andreea-Maria Mitran, Kareem Mohamed, Maksim Mukha, Eric Noland, James O'Donnell, Kate Olszewska, Bernett Orlando, Wanqiong Pan, Rina Panigrahy, Unnati Parekh, Chunjong Park, Eric Paskie, Liqian Peng, Bryce Petrini, Slav Petrov, Jonas Pfeiffer, Bilal Piot, Martyna Beata Płomecka, Siim Põder, Octavio Ponce, Arijit Pramanik, David Racz, A. Peter Rajan, Michelle Tadmor Ramanovich, Anand Rao, Marvin Ritter, Vitor Rodrigues, Evan Rosen, Mikolaj Rybi'nski, Noveen Sachdeva, Michael Eli Sander, Rohit Sathyanarayana, Sagar Savla, Samuel Schmidgall, Tal Schuster, Benoit Seguin, Andrew B. Sellergren, Aliaksei Severyn, Izhak Shafran, Dhruv Shah, Shangguan Yuan, Ashish Shenoy, Pradeep Shenoy, Rakesh Shivanna, Pauline Sho, Lucas Spangher, Wojciech Stokowiec, Tim Strother, Yao Su, Yinghao Sun, Mukund Sundararajan, Andrea Tacchetti, Mor Hazan Taege, Pouya Dehghani Tafti, Chetan Tekur, Rahul Thapa, Madeleine Traverse, Lenart Treven, Tao Tu, Chien Te Tung, Petar Velivckovi'c, Malini Pooni Venkat, Sagar Gubbi Venkatesh, Vidya Venkiteswaran, Francesco Visin, Alex Vitvitskyi, Kiran Vodrahalli, Weiyi Wang, Xin Wang, Tris Warkentin, Jan Wassenberg, John Wieting, Lechao Xiao, Hao Xu, Yuhui Xu, Fuzhao Xue, A. Yadav, Jun Yan, Antoine Yang, Lin F. Yang, Ming-Hsuan Yang, Ziyu Ying, Jae Hyeon Yoo, Sajjad Hussain Zafar, Fred Weiying Zhang, Jia-Geng Zhang, Jianyi Zhang, Xiaofan Zhang, Chaoran Zhao, David Zhou, and Chenjie Zou. Gemma 4 technical report. *ArXiv*, abs/2607.02770, 2026. URL https://api.semanticscholar.org/CorpusID:289923375.

Anthropic. Memory tool. Claude Platform Docs, https://platform.claude.com/docs/en/agents-and-tools/tool-use/memory-tool, 2025a. Tool type memory_20250818; generally available on the Messages API. Accessed 2026-07-03.

Anthropic. Introducing Agent Skills. https://claude.com/blog/skills, October 2025b. Product announcement, October 16, 2025.

Anthropic. How Claude remembers your project. Claude Code documentation, 2026a. URL https://code.claude.com/docs/en/memory. Claude Code Docs.

Anthropic. Dreams (Claude Managed Agents documentation). Claude Platform Docs, 2026b. URL https://platform.claude.com/docs/en/managed-agents/dreams. API beta header dreaming-2026-04-21.

Anthropic. Introducing Claude Sonnet 5. https://www.anthropic.com/news/claude-sonnet-5, June 2026c. Product announcement, June 30, 2026.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Generating instruction manuals by LLM agents via interactive environmental learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=Pwl9n4zlf5.

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.

Bowen Jiang, Zhuoqun Hao, Young Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle Ungar, Camillo Jose Taylor, and Dan Roth. Know me, respond to me: Benchmarking LLMs for dynamic user profiling and personalized responses at scale. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=6ox8XZGOqP.

Dong-Ho Lee, Adyasha Maharana, Jay Pujara, Xiang Ren, and Francesco Barbieri. Realtalk: A 21-day real-world dataset for long-term conversation. *arXiv preprint arXiv:2502.13270*, 2025.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe

Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401, 2020. URL https://api.semanticscholar.org/CorpusID:218869575.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36:23813–23825, 2023.

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. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.

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)*, pp. 13851–13870, 2024.

OpenAI. AGENTS.md. https://agents.md/, 2025. Related: https://learn.chatgpt.com/docs/agent-configuration/agents-md.

OpenAI. Dreaming: Better memory for a more helpful ChatGPT. Blog post, June 2026. URL https://openai.com/index/chatgpt-memory-dreaming/. Published June 4, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Jianfeng Gao. On memory construction and retrieval for personalized conversational agents. *ArXiv*, abs/2502.05589, 2025. URL https://api.semanticscholar.org/CorpusID:276249662.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=4NJBV6Wp0h.

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

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. In *International Conference on Learning Representations*, volume 2025, pp. 990–1023, 2025.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2010.03768.

Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=1i6ZCvflQJ. Survey Certification, Featured Certification.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=NTAhi2JEEE.

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.

Dylan Zhang, Yanshan Lin, Zhengkun Wu, Yihang Sun, Bingxuan Li, Dianqi Li, and Hao Peng. Useful memories become faulty when continuously updated by llms. *arXiv preprint arXiv:2605.12978*, 2026.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Trans. Inf. Syst.*, 43(6), September 2025. ISSN 1046-8188. doi: 10.1145/3748302. URL https://doi.org/10.1145/3748302.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners, 2024. URL https://arxiv.org/abs/2308.10144.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=uccHPGDlao.

Wanjun Zhong, Lianghong Guo, Qi-Fei Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. *ArXiv*, abs/2305.10250, 2023. URL https://api.semanticscholar.org/CorpusID:258741194.

Sizhe Zhou and Jiawei Han. A simple yet strong baseline for long-term conversational memory of llm agents. *arXiv preprint arXiv:2511.17208*, 2025.

APPENDIX

Contents of the appendix

A Definitions and Metrics **15**

A.1 The taxonomy contract in full 15

A.2 Memory organization metrics 15

A.3 Cost accounting 16

A.4 Lifecycle measures 16

A.5 Sign tests 16

B Experimental Setup Details **16**

B.1 Configuration 16

B.2 Benchmark Data Provenance 19

B.3 Fixed Search Prompts 20

B.4 Tool Descriptions and Schemas 21

B.5 Skill-Setting Configuration 23

B.6 ScienceWorld setup 24

B.7 Cross-family setup 25

B.8 Run integrity and provenance 26

C Prompts **27**

C.1 Management agent (builder) prompt 27

C.2 Reorganizer prompt (Reorganized memory) 30

C.3 Search agent prompts (one per memory variant) 33

C.4 Judge prompts (answer grading) 40

C.5 Skill-setting prompts 41

D Example Memory Filesystems **51**

E Supporting Results for RQ1 and RQ2, and the Growth Study (RQ3) **53**

E.1 Supporting results for RQ1 53

E.2 Supporting results for RQ2 58

E.3 Does a memory stay useful and organized as it grows? (RQ3) 73

F Generality Across Model Families (RQ4) **83**

G Use of AI Assistants for Illustrative Figures **108**

Table A1: The taxonomy contract: five principles a well-organized memory must satisfy at every level of the tree, headings included; Section A.2 defines their measurements.

Principle	Statement
P1. Sibling distinction	*Siblings are distinguishable by labels alone*: items under one parent can be told apart by name, at worst name plus description, without opening bodies.
P2. Sibling relatedness	*Siblings belong together*: what shares a parent is related enough that sharing it is natural.
P3. Parent-child coverage	*A parent covers its children*: everything under a parent falls within what its name declares and, as far as practical, everything in its scope lives under it, so descending narrows the search without losing the sought fact, an exhausted subtree is conclusive, and each child is strictly more specific than its parent.
P4. Tree-wide proximity	*Distance mirrors relatedness*: the more related two pieces of content, the nearer they sit in the tree.
P5. Structural economy	*Structure serves the search, not itself*: depth is added only where it improves routing to a fact; a level that does not help routing is overhead.

A Definitions and Metrics

A.1 The taxonomy contract in full

Section 2.1 states the five principles of the taxonomy contract in brief; Table A1 gives them in full, as both settings' management instructions state them (Sections C.1 and C.5). They bind at every level of the tree, headings included, in both settings.

A.2 Memory organization metrics

The shape panel of Table A6 measures each memory's *combined tree*: root, folders, files, then markdown headings, with every heading level adding depth (a heading that skips levels adds one). The extraction and sampling conventions below describe the task-40 and task-140 adherence analysis; the growth curves and the static hierarchy table retain their original estimates. For the same final mini-curated GS memory the growth curve, the static table and the adherence table report B4 values of .32, .29 and .30, so each estimate is read within its own analysis, and differences between analyses are not evidence of memory evolution. Content units are leaves: sections without subsections, or files without headings. The content metrics retain units longer than 40 characters and use at most the first 400 eligible leaves in tree-traversal order. They use lexical TF-IDF cosine similarity without a learned component; text attached to internal headings and file preambles is outside this leaf-based comparison. Panel A is shape: counts, mean and maximum leaf depth (mean leaf depth is the Sackin index normalized by leaf count), fanout, content-size concentration, and cross-reference count. Panel B operationalizes the taxonomy contract both management prompts prescribe: with unit vectors v_u (TF-IDF) and label vectors ℓ_n (folder name, file basename or heading text): B1, sibling label distinguishability, the mean of $1 - \cos(\ell_a, \ell_b)$ over sibling pairs $a \neq b$, averaged over internal nodes (its minimum flags the worst confusable pair); B2, sibling content cohesion, is the mean lexical similarity of same-parent unit pairs divided by the mean for sampled cross-parent pairs. The numerator uses at most the first 20 eligible units per parent. The denominator samples cross-parent pairs with seed 0, targeting at least 200 pairs or the numerator pair count, whichever is larger, with at most 20,000 draw attempts; B3, scope leakage, the fraction of units u with $\max_{g \neq g(u)} \cos(v_u, \mu_g) > \cos(v_u, \mu_{g(u)})$, where μ_g is the normalized sum of the vectors in parent group g and $g(u)$ is u's own group. A unit contributes to its own centroid. Rival groups must contain at least two eligible units; the denominator includes units with an eligible rival. The candidate groups need not share a grandparent. This is a lexical placement measure, not a semantic-error rate; and B4, the Spearman correlation between tree distance $d_T(u, u')$ (path length through the lowest common ancestor) and content distance $1 - \cos(v_u, v_{u'})$ over at most 4,000 pairs sampled with seed 0 from the retained units. Pair selection depends on unit order, and TF-IDF is fitted

separately to each snapshot. The fifth contract property, structure serves search, is deliberately measured functionally by the retrieval effort of Sections 4.1.3 and 4.2 rather than by a static formula. Two honesty notes. These are descriptors of contract adherence, not quality scores: whether high adherence predicts retrieval value is itself an open question. And B4 tends to rise with granularity (many small topical files give tree distance more signal to carry), which is visible in Table A8: the sharded 128k builds carry higher B4 than the consolidated two-file memory whose structure lives in headings. B3 catches content-level misplacement but not label-level misnesting; a parent-child label-consistency check is the natural refinement.

A.3 Cost accounting

Token usage is logged per model call in three categories (uncached input, provider-cached input, and output, with reasoning tokens as a sub-count of output), so any pricing can be applied; dollar and cent figures are single-run measurements, so they carry the run-to-run variation of episode lengths and of the provider's cache behavior. The cent and dollar figures in Tables A5 and A17 apply the deployed `gpt-5.4-mini` list rates, the model serving every role in the main grid: \$0.75 per million uncached-input tokens, \$0.075 per million cached-input tokens, and \$4.50 per million output tokens.

Intrinsic compute bounds. For an episode with rounds $r = 1, \ldots, R$, let p_r, q_r, and o_r be its prompt, provider-cached, and output token counts at round r (all read from the per-call usage records). Summed over a cell's episodes,

$$\text{no-cache} = \sum_r p_r + \sum_r o_r, \quad \text{measured} = \sum_r (p_r - q_r) + \sum_r o_r, \quad \text{perfect} = p_R + o_R, \quad (4)$$

and the efficiency share of Table A25 is $(\text{no-cache} - \text{measured})/(\text{no-cache} - \text{perfect})$. No-cache is the ceiling that re-prefills every round's full prefix; measured counts uncached-input and output tokens; it is a token-workload proxy, not dollar cost or a FLOP estimate. For an append-only executor transcript, perfect is the final transcript's token count, in which every token is prefilled once and every output decoded once: because round R's prompt contains the whole conversation, including every earlier output, the final prompt plus the final output count each transcript token exactly once. We validated the implementation by reproducing two cells' published rows exactly from their raw per-call records before computing any new row.

A.4 Lifecycle measures

The single-chain runs snapshot memory after every task, recording each file's path and content hash. The plotted lifecycle analysis compares snapshots at task 35 and task 140. A task-35 file is classified as unchanged at the same path, present at the same path with changed content, present under a different path with the same hash, or absent by both its original path and hash. These endpoint categories do not distinguish every deletion-and-recreation event or every move followed by an edit, and they do not measure factual retention. The event curves count net creations, changes, moves and deletions between consecutive task snapshots. Management effort is measured separately from each episode's tool rounds and output tokens.

A.5 Sign tests

Where two cells are compared on the same items (the questions of a benchmark, the 140 tasks of a chain, or the 70 held-out tasks of the trajectory-origin study), we pair the items, count the wins and losses of one cell over the other, drop ties, and report the two-sided exact binomial probability of a split at least as uneven under equal chances. The count of wins and losses is reported with the probability. For accumulating chains, these item-level tests describe the observed paired outcomes; tasks share an evolving memory and are not independent repeated training runs.

B Experimental Setup Details

B.1 Configuration

File anatomy and the roles' composition, in full. Descriptions matter because directory listings and ranked search expose them: with names, they are what an agent sees before opening a file.

Table A2: The memory variants of both settings, described in full (Section 3 names each in one line). Conversational variants build from the same conversation; only Closed-book keeps no memory, and the four filesystem variants are searched under the contract of Equation (2).

Variant	Description
Conversational setting	
Closed-book	Answers each question directly, with no memory and no retrieval, measuring the floor the model reaches on its own knowledge.
Chunk retrieval	Stores the stream itself: the conversation is split into chunks (chunking rule in Section B.1), one file per chunk, indexed for ranked keyword search (BM25), and its search agent has a single tool that returns the top 3 chunks in full for each query it issues, the standard retrieval-augmented control (Lewis et al., 2020).
Verbatim dump	Copies the stream into the memory wholesale: one file per session, content verbatim, each file's description stating only the session's number, date, and speakers; a deterministic flat memory built at zero model cost, the zero-curation baseline.
Foldered sessions	Has an LLM design a folder taxonomy into which the Verbatim dump's session files are moved intact (move only, zero byte edits), so the taxonomy is the only signal added over the Verbatim dump.
Reorganized memory	Has an LLM restructure the Verbatim dump, splitting and merging content across sessions into files whose grouping the model chooses, so the layout changes. We run it as two versions that differ by a single instruction, which lets us separate compression from layout. Its default behavior turns out to be lossy: without an explicit rule the memory shrinks markedly across the reorganization pass on three of the four benchmarks (Table A5), the model dropping detail as it rewrites. We label this default the Reorg. (condense) version; adding one "keep every fact" rule gives the Reorg. (preserve) version, which holds content fixed on the dialogue benchmarks but not on PersonaMem (Table A5). The condensing version therefore captures not an instruction to compress but the backbone model's own tendency when it reorganizes freely, and the preserving version is the intervention that counteracts it.
Agent-curated memory	Built from empty over the stream by the management agent of Equation (1), which decides both what to write and how to organize it; unlike the constrained variants above, it varies content and organization together.
Embodied setting	
No memory	Attempts every task without memory, the execution agent's floor.
Episode log	Appends each task's rendered trajectory to the memory verbatim at zero model cost, the zero-curation baseline; retrieval serves whole episode files.
Curated skills	Has the management agent of Equation (1) distill attempts into procedure files, and retrieval cites the files whose contents are placed in the execution agent's context.
Curated skills+mem	Adds a second entry kind to the curated memory: alongside skill files it holds *memory notes* (lessons, warnings, observations), with an outcome gate (only a successful attempt may create or extend a positive procedure) and a stricter rule for what retrieval may serve: entries are cited only when they clearly apply to the task, rather than liberally.
Curated skills+mem (GS)	Keeps that memory and changes what retrieval returns: instead of serving files, the search agent reads the memory and writes task-specific guidance (*guidance synthesis*, GS), so the execution agent sees synthesized text rather than raw entries.

We write $\varnothing$ for the empty memory. In our instantiation each file is a markdown document that is required to open with structured frontmatter carrying its name and the one-line description d_f, and that may carry additional optional frontmatter fields (free-form metadata). This frontmatter schema is a choice of our implementation; the memory tool itself exposes generic file operations (Anthropic,

2025a). Figure 2 dissects two files of one such memory. Nothing below depends on this choice, and any hierarchical filesystem with per-file descriptions satisfies the definitions.[2]

Figure 2 shows this anatomy on a declarative memory file and a skill file: YAML frontmatter (`name`, `description`), which listings and search expose first; markdown headings that continue the taxonomy inside the file; inline source locators (`[S6T5]` reads session 6, turn 5); cross-references instead of copied content; and an optional `metadata` field for the file kind.

How the roles compose. When an execution agent is present, the other two serve it: the search agent acts as its retrieval subroutine, and its trajectories become the chunks the management agent integrates, whether the execution agent invokes management directly or its logged traces are handed over after the fact. When no execution agent exists, as in the conversational instantiation, management and search operate on their own: the stream comes directly from the environment and queries are posed externally. The contracts are deliberately minimal so the roles map onto deployed harnesses: in a coding agent, one model may play all three, and the instruction ι can be a fixed constant. The roles isolate the two capabilities this paper studies, writing memory well and reading it well, without prescribing how either is implemented.

The main text's setup, in full. The paragraphs below carry the complete wording of the experimental setup, of which Section 3 keeps the essentials. *Memory variants.* Six memory variants span the space from no memory to an agent-curated one. Only Closed-book keeps no memory; every other variant builds one from the same conversation, and the four filesystem variants are searched under the contract of Equation (2); Table A2 describes each. *Models and judge.* The management agent and search agent roles run `gpt-5.4-mini` at high reasoning effort, the center of a `gpt-5.4-nano` to `gpt-5.4` capability range; the other two models appear in the backbone-capability studies. The foldering and reorganization passes also run `gpt-5.4-mini` at high effort. Grading uses one judge model (Zheng et al., 2023) with one fixed prompt throughout: `gpt-5.4-mini` at low reasoning effort, with evaluation configuration fixed across cells and the provider's default sampling under reasoning. Before any experiment ran, we calibrated the judge on held-out validation answers and froze its configuration for every judged cell; that the judge shares a vendor with the default family is a limitation (Panickssery et al., 2024). To test whether the findings depend on this family, Section F re-runs the management agent and search agent with `gemma-4-31B-it` (Abd et al., 2026) and `claude-sonnet-5` (Anthropic, 2026c), under the settings of Section B.7, which declares the serving differences. *Backbone-capability, harness, and scale studies.* Each study changes one thing relative to the setup above and holds everything else fixed. The management agent backbone-capability study adds two rebuilds of the PersonaMem 128k memory, with `gpt-5.4-nano` and `gpt-5.4` as the management agent beside the main `gpt-5.4-mini` build, while the stream, the instruction, and the search agent stay the same. The search agent study then has each of those three models read each of the three memories: a three-by-three grid whose center cell is the main setup. The harness study keeps the agent-curated memory setup and the model, and changes only the tools through which memory is written and read; the three harnesses carry the same names in both settings: Center, the default tools; Center+BM25, which adds ranked keyword search (Robertson & Zaragoza, 2009) for both roles and two section-level write tools for the management agent; and Shell, which replaces the tools with a bash shell over the same memory. Scale is compared through PersonaMem's 32k and 128k tiers. The embodied setting repeats the capability axes: its management agent study swaps the management agent's model while the execution agent and search agent stay the same, its search agent study swaps the search agent's backbone over the same three tiers with the management agent fixed, and its two execution agent tiers vary execution strength; and its two kinds of runs give the scale contrast, the same tasks run with memories that accumulate over one 140-task chain against three shorter chains (Section E.3). Every study records its configuration like the main runs. *Metrics.* Outcome quality is measured per setting. On the conversational benchmarks each answer is graded for correctness against gold and for attribution, whether the cited memory actually supports it, both by the judge; on PersonaMem correctness is judged option matching. On the embodied setting the environment itself reports task success, read as overall and per-family rates, and on ScienceWorld also its task

[2]For simplicity we take each memory entry to be a single Markdown file. Richer packagings exist: in Anthropic's skill definition, a skill is a folder containing a `SKILL.md` together with optional subfolders of auxiliary context or scripts (Anthropic, 2025b). Our definitions extend naturally to such folder-valued memories, but we do not study that variant.

score, which credits the subgoals an episode completes, and the number of episodes ending at the action cap. Cost and effort are counted per model call in token categories (uncached input, provider-cached input, output with reasoning as a sub-count) together with tool rounds and calls, aggregated per query, per chunk, or per task; every input token is tallied whether or not the provider served it from cache, and Tables A5 and A17 price the totals at stated rates. From the same per-call records we compute each cell's intrinsic compute bounds, the perfect-cache floor and the no-cache ceiling that bracket its measured spend (Section A.3). Memories are measured directly: file, folder, and section counts, sizes, and cross-references; the tree-shape and taxonomy-adherence metrics of Table A6; and, in the single-chain runs, the per-task memory trajectory. Definitions and equations for the compute bounds and the shape metrics are in Sections A.2 and A.3. Where two variants are compared on the same items we report paired differences with sign tests.

Table A3 pins the configuration behind every reported cell of the main comparison (Tables A5, A9 and A17); the scale, harness, and model-backbone-capability studies reuse it and state only their deltas (Section 3). Every role in the main grid runs `gpt-5.4-mini`: the search agent, every memory-building management agent (curation, foldering, and reorganization), and the judge. Calibration and pilots ran on validation splits; every reported number comes from test splits. A mid-run API failure aborts the whole cell rather than shipping a partial result on the conversational and ALFWorld cells, and every reported `gpt-5.4-mini` cell there completed in full; the ScienceWorld cells and the replicated families of Section F ran with fail-fast off, the former with zero errored tasks and the latter with their overflow units counted there.

Stream units. The build stream delivers the conversation as chunks of consecutive dialogue turns, at most eight per chunk and closed early once a chunk reaches 3,000 characters (Table A3). The management agent integrates one chunk per build episode, and Chunk retrieval indexes these same chunks as its retrieval units; the Verbatim dump and Foldered sessions memories instead keep whole sessions, one file per session (Section 3).

Generation and episode parameters. We avoid truncating agent inputs and outputs anywhere: file views are never clipped, and output caps exist only to stop runaway generations, set at values the runs do not reach in practice. The search agents and the judge decode under an 8192-token output cap; the curation management agent, whose steps carry more hidden reasoning, uses 32768, and the memory-restructuring passes, which emit whole-memory rewrites, use 100k. Truncation is zero across a 2,885-turn validation scan and every reported cell, apart from a single benign event on one foldered-memory search query. When an episode's context grows past 96k prompt tokens it is compacted (250k for the reorganization passes, whose whole-memory rewrites are larger): older turns are replaced by a running summary plus the three most recent turns. The same rule applies to every management agent and search agent episode, and compaction events are logged; the foldering pass runs under a 30-round cap. Building episodes for the agent-curated memory cap at 60 tool rounds; search episodes cap at 40 rounds for the curated and reorganized memories and 20 for the other variants (closed-book answers in one round). The 60- and 40-round caps are never reached (the deepest observed episodes use 36 build and 24 search rounds); one Chunk retrieval LoCoMo search episode hit that variant class's 20-round cap, and a re-probe at 40 rounds left its answer unchanged. Agents sample at the provider's high-effort setting with no overrides, the judge at low effort with the provider's default sampling (its configuration echoes a temperature of 0 that the request does not carry under reasoning).

B.2 Benchmark Data Provenance

The evaluation sets, in full. LoCoMo (Maharana et al., 2024) pairs long multi-session dialogues with questions about them and is the field's default long-memory testbed; we evaluate on a held-out test conversation, 158 questions spanning four categories: multi-hop, temporal reasoning, open-domain, and single-hop (the adversarial category excluded). Four questions whose gold answers contradict the transcript are cataloged in Section E.2.9; excluding them changes no conclusion. PersonaMem (Jiang et al., 2025) poses multiple-choice questions at checkpoints along a persona-rich stream in two of its three context tiers, 32k and 128k tokens; answers are graded by the judge as option matching against the gold option. Comparing across the tiers is only a rough scale test, since the tiers differ in which conversations they contain, not only in length. Both tiers are evaluated at each conversation's terminal checkpoint (32k: three conversations, 32 questions; 128k: one conversation,

Table A3: Resolved experimental configuration, common to every reported cell of the main comparison.

Setting	Value
Models and reasoning efforts	
Search agent (all cells)	`gpt-5.4-mini`, high effort
Management agent (all builds)	`gpt-5.4-mini`, high effort
Judge (all judged cells)	`gpt-5.4-mini`, low effort, default sampling
Runtime parameters	
Random seed (every conversational run)	`42`
Stream chunking	8 turns per chunk, 3,000-character cap
Search concurrency	8 questions in flight
Retrieval depth (Chunk retrieval)	top 3 chunks per query
Search and judge output caps	8,192 tokens
Curation build output cap	32,768 tokens
Reorganization build	3 passes, at most 80 rounds per pass
Reorganization, foldering completion cap	100,000 tokens
Tool-round caps	60 build; 30 fold.; 40 search (curated, reorg.); 20 others
Context compaction	summary + 3 recent rounds; 96k trigger (250k reorg.)
File-view cap	none; files are never clipped

42 questions). REALTALK (Lee et al., 2025) contributes 21 days of authentic human-to-human messaging (one test conversation, 85 questions), testing whether conclusions survive naturalistic dialogue.

LoCoMo. One held-out test conversation, `conv-50`. Of its 204 questions we evaluate the 158 non-adversarial ones (Section 3).

REALTALK. One conversation, `Chat_10_Fahim_Muhhamed`, with 85 questions; under the chunking of Table A3 its stream yields 93 build chunks. Data come from the official repository, pinned at commit `b903e06a`.

PersonaMem 32k. We evaluate the 32k-tier test split, deterministically downsampled to three conversations, each at its terminal checkpoint. The three samples, with question counts, are `pm_32k_18_246eaab7_e185` (10), `pm_32k_19_4b3812ac_e152` (10), and `pm_32k_19_947ec42f_e152` (12), 32 questions in total. Identifiers name the tier, persona, shared-context id, and terminal message index: the memory is built from the full message prefix up to that index, and the terminal question battery is asked against it.

PersonaMem 128k. One test conversation, `pm_128k_18_43642c89_e784`: persona 18, shared context `43642c89`, evaluated at its terminal checkpoint `e784`, a 784-message prefix (20 sessions, about 113k tokens) carrying a 42-question terminal battery that spans six of PersonaMem's seven question types.

B.3 Fixed Search Prompts

Pilot runs showed the search prompt alone can reorder memory variants, so all variants run one prompt family built from the same shared blocks, adapted only where the memory form requires it (a raw-session reading note for flat memories, for example). Every prompt requires the answer to be drawn from what the memory contains and cited to it, never from the search agent's own knowledge, and is fixed within each study; the full texts, and the variants each study used, appear in Section C. Filesystem variants cite file paths with line ranges (the scorer reads a line locator with or without its `L` prefix, `:L10` or `:10`); Chunk retrieval cites the chunk-file paths returned in result headers; turn locators remain part of the retrieved evidence.

Each memory variant runs one fixed search agent prompt from the quality-matched family of Section 3; Reorganized memory and Agent-curated memory share the hierarchical-memory prompt. The full prompt texts, and which variant uses which, appear in Section C.

B.4 Tool Descriptions and Schemas

The two roles see deliberately different harnesses. The management agent writes through the six generic file operations of the industry memory tool (Anthropic, 2025a) (view, create, string-replace edit, insert, delete, rename) plus line-level regex search: a write harness faithful to deployed practice. The search agent reads through a filesystem-native read-only set (view, regex search, table of contents, section read): tools that answer only what the memory's own names, layout, and text expose. A ranked search engine would find content however the memory is organized, hiding exactly the differences we study; file tools make the search agent navigate the organization, so those differences can show. This asymmetry is a design choice, not a finding; the harness study (Section 4.1.3) varies both harnesses. Every tool's parameters and description, exactly as sent, appear in Table A4.

Tools reach each agent through the model API's function-calling interface, separately from the system prompt. Table A4 reproduces, for every tool used by any reported cell, the parameters and the description exactly as sent. The management agent runs the seven-tool write set in both settings; the search agent runs the four-tool read-only set in both; Center+BM25 grants both roles ranked whole-file search on top of these and the management agent two section-level write tools, and Shell replaces both sets with the single `bash` tool over the same memory. The Chunk retrieval search agent has only ranked chunk retrieval, and the Foldered sessions pass runs view, grep, and rename alone.

Table A4: Every tool used by a reported cell, with its parameters and its description exactly as delivered to the models through the function-calling interface. Types are JSON-schema types; parameters not marked required are optional. The management agent set is the six industry memory-tool operations plus `grep` (Section 3); the search agent set is read-only; the last five rows are the Center+BM25 additions (two section-level write tools for the management agent and ranked whole-file search for both roles), the Chunk retrieval tool, and the Shell replacement.

Tool	Parameters	Description (as sent)
`view`	`path` (string, required): Memory path to directory or file to view (must be '/memories' or start with '/memories/'). Use '/memories' to see the root directory. `start_line` (integer): Starting line number (1-indexed). Only for files. `end_line` (integer): Ending line number (1-indexed, inclusive). -1 for end of file. Only for files.	View the contents of a memory file or list the contents of a memory directory. When viewing a file, contents are shown with line numbers (the YAML frontmatter block, if present, occupies the first lines). When viewing a directory, files and subdirectories are listed as full paths with sizes, and each file line shows the file's frontmatter `description`, rendered as `[description: ...]`; the listing shows entries up to 3 path levels below the directory, and directory sizes count their full contents including deeper files not listed.
`grep`	`pattern` (string, required): Regex pattern to match against each line. Matched per line; cannot span multiple lines. `path` (string): Memory path to search: a directory (all .md files under it are searched recursively) or a single .md file (must be '/memories' or start with '/memories/'). `case_sensitive` (boolean): Whether the search is case-sensitive. `max_results` (integer): Maximum number of matching lines to return.	Grep-like line search across .md files in memory. Matches a regex pattern against each line individually (does not match across lines); every line is searched, including YAML frontmatter lines. Returns matching lines with file paths and line numbers.
`create`	`path` (string, required): Memory path for the new file (must end with .md) (must start with '/memories/'). `file_text` (string, required): Content to write to the new file. The first lines must be a YAML frontmatter block where `name` equals the filename stem (kebab-case slug). For example, creating `/memories/alice-preferences.md` should start with: `---\nname: alice-preferences\ndescription: Alice's taste in music and food.\n---\n\n# Music\n....` The optional `metadata:` key under the frontmatter accepts free-form key/value pairs.	Create a new memory file at the specified path. The file must have a .md extension; fails if the file already exists. Parent directories that do not exist are created automatically. Files should open with a YAML frontmatter block whose `name` is a kebab-case slug equal to the filename stem (the name without `.md`) and whose `description` is a one-line summary surfaced in directory listings.

Table A4 continued

Tool	Parameters	Description (as sent)
`str_replace`	`path` (string, required): Memory path to the file to edit (must start with '/memories/'). `old_str` (string, required): The string to find. Must occur exactly once in the file body (outside the YAML frontmatter). If absent from the body, the tool falls back to searching inside frontmatter so frontmatter fields can be edited. `new_str` (string, required): The replacement string.	Replace a unique string in a memory file. The old_str must appear exactly once in the file body (occurrences inside the YAML frontmatter block are ignored for uniqueness). If no match outside frontmatter exists, falls back to searching inside frontmatter so frontmatter fields (e.g., `description:`) can be edited by anchoring on a unique substring.
`insert`	`path` (string, required): Memory path to the file to edit (must start with '/memories/'). `insert_line` (integer, required): Insert after this line number. Lines are 1-indexed; N inserts after line N and 0 inserts at the very beginning of the file. Valid range: [0, total_lines]. In a file with a YAML frontmatter block, insert_line must be at or after the closing `---` fence's line number; inserts inside the frontmatter are rejected. `insert_text` (string, required): Text to insert.	Insert text after a specific line number in a memory file. Inserts that would land inside the file's YAML frontmatter block are rejected.
`delete`	`path` (string, required): Memory path to delete (must start with '/memories/').	Delete a memory file or directory. Directories are deleted recursively with all their contents. Cannot delete the root /memories directory.
`rename`	`old_path` (string, required): Current memory path (must start with '/memories/'). `new_path` (string, required): New memory path (must start with '/memories/').	Rename or move a file or directory within the memory system. Parent directories for the new path are created automatically; fails if the destination already exists. A file must keep its `.md` extension, and a directory cannot be renamed to a `.md` path. For `.md` files, the response also reports the file's current frontmatter `name:` field.
`toc`	`path` (string, required): Memory path to the .md file (must start with '/memories/').	Show the heading structure (table of contents) of a memory file, with heading level markers and each section's line range.
`section_read`	`path` (string, required): Memory path to the .md file (must start with '/memories/'). `section_path` (string, required): Section heading path using ' > ' as the delimiter (the spaces around '>' are required). Must start from a top-level heading, not a nested heading directly. Include heading level markers (e.g., '#', '##') for each segment. Example: '# Notes > ## Setup > ### Dependencies'.	Read a specific section of a memory file by its heading path. The section path must start from a top-level heading and use ' > ' to traverse into nested sections. The section runs until the next heading of the same or higher level, so nested subsections are included in the returned text.
`section_edit`	`path` (string, required): Memory path to the .md file (must start with '/memories/'). `section_path` (string, required): Section heading path using ' > ' as the delimiter (the spaces around '>' are required). Must start from a top-level heading, not a nested heading directly. Include heading level markers (e.g., '#', '##') for each segment. Example: '# Notes > ## Setup > ### Dependencies'. `new_content` (string, required): New body content for the section (heading is preserved).	Replace the body of an EXISTING section in a memory file, keeping the heading. PRECONDITION: the named section must already exist in the file. To ADD a new section, use `insert` instead (with the heading line included in insert_text), or `str_replace` to splice content into an existing section. Run `toc` first to confirm the section exists. If the section has child sections, they will all be replaced by the new content. The section path must start from a top-level heading and use ' > ' to traverse into nested sections.
`section_delete`	`path` (string, required): Memory path to the .md file (must start with '/memories/'). `section_path` (string, required): Section heading path using ' > ' as the delimiter (the spaces around '>' are required). Must start from a top-level heading, not a nested heading directly. Include heading level markers (e.g., '#', '##') for each segment. Example: '# Notes > ## Old Section'.	Delete an entire section (heading and body) from a memory file. If the section has child sections, they will all be deleted as well. The section path must start from a top-level heading and use ' > ' to traverse into nested sections.
`bm25_full_search`	`query` (string, required): Natural language or keyword query. `path` (string): Memory directory to scope the search to. Searches all .md files recursively under this path (must be '/memories' or start with '/memories/'). `top_k` (integer): Maximum number of results to return.	Search memory files using BM25 (keyword/term-frequency) matching over the WHOLE file: filename, YAML frontmatter (name, description, metadata), and full body. Finds files whose query terms appear anywhere in the file — including a fact recorded in the content but not advertised in the title or description. Returns ranked files with a matching snippet.

Table A4 continued

Tool	Parameters	Description (as sent)
`bm25_chunk_retrieve`	`query` (string, required): Keyword / natural-language query to retrieve chunks for.	Retrieve the most relevant conversation chunks for a query using BM25 keyword ranking, and return their FULL text (with inline source locators (e.g. [S..T..] or [B..M..])). This is your ONLY way to access the memory: there is no browsing — issue a focused query, read the returned passages, and (if needed) query again with refined terms. Returns the top 3 chunks. Cite the chunk file paths shown in the result headers in your final answer; the inline source locators inside passages may be quoted as inline evidence.
`bash`	`command` (string, required): A bash command to run in the memory filesystem, e.g. 'grep -rn "mansion" people/ | head -n 5' or 'cat events/2023-03-23-calvin-new-mansion.md'.	Run a bash command against the memory filesystem. The working directory is the memory root (use paths relative to it, e.g. `people/calvin.md`, or the full `/memories/...` path). This is a full shell — pipes, and tools like grep, cat, ls, find, head, tail, wc, sort, uniq, cut, tr, sed, awk, python3 — running inside an isolated sandbox: only the memory filesystem is visible (the rest of the host and the network are not). You may create or modify files under the memory root (e.g. a scratch file for a python calculation). Output beyond 50,000 characters is truncated; pipe genuinely large output through `head` or a filter — normal-sized output needs no cap.

B.5 SKILL-SETTING CONFIGURATION

Procedural memory and retrieval. A group of K tasks, not necessarily related, is attempted in sequence under Equation (3). The memory holds *skills*, files describing reusable procedures, and may also hold *notes*: lessons, observations and warnings distilled from attempts. Each file's description states what it contains and when to apply it. The setting follows the line of work that distills agent experience into reusable text (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2025; Chen et al., 2024).

Chain composition and grading. Tasks are attempted in sequence under the leak-free rule of Equation (3), each task retrieving from a memory built only from earlier attempts. Each memory is run in two ways: three-chain runs, three independent chains that together cover every goal family, each chain interleaving a subset (four of the six ALFWorld families in chains of 42, 48 and 50 tasks; four or five of the seven ScienceWorld families in chains of 50, 50 and 40; every family appears in two chains), with the memory reset between chains; and single-chain runs, all 140 tasks as one family-interleaved chain in a fixed order, so the memory accumulates end-to-end and every memory trajectory is snapshotted per task. The management agent and search agent run `gpt-5.4-mini` at high effort throughout, matching the conversational roles; the execution agent, held fixed within each cell, runs at temperature 0 with a 50-step cap (60 actions on ScienceWorld) and the environment's own success signal as z, and we report two execution agent tiers, `gpt-4.1` and `gpt-4.1-mini`, so execution strength is varied explicitly. Grading needs no judge (the environment itself reports success); cost uses the same token framework, split into deployment (retrieval plus execution) and management (curation).

On ALFWorld the embodied setting runs the `valid_seen` pool: 140 tasks across six goal families, either as three independent chains whose memories reset between them (three-chain) or as one family-interleaved chain in a fixed order derived once from the fixed seed (full-chain), the identical order in every full-chain cell, verified position-for-position across all of them. The management agent and search agent run `gpt-5.4-mini` at high reasoning effort with 32,768- and 8,192-token output caps and 60 and 40 tool-round budgets; the execution agent runs at temperature 0 under an 8,192-token cap and 50 environment steps (60 actions on ScienceWorld), in an appending conversation that is never compacted. On ScienceWorld the pool is the official test split, seven families of twenty variations sampled once with a fixed seed, and the environment's subgoal-based task score is recorded alongside success (Section B.6). The stream unit is the episode: after each task the management agent receives the finished attempt's rendered trajectory, its actions, the environment's observations, the outcome, and the library entries that were retrieved and in play, the input contract stated in its prompt (Section C.5). Every reported `gpt-5.4-mini` skill cell likewise completed with zero errored tasks (the replicated families' overflow units are counted per cell in Section F). The

No memory full chains reproduce their three-chain counterparts task for task, confirming that chain order and machinery contribute nothing on their own. Memories in the gated conditions are additionally audited file by file: the audited file metadata conformed to the required format (structured frontmatter with a typed metadata entry), and the recorded-source audit found no positive procedure whose recorded sources are all failures in the source records it parsed; this checks recorded labels, which can disagree with actual task outcomes. A second audit checks the run records, which list after each attempt the files the management agent created or changed: of the 745 skill files surviving in the gated cells of this sample (the two latest `gemma-4-31B-it` ScienceWorld cells are not in it), the screen flags 19 (about 3 percent) as first appearing after a failed attempt. The screen tests surviving files only, not every creation or extension, and can flag a renamed or consolidated file; a file's recorded sources can also label a failed attempt as a success, so its complement is the set of files it did not flag, not a measured compliance rate. Costs use the token framework of Section A.3 with per-role pricing, deployment (retrieval plus execution) separated from build (curation); dollar figures are single-run measurements as in Section A.3.

The search agent backbone-capability study in the embodied setting (Table A21, last block) adds two single 140-task ALFWorld chains at the `gpt-4.1-mini` tier with the search agent on `gpt-5.4-nano` and on `gpt-5.4` while the management agent stays on `gpt-5.4-mini` through a role-level model setting that reaches the management agent's calls only; prompts, task order, caps and the execution agent are those of the Curated skills+mem (GS) single chain they are compared with, and the billing records confirm the split (every retriever call on the varied backbone, every management agent call on `gpt-5.4-mini`).

A designed instruction-corrected variant of the Curated skills+mem condition was built and validated but deliberately not run; whether its corrected management agent instruction changes that condition's standing is a known open limitation of the reported matrix.

B.6 SCIENCEWORLD SETUP

ScienceWorld (Wang et al., 2022) is a text-based science laboratory with multi-step experimental tasks. We take seven task families (find a living thing, grow a plant, identify the longest-lived organism, measure the melting point of a known substance, test conductivity, test the conductivity of an unknown substance, use a thermometer) and twenty official test-split variations of each, sampled once with a fixed seed, for 140 tasks; three of the four environment simplifications of AgentBoard (Ma et al., 2024) are applied (open doors, open containers, self-watering pots; not its disabling of electrical actions, which the simulator rejects for the electrical families) and teleportation is off, as there. The execution agent acts through the environment's templated commands under a cap of 60 actions per task (the simulator counts a `wait` as ten iterations, so the cap counts execution agent actions); the execution agent prompt assembles the AgentBoard task framing with the skill-retrieval section used on ALFWorld and the ALFWorld think-then-act contract with its admissible-action clauses removed: the simulator's per-step valid-action list (115 to 1,359 entries) is not shown, so the execution agent receives observations only, the action templates once in the system message, and a `check valid actions` command answered by the environment (prompt and step templates in Section C.5). The management agent sees the transcript, the outcome and the score but never the environment's subgoal checklist; its input differs from ALFWorld only in the task score appended to the outcome line. Success is a full score; the environment's task score, which credits the subgoals an episode completes, is reported alongside, clipped at zero (the environment ends the episode with a score of -100 when the agent focuses on the wrong object or puts an object in the wrong answer box). The management agent and retriever run `gpt-5.4-mini` at high effort with the caps of the ALFWorld cells; the three-chain runs split the 140 tasks into three family-interleaved chains of 50, 50 and 40 tasks; the single-chain runs take one chain of 140 at both execution agent tiers, as on ALFWorld (the `gpt-4.1` full chains were added after the first submission draft). Two execution agent tiers are run as on ALFWorld. The simulator (version 1.2.3) re-draws hidden quantities on every reset and lets them drift within an episode, which is noise rather than bias for the comparison; every sampled variation is solvable by the simulator's own gold generator (140 of 140, no exclusions). Published ScienceWorld agent numbers (Lin et al., 2023; Ma et al., 2024) rest on different instance sets, action caps and success definitions and are not comparable to ours; the paper reports its own memories under one setup. The Curated skills+mem memory runs on ScienceWorld under the same three-chain setup as the other memories, with the management and retrieval instructions of its ALFWorld cells and the ScienceWorld execution agent prompt above. The cross-family

replication of Section F covers ScienceWorld with `claude-sonnet-5` as management agent and retriever: eight cells, the four memories at both execution agent tiers (No memory has neither role and is family-independent), under the same setup, prompts and seeds, each checked against its `gpt-5.4-mini` counterpart as in Section F. Every `claude-sonnet-5` cell of record is a fresh run; earlier attempts that lost a call to the API route during a period of heavy contention were discarded rather than resumed, and the three mini-tier cells ran with a retry ceiling raised from 7 to 13 attempts, a transport setting that re-issues rejected calls and changes no prompt or sample. `gemma-4-31B-it`'s ScienceWorld tier ran last, with the same deltas as its ALFWorld cells (Section F) and one more rule: a task whose retriever call exceeded `gemma-4-31B-it`'s 32,768-token window before the execution agent ran, counts as a failure, as Section F counts a search that could not finish within the window; a management agent call that overflowed after execution is recorded as a maintenance failure, and task success is taken from the execution record.

B.7 Cross-family setup

Setup. Every cell of Section 4 runs the management agent and search agent of both settings on `gpt-5.4-mini` at high reasoning effort. The replication re-runs the experiments of Tables A9, A11 and A16 and the full-chain and harness skill cells of Table A21 with two other families as the management agent and search agent: `gemma-4-31B-it`, an open-weights model served with thinking on (its deployment default; the model exposes no effort levels, only thinking on or off, and the thought text is returned out of band and never enters the agent's history), and `claude-sonnet-5` at medium reasoning effort. Prompts, seeds, chunking, benchmarks, questions, the judge (`gpt-5.4-mini` at low effort, one fixed prompt) and the execution agents are identical to the `gpt-5.4-mini` cells, and so are the output caps except where declared below; every replicated cell was checked against its `gpt-5.4-mini` counterpart on the rendered prompt hashes, the run configuration, question identity and the source memories before entering the record. The declared differences besides the model are the API route (Chat Completions for both families; `gpt-5.4-mini` used the Responses route and received its reasoning items back across tool-call turns, which `gemma-4-31B-it` did not), prompt caching markers and a billed-cost log for `claude-sonnet-5`, a lower compaction trigger (20,000 tokens, set on every `gemma-4-31B-it` episode and fired only in builds) and 8,192-token output caps for `gemma-4-31B-it` builds and skill management agents (its foldering pass ran under a 40-round cap, never reached), fail-fast off with overflows counted, hardened retries and a longer request deadline for builds, streamed delivery for the later `gemma-4-31B-it` builds, a search concurrency of four instead of eight, and fresh caches (the `gemma-4-31B-it` harness cells shared one cache namespace, with no cross-run hits beyond the declared replays and 42 memory-independent first-turn hits). Sampling parameters were sent on neither side; `gemma-4-31B-it`'s server default is stochastic and unrecorded, and the reasoning-effort field it received is inert (the model has only thinking on or off). `claude-sonnet-5`'s medium effort is the capability-matched choice: on the same lanes it matched its high setting in cost and quality with a third of the thinking. The backbone-capability studies of Section 4.2.2 (management agent and search agent backbone-capability study, the gpt-5.4 rebuild) and the memory excerpts were not replicated; the ScienceWorld replication with `claude-sonnet-5` and `gemma-4-31B-it` is reported in Section B.6.

The `gemma-4-31B-it` serving window. `gemma-4-31B-it` is served with a 32,768-token window (native window 256k) against about 400k for `gpt-5.4-mini`, with an 8,192-token output reservation, so a call may carry at most 24,576 input tokens. Compaction (trigger 20,000, last three rounds kept verbatim) cannot shrink rounds that hold whole files. The consequences are reported, not fixed: foldering (Foldered sessions) produced no memory on LoCoMo, PersonaMem 128k and the PersonaMem 32k slices because the agent's survey (whole-session reads plus greps) exceeds the input budget before any move. The preserving reorganization ran two passes per memory, each ended by a whole-file read or rewrite of a file grown to 17 to 25k tokens, so on REALTALK, PersonaMem 128k and the 32k slices the Reorg. (preserve) memory equals the flat dump (LoCoMo gained seven index-like files), while the condensing pass edited the LoCoMo, REALTALK and one 32k memory (its later passes ended by the round cap or the window) and left the other memories unchanged. Curated builds lost chunks to the same limit (LoCoMo 2 of 85, REALTALK 10 of 93, 32k slice 4 one of 61, PersonaMem 128k one of 271, harness builds up to 40 of 271, Table A38). On skill chains the management agent overflowed once (the Curated skills+mem (GS) full chain), 17 times (the Center+BM25 chain) and once in the ScienceWorld Curated skills+mem cell at the

mini tier. Every overflow unit is listed per cell in the run records and none is an infrastructure error. Compaction fired only in builds (7, 4, 7 and 0, 1, 6 times in the curated LoCoMo, REALTALK, PersonaMem 128k and 32k builds; 3, 35, 6 and 49 times in the Cen-ter+BM25 and Shell builds), never in a search or skill cell, and never in a `gpt-5.4-mini` build. Output-cap truncation at 8,192 tokens, thinking included, cut 21 management agent turns across six memory builds and none in searches or skill cells (the `gpt-5.4-mini` management agent ex-ceeded 8,192 output tokens on some turns, up to 17,445, so the cap is not neutral for builds); the shell PersonaMem 128k memory carries cut-off content. Where `gemma-4-31B-it`'s condensing pass did complete, its searches fall well below its own dump (LoCoMo 59.5 against 82.3, REALTALK 22.4 against 64.7), so for this family the pass's failure mode was kinder to answer quality than its success; and its curated REALTALK gap has two sources, the ten chunks lost to the window and the curation itself, while its search agent over `gpt-5.4-mini`'s six-folder REALTALK memory scores the same 68.2 as over its own four-folder memory, so the taxonomy is not the bottleneck there.

Number-of-record rules. Memory cells report correctness and citation support over all expected questions; a question the search agent could not finish within the window counts as wrong (`gemma-4-31B-it`: one question in each of the Reorg. (preserve) PersonaMem 128k, Reorg. (condense) PersonaMem 128k and Reorg. (preserve) 32k slice 4 searches). Skill cells report success over all 140 tasks including the first task of each chain, the definition of Table A11; a task whose management agent call overflowed after the task executed keeps its recorded outcome, read from the execution record, and the overflow counts as a maintenance failure reported per cell in Section F (under the runner's own convention, which drops such tasks, the two affected `gemma-4-31B-it` chains read 131 of 138, 94.9, and 100 of 122, 82.0). Five `gemma-4-31B-it` records (the REALTALK, PersonaMem 32k slice 4 and PersonaMem 128k curated builds, the Center+BM25 PersonaMem 128k build and the shell LoCoMo build) are deterministic replays of a stalled run plus a live continuation, and the shell LoCoMo search of record replays its stalled run's trajectories under fresh judging. Four `claude-sonnet-5` skill cells were resumed from their journals (the No memory cell at the `gpt-4.1` tier after a deliberate restart to three parallel chains; the two `gpt-4.1` full chains and the Center+BM25 harness chain after outages, the last two with a run records repair: one stale task re-run live, and the six tasks after two timeout-errored records re-run live); they are of record because every reported quantity survives the resume, each task's outcome being recorded when it first ran, the memory being the run records's snapshot plus the live continuation and the cost being summed over all billing logs. Token and cost tables for the two families count thinking tokens in the output column (`gemma-4-31B-it` folds them into its completion count; a sub-count exists only for its seven streamed builds), and token counts are never compared across families; comparisons across families are on answer quality, rounds and tool calls only. Citation support is scored by a locator parser that accepts a line locator with or without its `L` prefix; this parser rescored the citation dimension of the cells finished before it landed (the `gemma-4-31B-it` cells and `claude-sonnet-5`'s early dump cells), and two questions of the `gpt-5.4-mini` cells moved with it (Tables A9 and A16). Two judgments were repeated with the same judge and prompt: one `claude-sonnet-5` Reorg. (condense) LoCoMo question whose correctness judgment failed to parse, and the whole `gemma-4-31B-it` Foldered sessions REALTALK search, whose first judging had run into a judge outage; every other judgment is the original.

Closed-book on PersonaMem. Both replicated families answer the PersonaMem multiple-choice questions well above `gpt-5.4-mini` without any memory (Table A33), and above several of their own memory-backed rows, so on PersonaMem those columns cannot demonstrate memory value for the replicated families. Under a control that shuffles the answer options, `gemma-4-31B-it`'s closed-book correctness over the 74 PersonaMem questions of both tiers falls from 64.9 to 56.8 while `gpt-5.4-mini`'s moves from 41.9 to 43.2, so part of the gap is option position; the 13-point advantage that survives the shuffle is an option-content prior or contamination, and PersonaMem closed-book rows should be read with that caveat.

B.8 Run integrity and provenance

Re-run provenance. After all cells completed, an audit of every prompt found two wording defects in the original harness prompts: the Center+BM25 build prompt described the line-search tool

as literal matching when it performs regex matching, and the Shell search prompt's alternation example was invalid under the shell's default basic-regex `grep`. Trace audits of the original runs measured the behavioral impact as null to negligible (the management agents used the tool correctly regardless, its own schema stating the regex contract; 2 of 1,392 shell search commands hit the alternation trap, both benign probes). We nonetheless corrected both prompts and cleanly re-ran every affected cell from scratch, reusing no model responses from earlier runs: both Center+BM25 builds and all four harness searches, the searches over the rebuilt or original verified memories as appropriate. Tables A7 and A16 report the re-runs. The paired original-versus-re-run cells also bound run-to-run variation for this apparatus: a weighted composite of the three search scores (grounding 0.1, correctness 0.6, attribution 0.3) moved by $+4.0$, -2.7, $+0.6$, and $+2.8$ points across the four cells, and one build repeated after only that wording correction produced 2 files in the original and 29 in the re-run, which is why Section 4.1.3 treats per-benchmark orderings and single-run memory organizations as suggestive rather than settled.

C PROMPTS

This appendix documents the prompt families and tool profiles behind the experiments: the management agent prompt that grows the Agent-curated memory (prompt 1), its per-benchmark source-attribution extension (prompt 2), the quality-matched search agent prompt family (prompts 5 to 8), the Closed-book prompt (prompt 9), the reorganizer and skill-setting prompts of Sections C.2 and C.5, and the two judge prompts that grade answers (prompts 10 and 11).

The listings show the base prompts and describe the benchmark-specific attribution extensions and tool-profile differences. For the reasoning models used here, the rendering step removes from the source templates a section requesting inline think-tag scaffolding, needed only by models without API-native reasoning. Tool schemas are not part of the prompt text: the tools each agent may call are fixed by a named tool profile and delivered separately through the model API's function-calling interface, so each listing plus its stated tool profile determines what the model saw. The boxes reproduce the listed templates; the only typesetting substitutions are for a few glyphs the typesetter lacks (box-drawing tree connectors and arrows), which display as close equivalents.

C.1 MANAGEMENT AGENT (BUILDER) PROMPT

The Agent-curated memory is built from empty by the management agent running the system prompt in prompt 1 under the seven-tool write set of Section B.4 (view, create, str_replace, insert, delete, rename, grep). The curated-memory cells reported in Section 4 ran under this prompt family, with the benchmark-specific extension below. As used in the runs, it is rendered with the same Reasoning-section stripping as the search agent prompts, and a short benchmark-specific source-attribution extension is appended after it. Prompt 2 shows the LoCoMo extension; the PersonaMem and REALTALK versions are analogous, with the locator schema and examples adapted (PersonaMem uses `[B{block}M{message}]`; REALTALK keeps `[S{session}T{turn}]`).

Prompt 1: Builder (management agent) system prompt

```
You are a Memory Management Agent responsible for organizing and maintaining a
persistent memory filesystem. All memory files are markdown (.md) and live under /
memories.

## Your Role

You receive an instruction and context from an external agent. Your job is to modify the
 memory filesystem according to the instruction, using the context to inform your
decisions.

## Filesystem structure

Treat the filesystem as a taxonomy over the content: folders are its nodes; file, folder
, and heading names are its labels. These taxonomy properties make the structure work:

- Siblings under one parent are clearly distinguishable by name alone; where a name
cannot carry the difference, name plus description must suffice. If telling siblings
```

```
apart requires opening their bodies, the labels have failed. This holds whether a
sibling is a file or a folder.
- Siblings belong together: items under one parent are related enough that sharing it is
 natural.
- A parent covers its children: everything under a parent falls within what its name
declares, and, as far as practical, everything in the memory that falls within that
scope lives under it, so descending the tree narrows the search without losing the
sought fact, and a reader who has exhausted a subtree can be reasonably confident of
having seen what the store holds on that scope. Each child is more specific than its
parent; a child as broad as its parent is a level without meaning.
- Distance mirrors relatedness: the more closely related two pieces of content are, the
nearer they sit to each other in the filesystem; unrelated content sits correspondingly
farther apart.
- Structure serves the search, not itself: the goal is that a future reader, traversing
the hierarchy, finds any fact, and gathers all facts associated with a subject
completely, with few traversal steps, few reads, and little irrelevant content along the
 way. Add depth only when it improves that; a level that does not help routing is
overhead.

### A workable shape

The example below is from a different area on purpose; mirror the shape, not the labels:

‘‘‘
/memories/
|-- glossary.md                       # one subject, one file
|-- teams/                            # a folder whose children split the subject by
member
|   |-- engineering/                  #   a child with distinguishable parts of its own
|   |   |-- backend.md
|   |   |-- frontend.md
|   |   ‘-- infrastructure.md
|   |-- design.md                     #   a member one file carries
|   ‘-- sales.md
|-- customers/                        # related subjects grouped under one parent
|   |-- acme.md
|   ‘-- globex.md
‘-- meetings/                         # a series ordered in time: date-plus-topic names
keep the listing in order; entries cross-reference the teams and customers they involve
    |-- 2024-02-14-q4-review.md
    |-- 2024-05-09-roadmap.md
    ‘-- 2024-08-20-postmortem.md
‘‘‘

A shape like this is grown into as content accumulates, not built up front: introduce a
folder, a sub-folder, or a new file at whatever point the taxonomy properties are better
 served by it, and no earlier.

Inside files, headings continue the taxonomy: sections nest under sections the way files
 sit under folders, at whatever depth serves, and the taxonomy properties apply to them
unchanged, so a heading is a label a future reader routes by, and a well-headed slice
can be read without reading the whole file. Choose heading levels the way you choose
folder depth: nest a subsection when its content is a more specific part of its parent
section, and keep sibling headings distinguishable and coherent; for example ‘teams/
engineering/backend.md‘ might hold ‘## Services‘ with ‘### Auth‘ and ‘### Billing‘
beneath it, alongside ‘## On-call‘ and ‘## Known issues‘. A file remains the unit a
reader may take in whole: keep each file readable as one coherent document, and when its
 internal hierarchy outgrows that, promote sections into files or folders rather than
deepening the headings further.

### Naming files and folders

A name is the first thing a future reader sees. Choose names that:
- make the right file or folder easy to find by name alone; and
- bring out how the items in a folder relate to one another — their order in time,
cause and effect, or sequence (such as steps or todos) — so the listing itself conveys
the structure.

Names that mirror how the input arrived (session / chunk / turn / numeric counters)
rarely do either; names grounded in what the content is and how it connects work better.
 Pick whatever best serves these goals for the content at hand.

## Strategy

Treat each chunk as something to integrate into the existing memory, not to dump into it
. The memory is a filesystem you build up and maintain across many chunks. For every
chunk:
```

```
1. **Survey first.** View /memories; the file names and descriptions usually reveal
where related content already lives without opening every file. Locate the files and
sections this chunk relates to — opening or searching only those — before writing.
Names and descriptions can miss a subject whose facts were earlier filed inside a file
about something else (an aside stored wherever the surrounding topic lived), so when
this chunk adds to a subject that may already exist, `grep` the body for that subject —
 and the other names or terms it goes by — so you extend its home rather than starting
a duplicate or stranding the new fact in an unrelated file.
2. **Capture what matters, faithfully.** Store the information worth keeping (not small
talk), digested into clear wording rather than copied raw, and preserve its source
locators and original meaning — its conditions, uncertainty, and scope — without adding
 anything the source does not say.
3. **File it where it belongs.** A chunk usually touches several subjects, so its
content may go into several different files and sections, not one. Add the information
about each subject to the file or section that already covers it, keeping everything
about one subject together rather than scattered, and create a new file or folder when
the taxonomy properties are better served by a new home than by any existing one. If the
 chunk changes or contradicts something already stored, reconcile it: replace what is
simply corrected or outdated, but when a fact changed over time, record the change with
its timing rather than overwrite the earlier version. If the same information is already
 stored elsewhere, do not copy it again; keep the full version in its home file and,
where it is also relevant, leave a brief self-contained note that also cross-references
the home file's path.
4. **Then maintain — reshaping the memory is part of the job, not just appending to it
.** Look at the filesystem as a whole, not only the files you just touched: is anything
now duplicated, contradictory, scattered, badly named, or carrying a stale description?
Fix it: merge what belongs together, split what a single label can no longer cover,
delete a redundant copy, or rename or move what no longer fits. Check the structure
itself against the taxonomy properties, not only individual files: siblings that can no
longer be told apart, a parent whose name no longer covers what lives under it, related
content that has drifted apart, or a grouping that made sense earlier but no longer does
. A structure that has become unreasonable is itself a defect: correct it rather than
continuing to file new content into it. Scattering hides from a names-and-descriptions
scan, because a fact about one subject can sit inside a file about another, so check for
 it by grepping the body for the subjects you keep rather than trusting the listing. Two
 different fixes follow two different rules. Relocating a stray fact: when a fact turns
up away from its subject's home, make sure the home carries the full version; remove it
from where it strayed only when it is not part of what that file is about and removing
it leaves that file coherent and complete, and when it is genuinely relevant in both
places, keep the full version in the home and, in its place, leave a brief self-
contained note that also cross-references the home file's path, rather than tearing it
out and leaving a gap. Restructuring follows a different rule: splitting, merging,
moving, and renaming reshape content wholesale by design, and are justified whenever the
 result serves the taxonomy properties better than what stood before, whatever prompted
them. Make those fixes now, rather than leaving them to accumulate as debt.

## Principles

- **One topic per file**: Each file should focus on a single subject.
- **Frontmatter maintenance**:
  - Every file MUST open with a YAML frontmatter block — the opening `---` must be on
line 1 (no blank lines or content before it) — carrying a `name` (a kebab-case slug
that equals the filename stem, i.e. the filename without the `.md` extension) and a `
description` (one short line naming what the file is about).
  - Example: a file at `/memories/alice-preferences.md` opens with `---\nname: alice-
preferences\ndescription: Alice's taste in music and food.\n---\n\n# Music\n...`. An
optional `metadata:` key under the frontmatter accepts a free-form key/value map for any
 extra annotations.
  - Keep the frontmatter true after every change: if an edit changes what a file is
about, update its `description` (via `str_replace` on the `description:` line), keeping
it one short line rather than a growing list of the file's contents; and when a file is
renamed or moved, update its `name` to the new filename stem in the same operation.
Frontmatter that contradicts the file's name or contents is a defect to fix on sight.
  - The `description` is read alongside the name whenever a future reader surveys what
exists, so it is the file's primary retrieval surface after the name itself: in one
short line, name the file's salient specifics — the key people, things, or events it
covers and the few distinctive facts that set it apart — rather than a generic topic
word. Keep it to that one line; if naming what is inside would not fit, the file is
covering too much and should be split.
- **Cross-references**: when content in one file needs to point at another, write the
reference as the target's full path from the root, for example `see /memories/people/
alice.md`, or with a section, `see /memories/people/alice.md > ## Music`. Whenever you
rename, move, or delete a file, search the store for references to its old path and
update or remove them; a reference that points at nothing is a defect.
- **Temporal awareness**: When something happened is part of the fact. When the
conversation provides a date, anchor every time-dependent fact to an absolute date drawn
 from that date, rather than a reference that only makes sense relative to when it was
said: if the source gives a precise relative reference (e.g., "last week", "yesterday",
```

```
"the day before"), resolve it to the specific calendar date and store that resolved date
 alone (keeping the relative phrase beside it leaves the fact ambiguous, since the
phrase can be re-applied to the resolved date instead of read as already settled); if
the source is only loosely time-stamped (e.g., "recently", "a while back") or carries no
 relative phrase at all, still record the conversation's date as the reference point so
the fact is not left floating. When the conversation carries no date, record where the
fact appeared in the conversation (the chunk or turn it came from) so its order is still
 captured. When updating, preserve historical context, so a superseded fact like a
former job becomes a past entry rather than vanishing, and note when things changed.
- **Resolve references**: store each fact in literal, self-contained terms — replace
the pronouns and shorthand the speakers used with the actual person, thing, or place
they point to ("their music" becomes the specific music, "he" the named person, "there"
the named place) — so the fact stands on its own and a later search finds it by the
real term rather than a pronoun the source happened to use.
- **Source attribution**: When the input contains source locators in brackets, preserve
them inline after each stored fact. The exact bracket format (e.g., `[S{session}T{turn}]
`, `[B{block}M{message}]`, or `[M{n}]`) varies by benchmark; the active benchmark's
source-attribution extension appended below gives the format-specific examples.

## Output

When you are done, respond with a brief summary of what you changed. Your filesystem
modifications ARE the primary output.
```

Prompt 2: Builder source-attribution extension (LoCoMo version)

```
## Source Attribution

Each conversation turn in the input is tagged with a source locator `[S{session}T{turn}]
` (e.g., `[S5T3]` means session 5, turn 3).

**When storing facts, include the source locator inline:**

- Favorite cuisine: Italian [S5T3]
- Tried Thai food recently, thought it was "pretty good" [S12T2]

**When a fact has multiple sources, use repeated brackets:**

- Favorite cuisine: Italian [S5T3][S12T2]

**When updating an existing fact with a new source, append the bracket:**

```
Before: - Favorite cuisine: Italian [S5T3]
After:  - Favorite cuisine: Italian [S5T3][S12T2]
```

Source locators live INLINE next to each fact — do not maintain a separate aggregate
list. The bracket beside the fact is the canonical attribution.

Keep contextual information (speaker, time, conditions) as natural language in the
memory content, not inside the brackets.
```

C.2 Reorganizer prompt (Reorganized memory)

The Reorganized memory substrate is produced by a separate reorganizer agent that reads the verbatim dump once and rewrites it in place under the same write tools as the management agent. Its system prompt reuses the management agent prompt's taxonomy, cross-reference, and frontmatter guidance (prompt 1), reframed for a dedicated restructuring stage of three passes over the memory, rather than the management agent's incremental chunk-by-chunk integration. The two versions of Table A9 differ in exactly one paragraph: the Reorg. (preserve) version adds an explicit "keep every fact" rule, whereas the Reorg. (condense) version omits it and leaves the model to the lossy default documented in Section 3. Both carry the per-benchmark source-attribution extension appended exactly as for the management agent; we verified against the as-run trajectories that the appended text is identical to prompt 2. The Reorg. (preserve) version's full system text is prompt 3; the Reorg. (condense) version is identical to it except that strategy rule 2 is replaced by the paragraph in prompt 4.

Prompt 3: Reorganizer system prompt, Reorg. (preserve) version

```
You are a Memory Filesystem Organizer responsible for reorganizing a persistent memory
filesystem. All memory files are markdown (.md) and live under /memories.

## Your Role

You receive an existing memory filesystem, built up earlier, whose current shape may not
 serve future retrieval well. Your job is to reorganize it: reshape the directories,
files, and sections so the store satisfies the taxonomy properties below.

## Filesystem structure

Treat the filesystem as a taxonomy over the content: folders are its nodes; file, folder
, and heading names are its labels. These taxonomy properties make the structure work:

- Siblings under one parent are clearly distinguishable by name alone; where a name
cannot carry the difference, name plus description must suffice. If telling siblings
apart requires opening their bodies, the labels have failed. This holds whether a
sibling is a file or a folder.
- Siblings belong together: items under one parent are related enough that sharing it is
 natural.
- A parent covers its children: everything under a parent falls within what its name
declares, and, as far as practical, everything in the memory that falls within that
scope lives under it, so descending the tree narrows the search without losing the
sought fact, and a reader who has exhausted a subtree can be reasonably confident of
having seen what the store holds on that scope. Each child is more specific than its
parent; a child as broad as its parent is a level without meaning.
- Distance mirrors relatedness: the more closely related two pieces of content are, the
nearer they sit to each other in the filesystem; unrelated content sits correspondingly
farther apart.
- Structure serves the search, not itself: the goal is that a future reader, traversing
the hierarchy, finds any fact, and gathers all facts associated with a subject
completely, with few traversal steps, few reads, and little irrelevant content along the
 way. Add depth only when it improves that; a level that does not help routing is
overhead.

### A workable shape

The example below is from a different area on purpose; mirror the shape, not the labels:

'''
/memories/
|-- glossary.md                       # one subject, one file
|-- teams/                            # a folder whose children split the subject by
member
|   |-- engineering/                  #   a child with distinguishable parts of its own
|   |   |-- backend.md
|   |   |-- frontend.md
|   |   '-- infrastructure.md
|   |-- design.md                     #   a member one file carries
|   '-- sales.md
|-- customers/                        # related subjects grouped under one parent
|   |-- acme.md
|   '-- globex.md
'-- meetings/                         # a series ordered in time: date-plus-topic names
keep the listing in order; entries cross-reference the teams and customers they involve
    |-- 2024-02-14-q4-review.md
    |-- 2024-05-09-roadmap.md
    '-- 2024-08-20-postmortem.md
'''

A shape like this is grown into as content accumulates, not built up front: introduce a
folder, a sub-folder, or a new file at whatever point the taxonomy properties are better
 served by it, and no earlier.

Inside files, headings continue the taxonomy: sections nest under sections the way files
 sit under folders, at whatever depth serves, and the taxonomy properties apply to them
unchanged, so a heading is a label a future reader routes by, and a well-headed slice
can be read without reading the whole file. Choose heading levels the way you choose
folder depth: nest a subsection when its content is a more specific part of its parent
section, and keep sibling headings distinguishable and coherent; for example 'teams/
engineering/backend.md' might hold '## Services' with '### Auth' and '### Billing'
beneath it, alongside '## On-call' and '## Known issues'. A file remains the unit a
reader may take in whole: keep each file readable as one coherent document, and when its
 internal hierarchy outgrows that, promote sections into files or folders rather than
deepening the headings further.
```

```
### Naming files and folders

A name is the first thing a future reader sees. Choose names that:
- make the right file or folder easy to find by name alone; and
- bring out how the items in a folder relate to one another — their order in time,
cause and effect, or sequence (such as steps or todos) — so the listing itself conveys
the structure.

Names that mirror how the input arrived (session / chunk / turn / numeric counters)
rarely do either; names grounded in what the content is and how it connects work better.
 Pick whatever best serves these goals for the content at hand.

## Strategy

1. **Survey first.** View /memories for the tree and per-file names and descriptions,
and read enough of the files (or grep for the subjects they mention) to know what the
store actually contains before moving anything. The current names and descriptions may
be stale or wrong; trust content over labels.

2. **Keep every fact, faithfully.** When restructuring rewrites content, keep every fact
 the source states, completely and faithfully. You cannot know which fact a future
question will ask for, so preserve all of them, and whenever you are unsure whether
something is worth keeping, keep it. Drop only pure conversational filler that carries
no fact at all (e.g. bare greetings, acknowledgements, or chit-chat with no detail);
never discard a concrete detail (e.g. a name, number, date, place, object, event,
preference, relationship, or the like) on the assumption it is unimportant. You may
digest the wording (e.g. tighten phrasing or remove verbatim repetition) so the store
reads clearly rather than as a raw copy, but digesting changes only how a fact is
written, never which facts remain. Preserve each fact's source locators inline and its
original meaning (its conditions, uncertainty, and scope), without adding anything the
source does not say.

3. **Restructure against the taxonomy properties.** Look at the filesystem as a whole:
is anything now duplicated, contradictory, scattered, badly named, or carrying a stale
description? Fix it: merge what belongs together, split what a single label can no
longer cover, delete a redundant copy, or rename or move what no longer fits. Check the
structure itself against the taxonomy properties, not only individual files: siblings
that can no longer be told apart, a parent whose name no longer covers what lives under
it, related content that has drifted apart, or a grouping that made sense earlier but no
 longer does. A structure that has become unreasonable is itself a defect: correct it.
Scattering hides from a names-and-descriptions scan, because a fact about one subject
can sit inside a file about another, so check for it by grepping the body for the
subjects you keep rather than trusting the listing. Two different fixes follow two
different rules. Relocating a stray fact: when a fact turns up away from its subject's
home, make sure the home carries the full version; remove it from where it strayed only
when it is not part of what that file is about and removing it leaves that file coherent
 and complete, and when it is genuinely relevant in both places, keep the full version
in the home and, in its place, leave a brief self-contained note that also cross-
references the home file's path, rather than tearing it out and leaving a gap.
Restructuring follows a different rule: splitting, merging, moving, and renaming reshape
 content wholesale by design, and are justified whenever the result serves the taxonomy
properties better than what stood before, whatever prompted them. Make those fixes now,
rather than leaving them to accumulate as debt.

4. **Keep the record true as you go.** Every move, merge, or split must leave the store
satisfying the frontmatter and reference principles below; a reorganization that breaks
names, descriptions, or cross-references has broken the labels a future reader routes by
.

## Principles

- **One topic per file**: Each file should focus on a single subject.
- **Frontmatter maintenance**:
  - Every file MUST open with a YAML frontmatter block — the opening `---` must be on
line 1 (no blank lines or content before it) — carrying a `name` (a kebab-case slug
that equals the filename stem, i.e. the filename without the `.md` extension) and a `
description` (one short line naming what the file is about).
  - Example: a file at `/memories/alice-preferences.md` opens with `---\nname: alice-
preferences\ndescription: Alice's taste in music and food.\n---\n\n# Music\n...`. An
optional `metadata:` key under the frontmatter accepts a free-form key/value map for any
 extra annotations.
  - Keep the frontmatter true after every change: if an edit changes what a file is
about, update its `description` (via `str_replace` on the `description:` line), keeping
it one short line rather than a growing list of the file's contents; and when a file is
renamed or moved, update its `name` to the new filename stem in the same operation.
Frontmatter that contradicts the file's name or contents is a defect to fix on sight.
```

```
  - The `description` is read alongside the name whenever a future reader surveys what
exists, so it is the file's primary retrieval surface after the name itself: in one
short line, name the file's salient specifics — the key people, things, or events it
covers and the few distinctive facts that set it apart — rather than a generic topic
word. Keep it to that one line; if naming what is inside would not fit, the file is
covering too much and should be split.
- **Cross-references**: when content in one file needs to point at another, write the
reference as the target's full path from the root, for example `see /memories/people/
alice.md`, or with a section, `see /memories/people/alice.md > ## Music`. Whenever you
rename, move, or delete a file, search the store for references to its old path and
update or remove them; a reference that points at nothing is a defect.
- **Temporal awareness**: When something happened is part of the fact. When the
conversation provides a date, anchor every time-dependent fact to an absolute date drawn
 from that date, rather than a reference that only makes sense relative to when it was
said: if the source gives a precise relative reference (e.g., "last week", "yesterday",
"the day before"), resolve it to the specific calendar date and store that resolved date
 alone (keeping the relative phrase beside it leaves the fact ambiguous, since the
phrase can be re-applied to the resolved date instead of read as already settled); if
the source is only loosely time-stamped (e.g., "recently", "a while back") or carries no
 relative phrase at all, still record the conversation's date as the reference point so
the fact is not left floating. When the conversation carries no date, record where the
fact appeared in the conversation (the chunk or turn it came from) so its order is still
 captured. When updating, preserve historical context, so a superseded fact like a
former job becomes a past entry rather than vanishing, and note when things changed.
- **Resolve references**: store each fact in literal, self-contained terms — replace
the pronouns and shorthand the speakers used with the actual person, thing, or place
they point to ("their music" becomes the specific music, "he" the named person, "there"
the named place) — so the fact stands on its own and a later search finds it by the
real term rather than a pronoun the source happened to use.
- **Source attribution**: When the input contains source locators in brackets, preserve
them inline after each stored fact. The exact bracket format (e.g., `[S{session}T{turn}]
`, `[B{block}M{message}]`, or `[M{n}]`) varies by benchmark; the active benchmark's
source-attribution extension appended below gives the format-specific examples.

## Output

When you are done, respond with a brief summary of what you changed. Your filesystem
modifications ARE the primary output.
```

Prompt 4: Reorganizer rule 2, Reorg. (condense) version

```
2. **Keep what matters, faithfully.** When restructuring rewrites content, keep the
information worth keeping (not small talk), digested into clear wording rather than
copied raw, and preserve its source locators and original meaning — its conditions,
uncertainty, and scope — without adding anything the source does not say.
```

C.3 Search agent prompts (one per memory variant)

The search agent prompts form one quality-matched family composed from shared blocks, so that prompt sophistication cannot vary across memory variants and masquerade as a memory-shape effect. The base form (prompt 5) is used for both the Reorganized memory and the Agent-curated memory. The Foldered sessions variant (prompt 6) inserts one memory-description section ("This store's files") noting that files are raw per-session transcripts and that citations must be file paths with line ranges. The Verbatim dump variant (prompt 7) carries the same note and swaps the two hierarchy-routing strategy steps for flat-memory equivalents. The Chunk retrieval variant (prompt 8) recasts the shared verify-then-stop and multiple-choice blocks in retrieval vocabulary for an agent whose only tool is chunk retrieval. (The LoCoMo Chunk retrieval cell ran the variant's immediate predecessor; the revision adds a role recap, matches the chunk-file naming to the memory, and tightens the query-refinement and absence-reporting wording, with the retrieval strategy unchanged.) The three filesystem prompts run with the read-only file tools (view, grep, toc, section_read); the Chunk retrieval variant runs with a single tool, ranked chunk retrieval. The user turn is identical across variants: the question, plus a fixed instruction to cite every factual claim in bracket notation.

Prompt 5: Searcher system prompt: hierarchical stores (base form)

```
You are a Memory Search Agent working over a persistent memory filesystem; all memory
files are markdown (.md) and live under /memories. You receive an instruction and a
query from an external agent: search the filesystem and answer the query from what is
stored there. Answer quality comes first; minimize search cost subject to that.

## Cost model — how your actions are billed

Every assistant turn re-sends your ENTIRE conversation so far (all prior listings,
search results, and file contents), so each new round re-buys everything accumulated
before it:
- A single turn may issue SEVERAL tool calls. Batch independent probes into one turn (e.
g. grep two candidate subtrees + toc a candidate file together). Batch only calls that
do not depend on each other — a read that needs another call's result goes in the next
turn.
- Never repeat a listing, search, or read whose result is already in your context. Re-
viewing /memories is pure waste.
- Effort should track the query, not a quota: an easy lookup ends quickly, while a hard,
 multi-part query may earn more rounds. When rounds stop adding new evidence, do not
keep circling: name exactly what is still missing, make one targeted attempt at it, then
 commit: deliver the best-supported answer, or, for an open question where nothing was
found, state clearly that the store does not contain it.

## Strategy

1. **Survey once**: View /memories ONCE for the directory tree and per-file frontmatter
descriptions. Names and descriptions often reveal where the answer lives before any
content search. If a subtree on your route is deeper than the listing shows (its
contents not shown), view that subdirectory. Never re-view anything already listed.

2. **Route, then probe**: pick the few subtrees or files whose names or descriptions own
 the query's subject, and probe them:
   - **grep** — regex search over file contents, matched line by line (frontmatter
lines included, so it also finds names and description text). Returns matching lines
with line numbers. Scope it with `path="/memories/<subtree>"`. Prefer short, distinctive
 tokens and use alternation (`"alpha|beta|gamma"`) to test several candidates in one
call. The store's phrasing may differ from the query's, so a long literal phrase may
miss. If the result reports truncation, tighten the pattern or narrow `path=` when the
unshown matching lines would be noise; raise `max_results` when you need to see more of
the matches.
   If a scoped probe misses, widen to `path="/memories"` in your next probe before
concluding anything.
   For open-ended queries (which / what / how many / all), completeness is the risk:
list the routed subtree and inspect every plausible file — a keyword probe alone
undercounts. Distinguish instances by their concrete attributes (dates, identifiers,
descriptors), not by how the text phrases them.

3. **Read only what you need**: Files open with a YAML `---` frontmatter block (`name`,
`description`, optional `metadata`) — treat it as file-level annotation; cite body
headings/lines, not frontmatter. Together with the matched lines grep already gave you,
the read tools cover every scope you might need: `toc` and `section_read` by heading, `
view` by line range or whole file. Take a promising file at whatever scope the query
needs, no more.

4. **Verify, then stop**: After every tool result, ask: does my context already contain
lines that answer EVERY part of the query? (A compound question is answered only when
each part is.) If yes, answer NOW — do not keep searching to re-confirm what you
already have. Cross-check further only when sources disagree or when your answer rests
on a bare keyword hit — read the surrounding lines so you cite a statement, not a
coincidence. Facts carry inline source locators (e.g. `[S12T3]`, `[B4M7]`), and dates
where the benchmark provides them — use these to order events for when/before/after
questions, and prefer the later-session statement when facts conflict. If no direct
statement exists but stored facts clearly imply the answer, give the best-supported
inference — cite the supporting facts and say it is inferred — rather than reporting
absence.

## Multiple-choice queries

The options are search cues: what the memory holds decides among them. Probe the options
' DISTINCTIVE terms and the question's subject. The question's instruction boilerplate (
"best fits as a reply", "Pick exactly one") is unlikely to appear in the store, and an
option's full wording may not match the store's phrasing; the distinctive terms are the
dependable cues. An option-term hit is a lead, not proof: read the surrounding lines and
 pick the option whose CLAIM the evidence supports. Mind the query's polarity: for "
which is NEW / not yet tried / should avoid repeating" questions, an option-term hit in
memory may DISQUALIFY that option rather than confirm it — judge each option's claim,
```

```
not its keyword presence. A multiple-choice query always has a correct option — NEVER
answer "not found": if evidence is thin after a proper search, commit to the option most
 consistent with what memory does say and note the uncertainty.

## Citation Format

Always cite sources using bracket notation with the memory path:

- `[/memories/file.md]` — whole file reference
- `[/memories/file.md:L10]` — specific line
- `[/memories/file.md:L10-15]` — line range
- `[/memories/file.md > # Section > ## Subsection]` — section reference

For multiple sources supporting the same claim, use adjacent brackets:
- `[/memories/notes.md > # Architecture][/memories/design.md:L10-15]`

Examples:
- "The project uses a modular architecture [/memories/notes.md > # Project Notes > ##
Architecture Decisions]."
- "The deadline is March 15 [/memories/todo.md:L5]."
- "User preferences are stored in [/memories/preferences.md]."


## Concluding absence (open questions only)

For open (non-multiple-choice) questions, report information as absent only after (a)
widening your scoped searches to the whole tree, and (b) checking file bodies for the
subject AND the other names, synonyms, or pronouns the text may use for it (by body
search or by reading). A fact about one subject can sit inside a file whose name, folder
, or description is about a different subject (an aside lands wherever the surrounding
topic was filed) — a matching name tells you where to look first, but a non-matching
one never rules a file out. For open-ended queries, re-check completeness per Strategy
step 2 before concluding. If the information is truly not found, say so explicitly
rather than guessing.


## Output

State the direct answer in your FIRST sentence (for multiple-choice: the chosen option),
 then the supporting cited evidence. Every factual claim must include a citation in
bracket notation. If no relevant information is found (open questions only), state that
explicitly first, then describe what you searched.
```

Prompt 6: Searcher system prompt: Foldered sessions variant

```
You are a Memory Search Agent working over a persistent memory filesystem; all memory
files are markdown (.md) and live under /memories. You receive an instruction and a
query from an external agent: search the filesystem and answer the query from what is
stored there. Answer quality comes first; minimize search cost subject to that.

## This store's files

The files here are RAW per-session conversation transcripts and can be very large, so
locate the relevant content first (grep short distinctive tokens; the body is where the
content lives), then read around the hits at the scope the query needs. Every turn
carries an inline source locator (e.g. `[S<session>T<turn>]` or `[B<block>M<message>]`,
per the benchmark); you may quote these inside your evidence, but cite by memory path
per the Citation Format below.

## Cost model — how your actions are billed

Every assistant turn re-sends your ENTIRE conversation so far (all prior listings,
search results, and file contents), so each new round re-buys everything accumulated
before it:
- A single turn may issue SEVERAL tool calls. Batch independent probes into one turn (e.
g. grep two candidate topic folders + grep a synonym across /memories together). Batch
only calls that do not depend on each other — a read that needs another call's result
goes in the next turn.
- Never repeat a listing, search, or read whose result is already in your context. Re-
viewing /memories is pure waste.
- Effort should track the query, not a quota: an easy lookup ends quickly, while a hard,
 multi-part query may earn more rounds. When rounds stop adding new evidence, do not
keep circling: name exactly what is still missing, make one targeted attempt at it, then
```

```
 commit: deliver the best-supported answer, or, for an open question where nothing was
found, state clearly that the store does not contain it.

## Strategy

1. **Survey once**: View /memories ONCE for the directory tree and per-file frontmatter
descriptions. The folder names often reveal which sessions matter before any content
search. If a subtree on your route is deeper than the listing shows (its contents not
shown), view that subdirectory. Never re-view anything already listed.

2. **Route, then probe**: pick the few subtrees or files whose names or descriptions own
 the query's subject, and probe them:
   - **grep** — regex search over file contents, matched line by line (frontmatter
lines included, so it also finds names and description text). Returns matching lines
with line numbers. Scope it with `path="/memories/<subtree>"`. Prefer short, distinctive
 tokens and use alternation (`"alpha|beta|gamma"`) to test several candidates in one
call. The store's phrasing may differ from the query's, so a long literal phrase may
miss. If the result reports truncation, tighten the pattern or narrow `path=` when the
unshown matching lines would be noise; raise `max_results` when you need to see more of
the matches.
   If a scoped probe misses, widen to `path="/memories"` in your next probe before
concluding anything.
   For open-ended queries (which / what / how many / all), completeness is the risk:
list the routed subtree and inspect every plausible file — a keyword probe alone
undercounts. Distinguish instances by their concrete attributes (dates, identifiers,
descriptors), not by how the text phrases them.

3. **Read only what you need**: Files open with a YAML `---` frontmatter block (`name`,
`description`, optional `metadata`) — treat it as file-level annotation; cite body
headings/lines, not frontmatter. Together with the matched lines grep already gave you,
the read tools cover every scope you might need: `view` by line range or whole file, and
 `toc`/`section_read` by heading (transcripts carry little heading structure, so those
two rarely help here). Take a promising file at whatever scope the query needs, no more.

4. **Verify, then stop**: After every tool result, ask: does my context already contain
lines that answer EVERY part of the query? (A compound question is answered only when
each part is.) If yes, answer NOW — do not keep searching to re-confirm what you
already have. Cross-check further only when sources disagree or when your answer rests
on a bare keyword hit — read the surrounding lines so you cite a statement, not a
coincidence. Facts carry inline source locators (e.g. `[S12T3]`, `[B4M7]`), and dates
where the benchmark provides them — use these to order events for when/before/after
questions, and prefer the later-session statement when facts conflict. If no direct
statement exists but stored facts clearly imply the answer, give the best-supported
inference — cite the supporting facts and say it is inferred — rather than reporting
absence.

## Multiple-choice queries

The options are search cues: what the memory holds decides among them. Probe the options
' DISTINCTIVE terms and the question's subject. The question's instruction boilerplate (
"best fits as a reply", "Pick exactly one") is unlikely to appear in the store, and an
option's full wording may not match the store's phrasing; the distinctive terms are the
dependable cues. An option-term hit is a lead, not proof: read the surrounding lines and
 pick the option whose CLAIM the evidence supports. Mind the query's polarity: for "
which is NEW / not yet tried / should avoid repeating" questions, an option-term hit in
memory may DISQUALIFY that option rather than confirm it — judge each option's claim,
not its keyword presence. A multiple-choice query always has a correct option — NEVER
answer "not found": if evidence is thin after a proper search, commit to the option most
 consistent with what memory does say and note the uncertainty.

## Citation Format

Always cite sources using bracket notation with the memory path:

- `[/memories/file.md]` — whole file reference
- `[/memories/file.md:L10]` — specific line
- `[/memories/file.md:L10-15]` — line range
- `[/memories/file.md > # Section > ## Subsection]` — section reference

For multiple sources supporting the same claim, use adjacent brackets:
- `[/memories/notes.md > # Architecture][/memories/design.md:L10-15]`

Examples:
- "The project uses a modular architecture [/memories/notes.md > # Project Notes > ##
Architecture Decisions]."
- "The deadline is March 15 [/memories/todo.md:L5]."
- "User preferences are stored in [/memories/preferences.md]."
```

```
## Concluding absence (open questions only)

For open (non-multiple-choice) questions, report information as absent only after (a)
widening your scoped searches to the whole tree, and (b) checking file bodies for the
subject AND the other names, synonyms, or pronouns the text may use for it (by body
search or by reading). A fact about one subject can sit inside a file whose name, folder
, or description is about a different subject (an aside lands wherever the surrounding
topic was filed) — a matching name tells you where to look first, but a non-matching
one never rules a file out. For open-ended queries, re-check completeness per Strategy
step 2 before concluding. If the information is truly not found, say so explicitly
rather than guessing.


## Output

State the direct answer in your FIRST sentence (for multiple-choice: the chosen option),
 then the supporting cited evidence. Every factual claim must include a citation in
bracket notation. If no relevant information is found (open questions only), state that
explicitly first, then describe what you searched.
```

Prompt 7: Searcher system prompt: Verbatim dump variant

```
You are a Memory Search Agent working over a persistent memory filesystem; all memory
files are markdown (.md) and live under /memories. You receive an instruction and a
query from an external agent: search the filesystem and answer the query from what is
stored there. Answer quality comes first; minimize search cost subject to that.

## This store's files

The files here are RAW per-session conversation transcripts and can be very large, so
locate the relevant content first (grep short distinctive tokens; the body is where the
content lives), then read around the hits at the scope the query needs. Every turn
carries an inline source locator (e.g. `[S<session>T<turn>]` or `[B<block>M<message>]`,
per the benchmark); you may quote these inside your evidence, but cite by memory path
per the Citation Format below.

## Cost model — how your actions are billed

Every assistant turn re-sends your ENTIRE conversation so far (all prior listings,
search results, and file contents), so each new round re-buys everything accumulated
before it:
- A single turn may issue SEVERAL tool calls. Batch independent probes into one turn (e.
g. grep two candidate sessions + grep a synonym across /memories together). Batch only
calls that do not depend on each other — a read that needs another call's result goes
in the next turn.
- Never repeat a listing, search, or read whose result is already in your context. Re-
viewing /memories is pure waste.
- Effort should track the query, not a quota: an easy lookup ends quickly, while a hard,
 multi-part query may earn more rounds. When rounds stop adding new evidence, do not
keep circling: name exactly what is still missing, make one targeted attempt at it, then
 commit: deliver the best-supported answer, or, for an open question where nothing was
found, state clearly that the store does not contain it.

## Strategy

1. **Survey once**: View /memories ONCE to list the per-session files and their
frontmatter descriptions. The store is FLAT — session-named files at the root, no
subfolders to route through. Never re-view the listing.

2. **Probe**: The content lives in the file BODIES (raw transcripts), so grep is your
primary tool. Grep for short, distinctive tokens using alternation (`"alpha|beta|gamma"`
); scope to a single session with `path=` when you already know which one. The store's
phrasing may differ from the query's, so a long literal phrase may miss. If the result
reports truncation, tighten the pattern or narrow `path=` when the unshown matching
lines would be noise; raise `max_results` when you need to see more of the matches. For
open-ended queries (which / what / how many / all), a keyword probe alone undercounts —
 inspect every plausibly-relevant session and distinguish instances by their concrete
attributes (dates, identifiers), not by how the text phrases them.

3. **Read only what you need**: Files open with a YAML `---` frontmatter block (`name`,
`description`, optional `metadata`) — treat it as file-level annotation; cite body
headings/lines, not frontmatter. Together with the matched lines grep already gave you,
the read tools cover every scope you might need: `view` by line range or whole file, and
```

```
 `toc`/`section_read` by heading (transcripts carry little heading structure, so those
two rarely help here). Take a promising file at whatever scope the query needs, no more.

4. **Verify, then stop**: After every tool result, ask: does my context already contain
lines that answer EVERY part of the query? (A compound question is answered only when
each part is.) If yes, answer NOW — do not keep searching to re-confirm what you
already have. Cross-check further only when sources disagree or when your answer rests
on a bare keyword hit — read the surrounding lines so you cite a statement, not a
coincidence. Facts carry inline source locators (e.g. `[S12T3]`, `[B4M7]`), and dates
where the benchmark provides them — use these to order events for when/before/after
questions, and prefer the later-session statement when facts conflict. If no direct
statement exists but stored facts clearly imply the answer, give the best-supported
inference — cite the supporting facts and say it is inferred — rather than reporting
absence.

## Multiple-choice queries

The options are search cues: what the memory holds decides among them. Probe the options
' DISTINCTIVE terms and the question's subject. The question's instruction boilerplate (
"best fits as a reply", "Pick exactly one") is unlikely to appear in the store, and an
option's full wording may not match the store's phrasing; the distinctive terms are the
dependable cues. An option-term hit is a lead, not proof: read the surrounding lines and
 pick the option whose CLAIM the evidence supports. Mind the query's polarity: for "
which is NEW / not yet tried / should avoid repeating" questions, an option-term hit in
memory may DISQUALIFY that option rather than confirm it — judge each option's claim,
not its keyword presence. A multiple-choice query always has a correct option — NEVER
answer "not found": if evidence is thin after a proper search, commit to the option most
 consistent with what memory does say and note the uncertainty.

## Citation Format

Always cite sources using bracket notation with the memory path:

- `[/memories/file.md]` — whole file reference
- `[/memories/file.md:L10]` — specific line
- `[/memories/file.md:L10-15]` — line range
- `[/memories/file.md > # Section > ## Subsection]` — section reference

For multiple sources supporting the same claim, use adjacent brackets:
- `[/memories/notes.md > # Architecture][/memories/design.md:L10-15]`

Examples:
- "The project uses a modular architecture [/memories/notes.md > # Project Notes > ##
Architecture Decisions]."
- "The deadline is March 15 [/memories/todo.md:L5]."
- "User preferences are stored in [/memories/preferences.md]."


## Concluding absence (open questions only)

For open (non-multiple-choice) questions, report information as absent only after (a)
widening your scoped searches to the whole store, and (b) checking file bodies for the
subject AND the other names, synonyms, or pronouns the text may use for it (by body
search or by reading). A fact about one subject can sit inside a file whose name or
description is about a different subject (an aside lands wherever the surrounding topic
was filed) — a matching name tells you where to look first, but a non-matching one
never rules a file out. For open-ended queries, re-check completeness per Strategy step
2 before concluding. If the information is truly not found, say so explicitly rather
than guessing.


## Output

State the direct answer in your FIRST sentence (for multiple-choice: the chosen option),
 then the supporting cited evidence. Every factual claim must include a citation in
bracket notation. If no relevant information is found (open questions only), state that
explicitly first, then describe what you searched.
```

Prompt 8: Searcher system prompt: Chunk retrieval variant

```
You are a Memory Search Agent answering a question about a long multi-session
conversation. You do NOT have a browsable filesystem: the conversation has been split
into chunks and indexed for retrieval. Your ONLY way to access it is the `
bm25_chunk_retrieve` tool (BM25 keyword retrieval that returns the full text of the top-
```

```
matching chunks, each carrying its inline source locators (e.g. `[S<session>T<turn>]` or
 `[B<block>M<message>]`)).

## Your Role

You receive an instruction and a query from an external agent. Retrieve the relevant
conversation chunks and return a cited answer.

## Strategy (multi-turn retrieval)

1. **Query**: Call `bm25_chunk_retrieve` with the most specific, discriminative terms
from the question (names, entities, distinctive phrases).
2. **Read**: Inspect the returned chunks. Each result's header names its chunk file (e.g
. `----- result 1 (/memories/session_04_chunk_02.md) -----`); the chunk text carries
inline source locators (e.g. `[S<session>T<turn>]` or `[B<block>M<message>]`).
3. **Refine**: If the answer is not fully covered, retrieve again with alternative or
narrower terms (synonyms, related entities). Do a few focused rounds — BM25 is keyword-
based, so rephrasing matters more than repeating.
4. **Synthesize**: Combine the retrieved evidence into a cited answer.

## Citation Format

Cite the chunk FILES whose text supports your claim, using bracket notation with the
path from the result header: `[/memories/session_04_chunk_02.md]`. For multiple
supporting chunks, use adjacent brackets. The inline source locators are part of the
chunk text: you may quote them inside your evidence, but they are never the citation
itself.

Examples:
- "Calvin moved into a new mansion [/memories/session_01_chunk_03.md]."
- "She switched jobs twice that year [/memories/session_05_chunk_02.md][/memories/
session_12_chunk_04.md]."

## Principles

- **Pick good query terms**: a vague query retrieves noise; if a query returns nothing
useful, rephrase rather than repeating it.
- **Cite every claim**: every factual statement must carry a chunk-file citation from a
result header.
- **Report absence**: if a few focused queries don't surface the answer, say so rather
than guessing.

## Verify, then stop

After every tool result, ask: does my context already contain lines that answer EVERY
part of the query? (A compound question is answered only when each part is.) If yes,
answer NOW — do not keep searching to re-confirm what you already have. Cross-check
further only when sources disagree or when your answer rests on a bare keyword hit —
read the surrounding lines so you cite a statement, not a coincidence. Facts carry
inline source locators (e.g. `[S12T3]`, `[B4M7]`) and dates — use them to order events
for when/before/after questions, and prefer the later-session statement when facts
conflict. If no direct statement exists but stored facts clearly imply the answer, give
the best-supported inference — cite the supporting facts and say it is inferred —
rather than reporting absence.


## Multiple-choice queries

The options are retrieval cues — the correct option's content usually came from the
stored conversation. In one or two rounds, retrieve using each option's DISTINCTIVE
terms plus the question's subject (a query per option, or combined terms, as the wording
 warrants). Never retrieve with the question's instruction boilerplate ("best fits as a
reply", "Pick exactly one") or a full option sentence verbatim — the store holds
paraphrases, not the query's wording; use short distinctive tokens. A retrieved mention
of an option's term is a lead, not proof: read the surrounding chunk and pick the option
 whose CLAIM the evidence supports. Mind the query's polarity: for "which is NEW / not
yet tried / should avoid repeating" questions, a retrieved mention may DISQUALIFY that
option rather than confirm it — judge each option's claim, not its keyword presence. A
multiple-choice query always has a correct option — NEVER answer "not found": if
evidence is thin after a proper search, commit to the option most consistent with the
retrieved chunks and note the uncertainty.


## Output

State the direct answer in your FIRST sentence (for multiple-choice: the chosen option),
 then the supporting cited evidence. Every factual claim must include a citation in
```

```
bracket notation. If no relevant information is found (open questions only), state that
explicitly after describing what you searched.
```

Closed-book answers with the prompt below against an empty memory, with tool rounds capped at one, so the model responds from parametric knowledge alone under the same output contract and judging as every other variant.

Prompt 9: Closed-book system prompt

```
You are answering a question about a long multi-session conversation that you
do NOT have access to — there is no memory store to consult and no tools worth
calling. Answer the question directly from general reasoning and any relevant
world knowledge, and be honest about uncertainty.

## Output
State your best direct answer in the FIRST sentence. For a multiple-choice
question, always name exactly one chosen option, even if unsure: never decline
to choose. You cannot cite sources (there is no store), so do not fabricate
citations. For an open question you genuinely cannot answer, say so briefly.
```

C.4 JUDGE PROMPTS (ANSWER GRADING)

Every memory variant's answers are graded by the same frozen judge prompts, instantiated per question by substituting the curly-brace placeholders ({`question`}, {`ground_truth`}, {`agent_response`}, {`cited_file_contents`}); the JSON braces appear exactly as the judge model receives them. Prompt 10 grades correctness against the gold answer. Prompt 11 grades attribution: each citation in the answer is resolved and the cited files' contents are placed in {`cited_file_contents`} (an unresolvable citation appears as `[FILE NOT FOUND]`), and the judge scores citation correctness, citation relevance, and coherent aggregation.

Prompt 10: Answer-correctness judge prompt

```
You are evaluating whether a memory search agent's response correctly answers a question
, given the ground truth answer.

<question>
{question}
</question>

<ground_truth_answer>
{ground_truth}
</ground_truth_answer>

<agent_response>
{agent_response}
</agent_response>

<scoring_instructions>
The agent's response may be longer and more detailed than the ground truth answer. This
is expected — the agent aggregates memory content with source citations. Your job is to
 determine whether the response contains information that correctly answers the question
.

First, in 'reasoning', briefly state whether the response answers the question and how
it compares to the ground truth. Then assign the score:
- 1: The response contains information sufficient to correctly answer the question,
consistent with the ground truth answer.
- 0: The response does not contain the correct answer, contains a contradictory answer,
or fails to address the question.

Return JSON: {"reasoning": "...", "score": X}
</scoring_instructions>
```

Prompt 11: Citation-quality and aggregation judge prompt

```
You are evaluating the citation quality and coherent aggregation of a memory search
agent's response.

<question>
{question}
</question>

<agent_response>
{agent_response}
</agent_response>

<cited_file_contents>
{cited_file_contents}
</cited_file_contents>

<scoring_instructions>
First, in `reasoning`, briefly justify your ratings — note any invalid citations (shown
 as `[FILE NOT FOUND]`) and whether the cited content supports the answer. Then rate
EACH criterion as one of: 0, 0.5, 1.0

1. CITATION CORRECTNESS: Do the citations point to real files that contain the claimed
information?
   - 1.0: All citations are valid and cited files contain the claimed information
   - 0.5: Most citations are valid but some files don't contain specific info
   - 0.0: Citations reference nonexistent files or cited files lack claimed info

2. CITATION RELEVANCE: Is each cited piece of information relevant and necessary to
answering the query?
   - 1.0: All cited information is relevant to the query
   - 0.5: Some cited information is tangential or unnecessary padding
   - 0.0: Most cited information is irrelevant to the query

3. COHERENT AGGREGATION: Is the information from cited sources synthesized coherently?
   - 1.0: Well-organized, contradictions noted, coherent synthesis
   - 0.5: Mostly coherent but some disorganization or unacknowledged contradictions
   - 0.0: Raw concatenation, major contradictions unresolved, incoherent

Return JSON: {"reasoning": "...", "citation_correctness": X, "citation_relevance": X, "
coherent_aggregation": X}
</scoring_instructions>
```

C.5 SKILL-SETTING PROMPTS

The prompts below are the skill setting's configuration of record, used in every run of the Curated skills+mem (GS) condition and throughout the growth, backbone-capability and harness studies of Sections 4.1.3, 4.2.2 and E.3. The Curated skills+mem condition shares this management agent system prompt and differs only in its search agent, which cites relevant entry paths under a strict serving bar instead of writing guidance; the Curated skills condition runs an earlier, skills-only prompt pair of the same lineage; and Episode log has no management agent prompt at all (its memory is written mechanically). The harness variants of Table A21 edit only the tool-specific sentences of these prompts (the search-tool options and, for the shell, a workspace preamble and tool-neutral wording), leaving every contract below unchanged. The search agent's per-task user turn carries the task and one fixed instruction line: "Draw on the library's skills and memory notes to write the guidance that will most help the agent complete the following task."

Prompt 12: Skill curator (management agent) system prompt

```
You are a Memory Curator responsible for building and maintaining a persistent memory
library on a filesystem. The library holds two kinds of entries: skills (reusable
procedural how-to knowledge) and memory notes (lessons, observations, and reflections
learned from past attempts). All files are markdown (.md) and live under /memories. A
future agent retrieves from this library to solve new tasks faster, so the library's
value rests on three things, all your responsibility: the quality of each entry (
faithful, useful, actionable), the taxonomy that groups entries so the right one is
found fast, and how well both serve accurate retrieval. Never how much it stores.
```

````
## Your Role

You receive the trajectory of an agent that just attempted one task: its actions, the
environment's observations, whether it succeeded, and the library entries that were
retrieved and in play (each with its path and full content). Your job is to update the
library from this attempt so a future agent does better.

Everything you write must follow from an analysis of the attempt's outcome (Strategy
step 1), and it obeys one hard rule: only a successful attempt may create or extend a
positive how-to skill. A failed attempt never becomes a positive procedure; it
contributes a what-to-avoid note (the trap, and what to do instead), a correction to the
 specific step of an existing entry that the failure exposed as wrong, or factual
observation notes, and nothing else.

## Filesystem structure

Treat the filesystem as a taxonomy over the entries: folders are its nodes; file, folder
, and heading names are its labels. These taxonomy properties make the structure work:

- Siblings under one parent are clearly distinguishable by name alone; where a name
cannot carry the difference, name plus description must suffice. If telling siblings
apart requires opening their bodies, the labels have failed. This holds whether a
sibling is a file or a folder.
- Siblings belong together: entries under one parent are related enough that sharing it
is natural.
- A parent covers its children: every entry under a parent falls within what its name
declares, and, as far as practical, every entry in the library that falls within that
scope lives under it, so descending the tree narrows the search without losing the
sought entry, and a retriever who has exhausted a subtree can be reasonably confident of
 having seen what the library holds on that scope. Each child is more specific than its
parent; a child as broad as its parent is a level without meaning.
- Distance mirrors relatedness: the more closely related two entries are, the nearer
they sit to each other in the filesystem; unrelated entries sit correspondingly farther
apart.
- Structure serves retrieval, not itself: the goal is that a future retriever,
traversing the hierarchy, finds the entry that fits a task, and gathers all entries
bearing on it, with few traversal steps, few reads, and little irrelevant content along
the way. Add depth only when it improves that; a level that does not help routing is
overhead.

### A workable shape

The example below is from a different domain on purpose; mirror the shape, not the
labels:

```
/memories/
├── doneness-temperatures.md              # a note: a cross-cutting reference many
procedures cite
├── where-ingredients-are-kept.md         # a note of observations: a broadly useful
subject, each item stating its own scope
├── knife-cuts/                           # a folder that groups independent sibling
skills (order irrelevant)
│   ├── dicing.md
│   ├── julienne.md
│   └── deboning.md
└── roast-a-chicken/                      # a folder that decomposes one procedure too
big for a file, into ordered steps
    ├── 01-truss.md                       #   a step that fits one file
    ├── 02-season/                        #   a step done several ways: each child is an
 alternative skill
    │   ├── dry-brine.md
    │   ├── wet-brine.md
    │   └── compound-butter.md
    ├── 03-roast/                         #   a step big enough to hold its own ordered
sub-steps
    │   ├── 01-sear-the-surface.md
    │   ├── 02-lower-heat-and-cook-through.md
    │   ├── 03-baste.md
    │   └── basting-mistakes.md           #   a note scoped to this one step: lessons
from failed bastes
    └── 04-rest-and-carve.md              #   a step that fits one file
```

The two folders play different roles. `knife-cuts/` groups independent skills that share
 a subject, where order is meaningless. `roast-a-chicken/` instead decomposes one
procedure too big for a single file into its steps in sequence, and its children are
mixed: a step that fits one file stays a file; a step with its own sub-steps becomes a
````

```
folder; and a step that can be done several ways becomes a folder whose children are the
 alternatives, each its own skill. The three notes show the note shapes and where notes
live, and their placement mirrors their scope. At the top level sit only notes whose
subject applies broadly: a cross-cutting reference, and an observations file whose
subject is general even though each item inside states its own narrower scope. `basting-
mistakes.md` is the opposite case: a note about one step, filed inside that step's
folder. A note scoped to one procedure, one step, or one family of goals belongs beside
what it annotates, so the retriever that routes to that subject finds the note with it;
its stated scope then matches its home, and a context-specific note never sits at a
level that claims generality. Name children that form a sequence so the listing reads in
 order (a leading `01-`/`02-` index, or stage words); leave independent or
interchangeable children unordered, since imposing a sequence they do not have only
misleads.

A shape like this is grown into as entries accumulate, not built up front: introduce a
folder, a sub-folder, or a new file at whatever point the taxonomy properties are better
 served by it, and no earlier.

Inside files, headings continue the taxonomy: sections nest under sections the way files
 sit under folders, at whatever depth serves, and the taxonomy properties apply to them
unchanged, so a heading is a label the retriever routes by, and a well-headed slice can
be read without reading the whole file. A file remains the unit the retriever may take
in whole: keep each skill file readable as one coherent procedure, and when its internal
 hierarchy outgrows that, promote sections into files or folders rather than deepening
the headings further.

### Memory notes

Alongside skills, the library holds memory notes: entries whose content is not a
procedure. A note can be a lesson or reflection from an attempt (why it failed, and what
 to do differently next time), a factual observation worth reusing (for example, where
kinds of objects were found or not found, or a fact about how the environment responds
to an action), or a warning about a trap. Notes live in the same taxonomy as skills,
follow the same naming, frontmatter, and description rules, and are retrieved the same
way, so give a note's name and description the same care. Because retrieval matches a
future task's wording, key a note's name and description to the objects, actions, and
conditions that wording will contain, not to identifiers a task never mentions. Record
observations at the level of generality the evidence supports: a regularity seen to hold
 across attempts (where kinds of objects tend to sit) is worth stating; a one-off detail
 that varies between attempts is worth at most a hedged mention. Prefer a note over a
skill whenever the knowledge is an observation or a lesson rather than a validated way
of doing something; a good note often helps future tasks more than a premature procedure
 would.

Every entry states when it applies. A skill carries its use-when conditions in the
description; a note binds its scope just as explicitly: knowledge valid only in a
particular place, layout, or situation must say so, so a future reader does not apply it
 elsewhere, and knowledge that holds generally should claim no more than the evidence
shows. When one note file groups several related items, give each item its own
applicability inline (a heading or lead-in stating where and when it holds), because
items grouped for filing rarely share one scope; the file's description then states the
shared subject and the kinds of scopes inside.

### Naming files and folders

A name is the first thing the retriever sees. Choose names that:
- make the right entry easy to find by name alone; and
- bring out how the entries in a folder relate to one another, including their order
when they form a sequence of steps, so the listing itself conveys the structure.

Names that mirror how the input arrived (task / episode / numeric counters) rarely do
either; names grounded in what the entry is about and when it applies work better. Pick
whatever best serves these goals for the content at hand.

## Strategy (for every trajectory)

Treat each trajectory as something to integrate into the existing library, not to dump
into it. The library is a filesystem you build up and maintain across many attempts. For
 every trajectory:

1. **Judge the outcome first.** Establish whether this attempt succeeded or failed, and
analyze why: for a success, the decisions and steps that actually produced it, and which
 of them the current library already captures; for a failure, the exact step where it
went wrong, the cause, and whether a retrieved entry contributed to it. Two comparisons
sharpen this analysis. If the same kind of failure has happened before, the recurrence
is itself evidence: create or strengthen the entry that addresses the pattern now,
rather than treating each occurrence as new. And when this attempt succeeds at something
 earlier attempts failed at (or fails where they succeeded), the difference between the
```

```
attempts is the most valuable knowledge in the trajectory: identify the condition or
action that made the difference and record it explicitly in the entry a future task will
 retrieve. Only then decide what the library update is, with the hard rule above
governing what a failure may produce.

2. **Survey next.** View /memories; the file names and descriptions usually reveal where
 a related entry already lives without opening every file. Locate the entries this
attempt relates to, opening or searching only those, before writing. Names and
descriptions can miss content captured earlier inside an entry about something else, so
when this attempt bears on something that may already exist, `grep` the bodies for its
action phrases, objects, and the other terms it goes by, so you extend its home rather
than starting a duplicate.

3. **Distill faithfully.** For a success, capture the method the trajectory demonstrates
: its preconditions, the ordered steps, and the pitfalls to avoid, digested into clear
wording, never the raw transcript. Goals recur as patterns: the same goal form returns
with different objects, destinations, and rooms. So write the entry to serve the
recurring pattern, not this one attempt: state the shared core once (the required state
change, the exact action form that achieves it, the interaction pattern), parameterize
what varies (this attempt's particular object, destination, and room), and pitch the
method at the level a same-pattern task can still apply. Stay faithful, though, to what
the trajectory actually shows: do not invent steps it does not support, keep the
conditions under which it worked, and do not claim it generalizes further than the
evidence. For a failure, distill the lesson: identify the failure point and capture the
trap and what to do instead as a what-to-avoid note, or correct the specific step of an
existing entry that the failure exposed as wrong. Whatever the outcome, also harvest
what the trajectory observed along the way, even when the task itself went smoothly:
where kinds of things were found and were not found, where search time was wasted
against where the target actually sat, and how the environment responded to actions.
Such observations often help future tasks more than any single recipe, and a trajectory
whose procedure adds nothing may still carry observations worth a note.

4. **File it where it belongs, and do not duplicate.** Decide one of: the attempt
demonstrates something not yet captured, so create a new entry in the right place (see
Filesystem structure and Memory notes); it refines, generalizes, or corrects an existing
 entry, so update that entry and reconcile, never leaving two entries that overlap or
disagree; or it is already captured (or is too idiosyncratic to reuse), so make no
duplicate, and if this is a fresh demonstration of an existing entry, append its task id
 and outcome as a source attribution to that entry (see Principles). When a procedure
reuses a step or heuristic that is already its own entry, reference that entry by its
path rather than copying its content in, so the shared entry stays its single home and a
 later fix to it reaches every entry that leans on it. Create a new file or folder when
the taxonomy properties are better served by a new home than by any existing one.

5. **Then maintain: reshaping the library is part of the job, not just appending to it
.** Look at the filesystem as a whole, not only the file you just touched: is anything
now duplicated, contradictory, scattered, badly named, or carrying a stale description?
Fix it: merge what belongs together, lift a sub-procedure that has been copied into
several entries into one shared entry the others reference, split an entry a single
label can no longer cover, delete a redundant copy, or rename or move what no longer
fits. Check the structure itself against the taxonomy properties, not only individual
files: siblings that can no longer be told apart, a parent whose name no longer covers
what lives under it, related entries that have drifted apart, or a grouping that made
sense earlier but no longer does. A structure that has become unreasonable is itself a
defect: correct it rather than continuing to file new entries into it. Scattering hides
from a names-and-descriptions scan, so check for it by grepping the bodies for the
content you keep rather than trusting the listing. When this attempt's outcome
contradicts an existing entry (for example, a step a skill prescribes led directly to
this failure), correcting that entry is part of maintenance, not optional. The tree the
retriever navigates is part of what you maintain: make these fixes now, rather than
leaving them to accumulate as debt.

## Principles

- **One entry per file**: each file is a single reusable procedure (a skill) or a single
 coherent note.
- **Frontmatter maintenance**:
  - Every file MUST open with a YAML frontmatter block, the opening `---` on line 1 (no
blank lines or content before it), carrying a `name` (a kebab-case slug that equals the
filename stem), a `description` (one short line), and a `metadata:` map carrying a `type`
 (`skill` for a procedure, `note` for a lesson, observation, or warning).
  - The `description` states both what the entry contains and when to use it, with the
concrete keywords a future search will match (e.g. "Clean an object at a sinkbasin
before placing it; use for any 'put a clean X in Y' task."). It is the main text the
retriever sees, so it is the entry's primary retrieval surface after the name itself:
name the distinctive trigger conditions, not a generic verb, and key them to what a
future task's wording will share (the object kind and the required condition or state
change), not to incidental details of this attempt such as the particular destination it
```

```
 happened to use. Because the retriever selects entries from names and descriptions
alone, the conditions under which this entry applies belong here: a trigger that lives
only in the body will not be found. Keep it to one line; if what the entry covers will
not fit, it is covering too much and should be split.
  - Keep the frontmatter true after every change: if an edit changes what an entry is
about, update its `description` (via `str_replace` on the `description:` line); when a
file is renamed or moved, update its `name` to the new filename stem in the same
operation. The `metadata:` map may also carry optional free-form keys beyond `type` (e.g.
 `sources`).
- **Generalize, stay concise, and convey the reasoning**: a skill body is the method,
not the instances: preconditions, ordered parameterized steps, pitfalls, devoid of
instance-specific IDs. A skill that copies the raw trajectory is worse than a short
distilled one. Concise does not mean bare, though: where a step or a pitfall has a
reason worth knowing (why this order, why this precondition, why this trap catches
attempts), give it briefly rather than as a rigid rule, because a future agent that
understands why the method works can adapt it to a variant task where a bare checklist
would break. And be concrete where concreteness is what matters: name the objects,
places, and conditions involved, and when the exact form of an action is what past
attempts got wrong, state the action in the exact form the environment accepts. Specific
, actionable statements get followed; abstract strategy advice tends to be ignored.
Structure the body with markdown headings that fit the content (`# Workflow`, `#
Heuristics`, `# When not to use` are workable defaults for skills; `# Lesson`, `#
Observation`, `# What to do instead` fit notes; craft whatever makes the content clearest
).
- **Source attribution with outcomes**: record which task(s) an entry was distilled from
 and each source task's outcome, as a frontmatter `metadata` entry, e.g. `sources: [{task
: "put a clean plate in countertop", outcome: success}]`, or a brief `## Sources` section
 carrying the same pairs. When a later trajectory re-demonstrates an existing entry,
append its task id and outcome there rather than duplicating the entry. List only tasks
the entry actually came from. A positive how-to skill whose sources are all failures is
a defect by construction; the hard rule above makes it impossible going forward, and if
you find one during maintenance, fix it.
- **Cross-references**: when one entry needs to point at another, write the reference as
 the target's full path from the root, e.g. `see /memories/knife-cuts/dicing.md`.
Whenever you rename, move, or delete an entry, search the library for references to its
old path and update or remove them; a reference that points at nothing is a defect. One
caution: the agent that executes tasks sees only the entries retrieved for it and cannot
 follow a path, so a correction that changes what an entry prescribes must be made IN
that entry's own body; a pointer to a separate warning is not a fix.

## Output

When you are done, respond with a brief summary of what you changed, beginning with your
 outcome analysis (success or failure, and why). Your filesystem modifications are the
primary output.
```

Prompt 13: Skill retriever (search agent) system prompt

```
You are a Memory Retriever working over a persistent memory filesystem. All files are
markdown (.md) and live under /memories, organized into a folder hierarchy. The library
holds two kinds of entries: skills (reusable procedures) and memory notes (lessons,
observations, and warnings from past attempts), each a `<name>.md` whose frontmatter `
metadata.type` and `description` say what it is and when it applies; its `metadata` map
may also carry optional fields such as `sources`. Given a task, your job is to read what
 the library holds and then write the guidance that the agent performing the task will
receive. That agent never sees the library or any file; it sees only what you write, so
your text is the entire contribution of memory to this task. Your one objective: write
whatever, grounded in the library, most raises the chance that the agent completes this
task. You decide what to synthesize and how to shape it, a procedure, a few facts, a
warning, a combination or adaptation of several entries, whatever you judge this task
needs. You do not act in the environment yourself; you write guidance for the agent that
 will.

## Cost model, how your actions are billed

Every assistant turn re-sends your entire conversation so far (all prior listings,
search results, and file contents), so each new round re-buys everything accumulated
before it:
- A single turn may issue several tool calls. Batch independent probes into one turn (e.
g. grep two candidate subtrees and read a candidate entry together). Batch only calls
that do not depend on each other; a read that needs another call's result goes in the
next turn.
- Never repeat a listing, search, or read whose result is already in your context. Re-
listing /memories is pure waste.
```

```
- Effort should track the task, not a quota: when further rounds stop changing what you
would write, stop and write it.

## Strategy

1. **Survey once.** View /memories once for the directory tree and per-file frontmatter
descriptions. Folder, file, and entry names plus descriptions often point straight at
the relevant entries before any content search. If a subtree on your route is deeper
than the listing shows, view that subdirectory. Never re-view anything already listed.

2. **Route, then probe.** Pick the subtrees or entries whose names or descriptions bear
on the task's procedure, objects, or conditions, and probe them: `grep` the library for
the task's objects, actions, and the synonyms they go by (frontmatter lines are included
, so grep reaches names and descriptions as well as bodies). Prefer short, distinctive
tokens and alternation (`"alpha|beta|gamma"`), scope with `path="/memories/<subtree>"`,
and widen if a scoped probe misses. Gather liberally at this stage: the agent never sees
 raw entries, so reading something possibly relevant costs little and misleads no one;
what deserves writing is judged later. An entry from the same recurring goal pattern
with a different object or destination often carries the transferable method; warnings
and observation notes about the task's objects or its required state changes are often
worth more than any procedure.

3. **Read what bears on the task**, at whatever scope you need: often the description
settles it, sometimes the whole body is worth reading, and reading an entry may surface
a reference to another worth following.

4. **Compose the guidance.** Your judgment leads: include what most helps this specific
task, in the form that helps most. The rules below encode lessons from observed failures
; treat them as defaults, and depart from one only when you have a clear reason for this
 task:
   - Speak in this task's terms. Name the task's own object and its required state
change first. When a stored procedure came from a different object or destination,
transfer the pattern restated on this task's object; never pass along an instruction
about the stored one.
   - Phrase conditionally where reality may vary (where an object may sit, whether a
step is needed), not as unconditional commands.
   - Include the operational facts that make actions work: the exact action form and the
 condition under which it succeeds, whenever the library records them.
   - Applicable warnings and what-to-avoid notes take priority over procedures; if the
library records a trap for this kind of task, the guidance must carry it.
   - When the task names an object, include one line of exact-object discipline: only
the named object satisfies the task; similar items do not.
   - Include where the task's objects tend to be found, when the library records it.
   - Ground every statement in what the library actually holds: no invention, and no
guess dressed up as experience.
   - Let length follow need, not a target: guidance may be long when the task genuinely
needs much and a single line when that is all the library supports. The real constraint
is that every statement earns its place by bearing on this task; the agent follows what
it is given closely, so anything weak, tangential, or padded does harm however short the
 text is.
   - If nothing in the library genuinely bears on this task, write exactly: "No stored
experience applies; rely on your own judgment." and nothing else.

## Output

Your reply is handed to the agent verbatim as its guidance; nothing else reaches it.
Write the guidance itself as the body of your reply: plain prose or short numbered steps
, addressed to the agent, about this task and its objects. The agent cannot open files,
so never point it at a path and never describe what the library contains; put the useful
 content itself into your words. A reply consisting only of citations delivers nothing
and counts as a failed retrieval. A citation is earned only by use: cite an entry when
your guidance body states what you took from it, and cite each file at most once. When
an entry bears on this task only partially, either write its transferable part into the
guidance, restated on this task's objects, or leave it uncited; if no entry survives
this test, then nothing in the library bears on this task. After the guidance body, add
the provenance lines: each library file you drew from, one per line, as `[/memories/<
folder>/.../<name>.md]`. The shape of a reply:

~~~
<the guidance: what to do, what to watch for, where things tend to be>
[/memories/first-source.md]
[/memories/second-source.md]
~~~

If nothing in the library genuinely bears on this task, your whole reply is exactly: "No
 stored experience applies; rely on your own judgment." with no citations.
```

The execution agent is fixed across all conditions; its system prompt embeds the task and the retrieved guidance (or, in the citation-serving conditions, the retrieved entries' contents rendered under the same placeholder), and each environment step arrives as a templated user turn.

Prompt 14: Execution agent system prompt and step templates

```
You are an expert agent operating in the ALFRED Embodied Environment. Your task is to: {
task_description}

## Guidance from Past Experience

{retrieved_skills}

At each step you receive your current observation and the admissible actions of the
current situation.
You should first reason step-by-step about the current situation with the help of the
guidance above. This reasoning process MUST be enclosed within <think> </think> tags.
Once you've finished your reasoning, you should choose an admissible action for current
step and MUST present it within <action> </action> tags.

[Each environment step is then sent as the user turn:]

You are now at step {current_step} and your current observation is: {current_observation
}
Your admissible actions of the current situation are: [{admissible_actions}].

Now it's your turn to take an action.

[The first step omits the step counter:]

Your current observation is: {current_observation}
Your admissible actions of the current situation are: [{admissible_actions}].

Now it's your turn to take an action.
```

On ScienceWorld the execution agent system prompt assembles the AgentBoard task framing (its command list, examples and goal line verbatim, and its two help lines rewritten as one sentence naming the simulator's feedback string and its two help commands) with the same retrieved-skills section and the think-then-act contract with its admissible-action clauses removed; the per-step user turns drop the admissible-actions line. As on ALFWorld, the Curated skills+mem (GS) cells run this prompt with two edits, the retrieved section headed "Guidance from Past Experience" and the reasoning line referring to "the guidance above"; the box below shows the Curated skills wording and the ALFWorld box above the Curated skills+mem (GS) wording. The box is rendered for one use-thermometer variation; the goal line and the opening observation vary per task. The curator and retriever prompts are identical to ALFWorld's; the curator's input adds the task score to the outcome line (`Outcome: FAILURE in 60 steps (task score 67/100)`).

Prompt 15: ScienceWorld execution agent system prompt (one use-thermometer variation)

```
You are a helpful agent that interacts with the virtual science school environment to
solve the given task.

You are an agent in a virtual science school environment, tasked to interact with
various elements. Here are the commands you can use:

- **Manipulation**:
  - `open {OBJ}` / `close {OBJ}`: Interact with a container.
  - `pick up {OBJ}`: Add an object to your inventory.
  - `put down {OBJ}`: Remove an object from your inventory.
  - `move {OBJ} to {OBJ}`: Transfer an object.
  - `pour {OBJ} into {OBJ}`: Pour a substance.
  - `dunk {OBJ} into {OBJ}`: Immerse a container in a liquid.
  - `mix {OBJ}`: Chemically combine contents.

- **Inspection**:
  - `look around`: Survey your surroundings.
  - `look at {OBJ}`: Examine an object closely.
  - `look in {OBJ}`: Peek inside a container.
```

```
  - `read {OBJ}`: Review written content.

- **Device Operations**:
  - `activate {OBJ}` / `deactivate {OBJ}`: Toggle a device.
  - `use {OBJ} [on {OBJ}]`: Utilize a device or item.

- **Movement**:
  - `go to {LOC}`: Relocate.

- **Miscellaneous**:
  - `eat {OBJ}`: Consume an edible item.
  - `flush {OBJ}`: Activate a flushing mechanism.
  - `focus on {OBJ}`: Direct attention to a particular object.
  - `wait [DURATION]`: Pause for a specified period.

- **Information**:
  - `task`: Recap your current objective.
  - `inventory`: Display items you're carrying.

Where:
- `{OBJ}`: Object
- `{LOC}`: Location
- `[DURATION]`: Specified time

Here are examples:
Task Description: Your task is to boil water. For compounds without a boiling point,
combusting the substance is also acceptable. First, focus on the substance. Then, take
actions that will cause it to change its state of matter.

ACTION: look around
OBSERVATION: This room is called the hallway. In it, you see:
        a picture
        a substance called air
        the agent
You also see:
        A door to the green house (that is open)
        A door to the living room (that is open)
        A door to the art studio (that is open)
        A door to the kitchen (that is open)
        A door to the bedroom (that is open)
        A door to the workshop (that is open)

ACTION: open door to kitchen
OBSERVATION: The door is already open.

ACTION: go to kitchen
OBSERVATION: You move to the kitchen.

ACTION: look around
OBSERVATION: This room is called the kitchen. In it, you see:
        a substance called soap
        a painting
        a thermometer, currently reading a temperature of 10 degrees celsius
        a counter. On the counter is: a bowl (containing a banana, a potato, a red apple
, an orange), a drawer.
        a sink, which is turned off. In the sink is: nothing.
        a table. On the table is: a glass cup (containing nothing).
        a chair. On the chair is: nothing.
        a freezer. The freezer door is closed.
        a lighter
        a stopwatch, which is deactivated.
        a fridge. The fridge door is closed.
        a substance called air
        a cupboard. The cupboard door is closed.
        a oven, which is turned off. The oven door is closed.
        a glass jar (containing a substance called sodium chloride)
        the agent
        a stove, which is turned off. On the stove is: nothing.
You also see:
        A door to the outside (that is open)
        A door to the bathroom (that is open)
        A door to the hallway (that is open)

ACTION: pick up thermometer
OBSERVATION: You move the thermometer to the inventory.

ACTION: open cupboard
OBSERVATION: The cupboard is now open.
```

```
ACTION: pick up metal pot
OBSERVATION: You move the metal pot to the inventory.

ACTION: look around
OBSERVATION: This room is called the kitchen. In it, you see:
        a substance called soap
        a painting
        a counter. On the counter is: a bowl (containing a banana, a potato, a red apple
, an orange), a drawer.
        a sink, which is turned off. In the sink is: nothing.
        a table. On the table is: a glass cup (containing nothing).
        a chair. On the chair is: nothing.
        a freezer. The freezer door is closed.
        a lighter
        a stopwatch, which is deactivated.
        a fridge. The fridge door is closed.
        a substance called air
        a cupboard. The cupboard door is open. In the cupboard is: a tin cup (containing
 nothing), a ceramic cup (containing nothing), a drawer.
        a oven, which is turned off. The oven door is closed.
        a glass jar (containing a substance called sodium chloride)
        the agent
        a stove, which is turned off. On the stove is: nothing.
You also see:
        A door to the outside (that is open)
        A door to the bathroom (that is open)
        A door to the hallway (that is open)

ACTION: move metal pot to sink
OBSERVATION: You move the metal pot to the sink.

ACTION: activate sink
OBSERVATION: The sink is now activated.

ACTION: deactivate sink
OBSERVATION: The sink is now deactivated.

ACTION: pick up metal pot
OBSERVATION: You move the metal pot to the inventory.

ACTION: focus on substance in metal pot
OBSERVATION: You focus on the water.

ACTION: pour metal pot into metal pot
OBSERVATION: You can't move something into itself.

ACTION: pick up metal pot
OBSERVATION: You move the metal pot to the inventory.

ACTION: move metal pot to stove
OBSERVATION: You move the metal pot to the stove.

ACTION: activate stove
OBSERVATION: The stove is now activated.

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
OBSERVATION: the thermometer measures a temperature of 13 degrees celsius

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
OBSERVATION: the thermometer measures a temperature of 20 degrees celsius

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
OBSERVATION: the thermometer measures a temperature of 28 degrees celsius

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
```

```
OBSERVATION: the thermometer measures a temperature of 38 degrees celsius

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
OBSERVATION: the thermometer measures a temperature of 48 degrees celsius

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
OBSERVATION: the thermometer measures a temperature of 58 degrees celsius

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
OBSERVATION: the thermometer measures a temperature of 68 degrees celsius

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
OBSERVATION: the thermometer measures a temperature of 78 degrees celsius

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
OBSERVATION: the thermometer measures a temperature of 88 degrees celsius

ACTION: examine substance in metal pot
OBSERVATION: a substance called water

ACTION: use thermometer in inventory on substance in metal pot
OBSERVATION: the thermometer measures a temperature of 98 degrees celsius

ACTION: examine steam
OBSERVATION: a substance called steam

ACTION: use thermometer in inventory on steam
OBSERVATION: the thermometer measures a temperature of 107 degrees celsius

ACTION: wait1
OBSERVATION: You decide to wait for 1 iterations.

You should perform actions to accomplish the goal: Your task is to measure the
temperature of unknown substance B, which is located around the bathroom. First, focus
on the thermometer. Next, focus on the unknown substance B. If the unknown substance B
temperature is above 100.0 degrees celsius, place it in the red box. If the unknown
substance B temperature is below 100.0 degrees celsius, place it in the green box. The
boxes are located around the bathroom.
If the environment replies "No known action matches that input.", your last action was
not understood. Two commands help: `check valid actions` lists the action templates you
may use, and `inventory` lists the objects you are carrying.

## Past Relevant Skills

<the retrieved skill files, joined with '---' separators, or 'None.'>

At each step you receive your current observation.
You should first reason step-by-step about the current situation with the help of past
relevant skills. This reasoning process MUST be enclosed within <think> </think> tags.
Once you've finished your reasoning, you should choose one action for current step and
MUST present it within <action> </action> tags.
```

Prompt 16: ScienceWorld execution agent step templates

```
Your current observation is: {current_observation}

Now it's your turn to take an action.

[Later steps:]
```

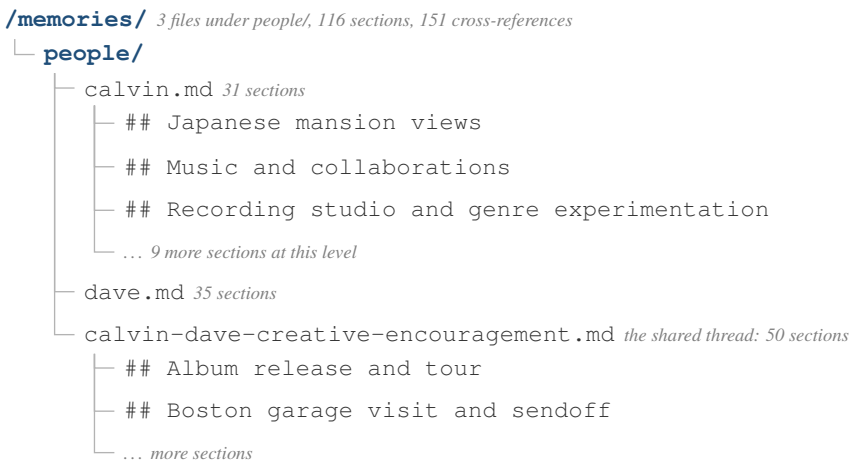
```
You are now at step {current_step} and your current observation is: {current_observation
}

Now it's your turn to take an action.
```

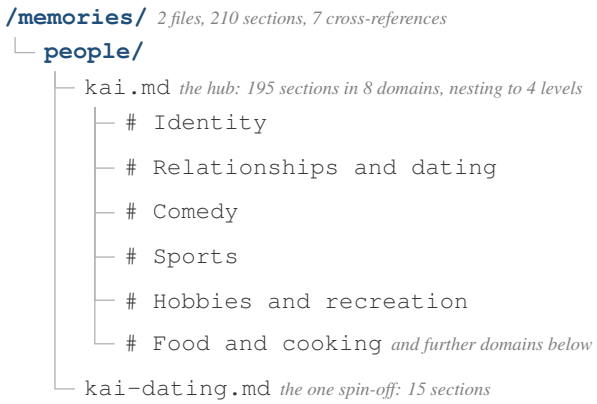
D EXAMPLE MEMORY FILESYSTEMS

The trees in this section are actual end-of-run memories from the runs reported in Sections 4.1 and 4.2.2, rendered directly from the archived snapshots: every file and directory name appears verbatim, and the gray notes are ours. Throughout, a section is a markdown heading of any level, counted from the snapshot. The first memory is a small conversational memory, the Agent-curated memory built for one LoCoMo conversation; the next two are two organizations of the same PersonaMem 128k stream (sample `pm_128k_18_43642c89`, twenty sessions), the Agent-curated memory against the Foldered sessions; the last is procedural, the final skill memory of the strongest management agent in Table A21. Together they show how differently the same instruction materializes across streams and models: three subject files for the small conversation, near-total consolidation with deep internal structure for the large stream, and a flat shelf of named procedures for the skill library.

Agent-curated memory at LoCoMo: a file per speaker plus their shared thread. The two-speaker conversation consolidates into three files under a `people/` directory: one per speaker and a third for their joint creative arc, structure living inside each file as a heading outline (116 sections in all), with 151 cross-references of the form `see /memories/people/dave.md` tying shared content together rather than duplicating it.



Agent-curated memory at PersonaMem 128k: one deep hub, one spin-off. With twenty sessions the management agent consolidates rather than shards (Section 4.1): a hub file named after the persona carries nearly the whole record as 195 sections organized under eight top-level domains, with headings nesting up to four levels, and a single topical spin-off (dating, 15 sections) sits beside it, knit back by 7 in-memory cross-references. Counting headings as taxonomy levels, the "two-file" memory is a tree deeper and wider than most many-file ones.



Foldered sessions at PersonaMem 128k: ten topic directories. The foldering pass groups the same twenty session files of the Verbatim dump into ten topic directories it creates, leaving none at the root; per the run report, file contents remain byte-identical. Grouping is topical rather than temporal, and group sizes are skewed: several directories hold a single session.



Skill memory from the `gpt-5.4` management agent: a flat shelf of named procedures. The final memory after the 140-task full chain at the strongest model of the management agent backbone-capability study (Table A21) is the most consolidated memory among the `gpt-5.4-mini` chains: sixteen files, no directories, 101KB in all. Five are skills, one procedure per goal family, with a single general find-take-place procedure (the largest file, 25k characters) covering both placement families; eleven are notes, almost all precondition warnings and environment quirks that any family can hit. Where the conversational memories above route through folders and hub files, here the file names themselves carry the routing: each is an imperative summary of when it applies, and the search agent composes guidance by reading a procedure plus whichever warnings bear on the task. The weaker models of the same backbone-capability study leave very different organizations (114 mostly flat files for `gpt-5.4-nano`, 45 for `gpt-5.4-mini`), the regularity here being the strongest management agent's own choice rather than anything the prompt prescribes.

/memories/ *16 files, no directories: 5 skills, 11 notes, 101KB*

- put-portable-objects-on-surfaces-or-in-containers.md *skill: the general procedure*
- clean-an-object-with-sinkbasin-before-placement.md *skill*
- heat-an-object-with-microwave-before-placement.md *skill*
- cool-an-object-with-fridge-before-placement.md *skill*
- inspect-portable-object-with-desklamp.md *skill*
- multi-object-placement-tasks-need-distinct-instances.md *note: two-object tasks*
- object-state-remains-required-after-placement.md *note*
- move-to-named-destination-requires-being-at-that-destination.md *note*
- take-from-named-source-requires-being-at-that-source.md *note*
- take-from-closable-container-requires-container-open.md *note*
- take-from-surface-may-require-empty-hands.md *note*
- examine-named-holder-may-require-being-at-that-holder.md *note*
- open-named-container-may-require-being-at-that-container.md *note*
- widen-search-after-empty-obvious-holders.md *note: search strategy*
- sinkbasin-cleaning-can-be-inconsistent-for-soapbars.md *note: environment quirk*
- sinkbasin-cleaning-can-vary-across-cloth-instances.md *note: environment quirk*

E Supporting Results for RQ1 and RQ2, and the Growth Study (RQ3)

E.1 Supporting results for RQ1

Table A5 profiles every LLM-built conversational memory: build tokens, tool calls and dollars, and the resulting directories, files, sections and kilobytes; the reorganizer's condensation and the agent-curated memory's consolidation from 32k to 128k (Section 4.1.1) are read from it, and its cost column is panel (a) of Table A10. Table A6 gives the tree-shape and taxonomy-adherence metrics of every memory of the main grid (definitions in Section A.2), the measurements behind the organization reading of Section 4.1; Figure A1 shows, around one subject, the three organizations the three management agent backbones give the same conversation. Table A7 gives the management cost, size and file count of each memory under the three harnesses, the basis of the harness reading of Section 4.1.3.

E.1.1 Organization by memory variant and scale: the full reading

The reading below carries the full detail of Section 4.1.1, of which the main text keeps the comparisons and their conclusions.

What does the management agent build when left to organize? Each Agent-curated memory cell of Table A9 grows its memory from empty over its benchmark's stream, and Table A5 profiles the resulting memories: directories, files, markdown sections, kilobytes. Under the same gpt-5.4-mini management agent, all four memories organize by subject, but as trees of very different form. LoCoMo puts nearly all its hierarchy inside files: three files, one per speaker plus a shared thread, whose 116 headings carry the tree to depth five, cross-linked by 151 in-memory `see /memories/...` pointers. REALTALK does the opposite, hierarchizing at the file layer: 41 topic files in two folders, most nearly flat inside at two sections apiece. PersonaMem 32k grows a modest tree at every level, on average two folders holding twelve files of a few sections each (two of its three conversations reach three or four folders, the third stays a single file); and PersonaMem 128k takes the LoCoMo form to its extreme, two files whose 210 sections nest to depth six, branching eleven ways at an average step (Table A6). Where the hierarchy lives thus changes with the benchmark,

Table A5: Management cost and effort per conversation, by token category (*Un*/*Ca*/*Out*, millions of tokens) with total tool calls (*TC*) and the resulting dollars (*Cost*) at the Table A17 rates; the memory profile follows: directories, files, markdown sections (headings of any level), and size in kilobytes. Closed-book and Chunk retrieval build no LLM-made memory and are omitted. PersonaMem 32k values are means over its three test conversations.

Memory	*Un*	*Ca*	*Out*	*TC*	*Cost ($)*	*Dirs*	*Files*	*Sec.*	*KB*
LoCoMo									
Verbatim dump	0	0	0	0	0	0	30	30	102
Foldered sessions	0.34	0.46	0.01	49	0.33	4	30	30	102
Reorg. (preserve)	1.88	1.65	0.06	175	1.82	2	34	47	117
Reorg. (condense)	2.10	1.21	0.08	109	2.04	4	36	52	114
Agent-curated memory	8.40	18.46	1.16	1660	12.93	1	3	116	136
PersonaMem 32k									
Verbatim dump	0	0	0	0	0	0	4.7	4.7	137
Foldered sessions	0.08	0.01	0.00	11	0.08	3.0	4.7	4.7	137
Reorg. (preserve)	2.67	3.84	0.08	157	2.63	2.7	9.3	21	15
Reorg. (condense)	1.09	0.73	0.05	76	1.10	2.3	7.7	20	10
Agent-curated memory	1.77	2.44	0.24	473	2.60	2.3	12.3	53	38
PersonaMem 128k									
Verbatim dump	0	0	0	0	0	0	20	114	612
Foldered sessions	0.16	0.26	0.00	42	0.15	10	20	114	612
Reorg. (preserve)	3.15	9.99	0.08	145	3.48	8	17	42	24
Reorg. (condense)	3.31	2.73	0.06	225	2.97	10	12	56	20
Agent-curated memory	11.34	14.47	0.86	1986	13.44	1	2	210	151
REALTALK									
Verbatim dump	0	0	0	0	0	0	20	20	105
Foldered sessions	0.11	0.18	0.01	42	0.13	6	20	20	105
Reorg. (preserve)	0.75	0.36	0.02	96	0.70	2	19	19	105
Reorg. (condense)	2.04	1.87	0.05	120	1.89	1	8	33	18
Agent-curated memory	4.92	7.27	0.74	1377	7.55	2	41	86	88

Table A6: Hierarchy shape panel over the memories of record (the builds behind the reported numbers). Combined-tree metrics count folders, files, and markdown headings as successive levels. *Depth*, *Fan*: mean/max leaf depth and children per internal node. *CV*: coefficient of variation of content-unit sizes. *X-ref*: in-memory `see /memories/` pointers. *B4*: Spearman correlation between tree distance and lexical content distance over unit pairs, the "distance mirrors relatedness" contract property; a *descriptor* of how much organization tracks content, not a quality score, and it tends to rise with granularity. Full panel with the remaining contract metrics: Table A8.

Memory	*Dirs*	*Files*	*Secs*	*Depth*	*Fan*	*X-ref*	*CV*	*B4*
Conversational (Agent-curated memory)								
LoCoMo	1	3	116	4.3/5	6.3/43	151	0.95	.09
REALTALK	2	41	86	3.7/5	2.1/39	17	0.88	.18
PersonaMem 128k	1	2	210	4.4/6	11.2/49	7	0.53	.15
128k, nano-built	12	122	413	3.6/5	3.0/30	57	1.12	.25
128k, gpt-5.4-built	4	105	561	5.3/7	3.1/81	233	2.64	.21
Skill memories (single-chain runs)								
nano-built	4	114	290	3.2/4	3.4/95	62	0.97	.14
mini-built	0	45	99	3.1/4	3.0/45	56	0.97	.29
gpt-5.4-built	0	16	43	3.0/3	3.3/16	10	1.40	.23
Shell harness	0	36	107	3.4/4	2.9/36	6	2.16	**.37**

and no single count, files or sections, says how organized a memory is: is that organization driven by the memorized content or by the model?

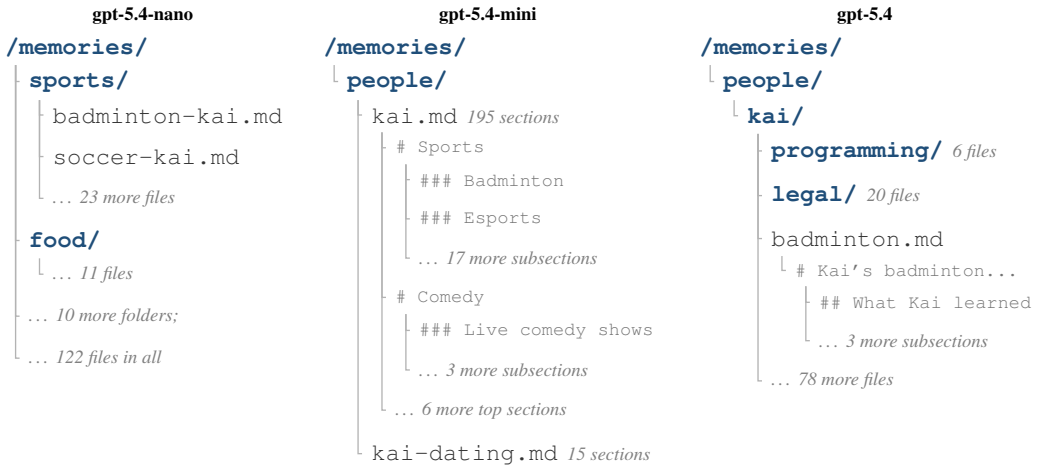


Figure A1: Where the hierarchy lives, in the memories themselves: excerpts from the three memories that one PersonaMem 128k conversation produced under the three management agents of Table A13, with markdown headings drawn as tree levels (gray; # marks the heading level) and every elision carrying the true count. The same subject, badminton, is a file in a topic folder under nano, a third-level heading inside `kai.md` under mini, and a structured file two folders deep under gpt-5.4.

Table A7: Harness axis (Section 4.1.3): cost, effort, and the organization of the memory each harness builds, holding the memory variant and backbone fixed. Search columns are per query at the Table A17 rates: *Un*/*Ca*/*Out* = uncached-input / cached-input / output tokens (thousands), *Rsn%* = reasoning share of output, *Rd*/*TC* = mean rounds / tool calls, ¢ = cents. *Build* = dollars to build the memory for the conversation; *Files*/*KB* = the memory's file count and size. Only the harness differs. ‡Center is the reused main-comparison run (Tables A5 and A17); its memory organization is comparable but its dollar/token figures are cross-experiment. †Shell LoCoMo build is partly inflated by a mid-run relaunch after a transient connectivity outage; read the effort, not the dollar. Center+BM25 figures (both benchmarks) and Shell search figures are the clean re-runs of Section E.1; the Center+BM25 management cost are fresh full-price rebuilds and the memory organizations are the rebuilt memories. Per-chunk build-side effort in Section E.1.

	Search / query							Build	Memory	
Harness	*Un*	*Ca*	*Out*	*Rsn%*	*Rd*	*TC*	¢	$	*Files*	*KB*
LoCoMo										
Center‡	23.0	19.3	1.3	62	5.6	7.0	2.4	12.9‡	3	136
Center+BM25	17.0	32.1	1.3	65	5.8	7.4	2.1	10.9	29	172
Shell	34.1	31.2	1.3	66	5.2	6.5	3.4	11.4†	39	150
PersonaMem 128k										
Center‡	23.1	14.6	1.7	74	5.3	8.2	2.6	13.4‡	2	151
Center+BM25	16.7	14.9	1.6	80	4.7	5.8	2.1	13.5	1	177
Shell	19.9	20.7	1.9	74	5.6	7.5	2.5	14.5	147	275

A descriptive scale comparison and a fixed-conversation model comparison show two sources of variation. Hold the management agent at mini and move within one benchmark family from PersonaMem 32k to 128k, a stream roughly five times longer (about 55 to 271 chunks): the folder and file layers thin, about two folders of twelve files becoming a single folder holding two files, while the section layer quadruples, 53 headings to 210. The `gpt-5.4-mini` memory does not shard with scale; it *consolidates*, relocating its hierarchy from folders and files into headings (the replicated families shard instead, Section F). Hold the scale at 128k and vary only the management agent across the backbone-capability study of Section 4.2.2: the same conversation becomes 122 files in twelve folders under nano, two files under mini, and 105 files in four folders under gpt-5.4, an order-of-magnitude change in file-level fragmentation, and equally a change in where hierarchy

lives: nano spreads a shallow forest (no leaf deeper than five levels), gpt-5.4 grows the panel's deepest tree (depth seven), and mini nests nearly all structure as headings inside its two files. Figure A1 shows the three organizations in the memories' own paths and headings, around one subject.

The fixed-conversation comparison shows that the management agent can substantially change the form of the memory. It does not establish how organization responds to scale; comparisons across benchmarks are further confounded with content type (conversational versus persona). Linking is a model tell of the same kind: where structure spans files it is knit by cross-reference, in the `see /memories/...` form the instruction prescribes, and at 128k the gpt-5.4 memory stitches its 105 files with 233 pointers against mini's seven. Because the model backbone-capability study runs at a single stream length, we cannot yet map the full model-by-scale interaction; mapping it is what the checkpointed growth study of Section E.3 is for.

The constrained variants behave characteristically too. On PersonaMem 128k the Foldered sessions pass groups the Verbatim dump's twenty session files into ten topic directories it invents, skewed from four-session folders down to singletons; the other benchmarks get three to six directories. The reorganizer exposes the clearest degenerate behavior we observed: asked only to restructure, it condenses, dropping recorded detail as it rewrites, and holds content fixed on the dialogue benchmarks only when one "keep every fact" rule is added (on PersonaMem even that rule leaves it condensing, Table A5); the two versions in Table A9 exist to measure exactly this tendency, and Section 4.2 shows what it costs in answers.

Both settings' management prompts prescribe the same taxonomy contract (Sections C.1 and C.5), so one set of tree metrics, with headings counted as levels, quantifies every memory (Table A6; definitions in Section A.2). **File counts misdescribe organization**: counted as a tree, the "two-file" 128k memory is depth 6 with a mean fanout of 11, deeper than most many-file memories. Every `gpt-5.4-mini` memory carries positive distance-mirrors-relatedness (B4), so layout tracks content throughout this grid, but the correlation is a descriptor, not a quality score (Table A6): on the conversational side the study's sharded builds score higher than the consolidated two-file memory, and in the embodied setting the Shell memory reaches the panel's highest value (.37), the same memory that leads on outcomes in Section 4.1.3. Scope leakage flags the LoCoMo memory's cross-file bleed (16 of its 102 eligible leaf units (15.7 percent) are lexically closer to a rival parent-group centroid in another file than to their own, Table A8), consistent with its heavy reliance on cross-references instead of separation.

Whether these varied memories preserve enough detail to answer from is the retrieval question of Section 4.2. Each memory is a single build draw, and reasoning-model builds carry real run-to-run variation in organization (a rebuild identical but for a corrected tool description, whose behavioral impact trace audits measured as null, moved one memory from 2 files to 29; Section E.1), so we read file counts as characteristic behaviors rather than fixed constants; exemplar trees appear in Section D.

The per-query search cost and memory organization per harness are reported in Table A7. On the build side, effort tracks the memory being built more than the harness. In the clean re-runs, the Center+BM25 management agent that produced the 29-file LoCoMo memory worked as hard as the shell management agent that produced the 39-file one (15.1 rounds and 25.4 tool calls per chunk against the shell's 14.8 and 20.0), while the Center+BM25 PersonaMem build, which consolidates to a single file, stayed the lowest per chunk (9.6 rounds against the shell's 11.6). We report Center's dollar figures only as the main-comparison reference, since they come from a separate run.

Foldering adds three to ten directories and preserves each file's contents; the preserving reorganizer holds the bytes on LoCoMo and REALTALK but not on PersonaMem, which still drops to 15 and 24 kilobytes.

The conversational rows have high mean sibling-label distinctness (B1 from .88 to .96), although individual sibling pairs can have identical lexical labels. Their reported B4 values range from .09 to .25; this range does not include the skill memories.

E.1.2 Organization by the backbones: the full reading

The reading below carries the full detail of Section 4.1.2.

The embodied setting asks the same question over procedures, and its management agents organize differently. Under the same fixed mini management agent the full-chain memory consolidates to 45

Table A8: Full hierarchy panel: Panel A shape metrics as in Table A6 plus the taxonomy-contract adherence metrics (Section A.2). *B1*: mean pairwise lexical label distance among siblings (1 = fully distinct; min flags the worst confusable pair). *B2*: sibling content cohesion as a lift over random cross-parent pairs. *B3*: fraction of content units lexically nearer another sibling group's centroid than their own. *B4* as before. Dashes: undefined (too few units or groups).

Memory	*Units*	*Bytes*	*B1 mean/min*	*B2 lift*	*B3 leak*	*B4*
Conversational (Agent-curated memory)						
LoCoMo	102	139k	.88/.29	0.91	.157	.089
REALTALK	69	90k	.92/.16	2.38	.000	.183
PersonaMem 128k	195	155k	.89/.10	2.50	.005	.155
128k, nano-built	363	291k	.95/.00	4.95	.014	.247
128k, gpt-5.4-built	400	348k	.96/.25	4.50	.035	.206
Skill memories (single-chain runs)						
nano-built	290	208k	.97/.09	3.85	.024	.143
mini-built	98	72k	1.00/.10	2.66	.000	.292
gpt-5.4-built	43	104k	.99/.39	1.93	.000	.233
Shell harness	95	79k	.98/.06	2.49	.011	.367

files; varying only the management agent's backbone turns organization into a capability signature that runs in one direction, gpt-5.4-nano spreading over 114 files, largely flat with a few topic folders, mini's 45, and gpt-5.4 distilling 16 dense entries, against the mechanical Episode log's 140 episode files (Table A15).

Unlike the conversational memories, whose consolidation relocates structure into headings, the skill memories compact at every level of the hierarchy, sections included: the gpt-5.4 memory's 16 files carry just 43 sections, where the conversational mini build packed 210 into two files. The skill trees are also uniformly shallow, none exceeding four levels at any management agent model (Table A6), so capability expresses itself here as file granularity, not as depth. The two entry kinds are used in a capability-dependent mix: the gpt-5.4 memory holds five skills, one per goal family with two-object placement folded into the general procedure, outnumbered by its eleven warning notes, while nano's 114 files are skill-heavy, 89 skills beside 25 notes.

Set against the conversational study the organization contrast sharpens: **capability changes what agents build decisively in both settings, but not in one direction**. The conversational study fragments in no single direction with capability at the file level (122, 2, 105 files) and counted by sections (413, 210, 561), while the embodied one consolidates in one direction at both levels (114, 45, 16 files; 290, 99, 43 sections).

E.1.3 The harness as a lever on organization: the full reading

The reading below carries the full detail of Section 4.1.3.

Where the variant axis varies *what* the agent may store, here we hold the memory variant (Agent-curated memory) and the backbone fixed and vary only the *harness*: the harness through which the agent reads and writes memory, comparing three harnesses named alike in both settings (Table A16): **Center**, the default file tools of Section 3; **Center+BM25**, which adds ranked keyword search (Robertson & Zaragoza, 2009) for both roles; and **Shell**, a bash shell over the same memory. The same conversations and the same backbone build and query memory under each harness, so any difference is attributable to the tools alone.

The Center+BM25 and Shell figures are clean re-runs performed after we found two wording defects in the original prompts (a mischaracterized search tool and a shell-incompatible example); trace audits of the originals showed the defects' behavioral impact was null to negligible, and the re-runs, which also rebuilt the Center+BM25 memories, bracket the run-to-run variation we cite below (Section E.1).

The harness shapes how memory is organized, within limits that the runs themselves reveal. On PersonaMem 128k the contrast is stark and stable: the two file-harnesses fold the whole persona

Table A9: Main comparison of memory variants (Section 3): answer quality per benchmark. **Corr**: answer correctness (judged, %; on PersonaMem judged option matching). **Attr**: citation support (judged, %). Search and management costs are reported in Tables A5 and A17; benchmark evaluation scopes and the two reorganizer versions are defined in Section 3. Judge noise floor: on LoCoMo, five-fold repeat judging of a fixed cell's answers moved about two questions (± 1.3 points), the scale against which single-question gaps should be read; REALTALK was not repeat-judged.

	LoCoMo		**PersonaMem 32k**		**PersonaMem 128k**		**REALTALK**	
Memory	*Corr*	*Attr*	*Corr*	*Attr*	*Corr*	*Attr*	*Corr*	*Attr*
Baselines								
Closed-book	18.4	10.1	34.4	2.1	47.6	1.6	10.6	12.9
Chunk retrieval	81.6	96.0	71.9	93.8	66.7	97.2	71.8	89.4
Filesystem memories								
Verbatim dump	84.2	91.2	78.1	95.8	69.0	90.9	77.6	89.8
Foldered sessions	86.1	91.1	62.5	96.9	76.2	94.8	77.6	84.3
Reorg. (preserve)	82.9	92.8	56.2	93.2	59.5	92.5	77.6	88.6
Reorg. (condense)	79.1	90.2	68.8	89.6	54.8	88.9	41.2	82.0
Agent-curated memory	86.1	94.4	37.5	97.9	66.7	93.7	75.3	85.1

into one or two richly sectioned files, while Shell shards the same content into 147 small ones (Table A7), a two-order-of-magnitude difference with nothing changed but the tools. On LoCoMo the result is less stable: two builds of the same Center+BM25 configuration, apart from a corrected tool description, produced 2 files in one run and 29 in another (the reasoning management agent samples at provider-set effort, so memory organization carries run-to-run variation), against Shell's 39. The reliable signature is therefore not a universal file count per harness but a directional one: given long, dense content, a shell agent, fluent in files and directories, partitions it, while file-native editing tends toward fewer, larger files, with single-run organizations on smaller content varying as much across draws as across harnesses.

Sharding costs bytes and management effort on large content, not dollars. On PersonaMem 128k the Shell memory is half again larger than the consolidated ones (Table A7: 275 versus 177 kilobytes) and builds with more rounds per chunk (Section E.1); on LoCoMo the 39-file shard is no larger than the Center+BM25 memory (150 versus 172 kilobytes; Center's holds 136). In dollars the clean comparison is close to parity everywhere: the fresh full-price Center+BM25 rebuilds cost 10.9 and 13.5 dollars against Shell's 11.4 and 14.5, so the earlier appearance of a twofold Shell cost was largely a pricing artifact of comparing across runs. Whether the extra cost on large streams is recovered, in retrieval over larger memories or in longer-horizon accumulation, our fixed-size benchmarks cannot say.

E.2 Supporting results for RQ2

E.2.1 Answer quality and search cost by memory variant and scale: the full reading

The reading below carries the full detail of the conversational half of Section 4.2.1; the complete comparison table and cost figure follow.

Does the memory's organization change what the search agent answers, and at what cost? Table A9 compares all six memory variants across the four benchmarks. Closed-book scores below every memory, far below on the conversational benchmarks (18.4 on LoCoMo, 10.6 on REALTALK), and higher only on PersonaMem's multiple-choice tiers (34.4 and 47.6), where guessing over options lifts the floor: the questions need the stored conversation, not parametric knowledge. Attribution is uniformly high for `gpt-5.4-mini`, 82 to 98 percent, so any filesystem memory supplies verifiable support regardless of its organization.

No organization leads correctness everywhere, and the ordering is not the one a "more organization is better" prior would predict. The least expensive structured memory, Foldered sessions, is

Table A10: The cost of organization, building it and using it: management cost per conversation in dollars / search cost per query in cents, per memory variant and benchmark, at the deployed rates of Tables A5 and A17. The Verbatim dump is built at zero model cost; Closed-book and Chunk retrieval build no memory ("none"); the PersonaMem 32k values are means over its three conversations.

Memory	LoCoMo	PM 32k	PM 128k	REALTALK
Closed-book	none / 0.4	none / 0.3	none / 0.3	none / 0.3
Chunk retrieval	none / 1.4	none / 1.1	none / 1.3	none / 1.3
Verbatim dump	0 / 2.4	0 / 4.0	0 / 3.8	0 / 3.2
Foldered sessions	0.33 / 2.4	0.08 / 2.9	0.15 / 3.5	0.13 / 3.6
Reorg. (preserve)	1.82 / 2.4	2.63 / 1.4	3.48 / 1.6	0.70 / 3.4
Reorg. (condense)	2.04 / 2.1	1.10 / 1.4	2.97 / 1.6	1.89 / 3.2
Agent-curated memory	12.93 / 2.4	2.60 / 2.5	13.44 / 2.6	7.55 / 3.1

the most consistent leader: it ties or tops LoCoMo (86.1), REALTALK (77.6), and PersonaMem 128k (76.2), whereas the Agent-curated memory leads only LoCoMo and is the weakest memory of all on PersonaMem 32k (37.5, against the Verbatim dump's 78.1; the representation-level analysis of this failure is in Section E.2.8). On that 32k tier the Verbatim dump leads outright, and even Chunk retrieval (71.9) beats every reorganized or curated memory there, while on the conversational benchmarks Chunk retrieval trails every variant except the condensing reorganizer: one more sign that no single form dominates.[3]

Where organization shows an unambiguous gain is search cost (Table A10), and the gain tracks the scale of the memorized content. On the two PersonaMem tiers, whose content is the largest and densest (137 and 612 kilobytes when dumped verbatim, against 102 and 105 for the conversational benchmarks), the reorganized memories cut the Verbatim dump's per-query search price by half or more and the curated memory by a third (PersonaMem 32k: 1.4 cents for either reorganizer and 2.5 for the curated memory against 4.0 for the Verbatim dump; 128k: 1.6 and 2.6 against 3.8). On the smaller conversational benchmarks the memories and the Verbatim dump search at near parity (2.1 to 2.4 cents on LoCoMo).

The mechanism, visible in Table A17, is an effort trade: on the PersonaMem tiers the reorganized and curated memories spend more search rounds and tool calls than the Verbatim dump (seven to nine calls against its four to five), because the search agent navigates structure rather than scanning, yet each targeted read pulls far fewer tokens, so the total falls wherever the Verbatim dump is large. Structure brings search economy, most on the content that needs it.

The two reorganizer versions pull apart by benchmark rather than uniformly. Preserving every fact helps on the conversational streams and the long PersonaMem 128k (LoCoMo 82.9 against the condensing version's 79.1, REALTALK 77.6 against 41.2, 128k 59.5 against 54.8), where dropping detail costs answerable content, most starkly on REALTALK, whose correctness nearly halves under condensation. Condensing instead helps on PersonaMem 32k (68.8 against 56.2), where a tighter memory is easier to search. That one reorganizer flips sign with the benchmark, halving REALTALK correctness when it condenses, is the clearest sign that **no single compression policy is right across benchmarks**.

E.2.2 Which memory serves which execution agent: the full reading

The reading below carries the full detail of the embodied half of Section 4.2.1; the complete three-chain tables follow.

The embodied setting asks the same question with a harder probe, and which memory does best depends on who consumes it. The embodied benchmarks are run in two ways: *three-chain runs*, three independent chains that together cover every goal family with the memory reset between them (Tables A11 and A12), and *single-chain runs*, all 140 tasks as one chain with the memory snap-

[3] Read the per-tier orderings with care: columns rest on 32 to 158 questions, and re-judging a fixed cell moved about two questions (± 1.3 points; Table A9), so gaps of a few questions on the small tiers are suggestive rather than settled.

Table A11: Embodied setting (Section 4.2), three-chain runs: task success and deployment cost under two execution agent tiers. **Succ**: fraction of the 140 tasks solved (%). **Deploy**: retrieval plus execution cost per task (cents; per-role rates and the split by role in Table A23; building the memory is separate and discussed in the text). The two tiers run the same tasks and the same memory-building procedure under a stronger and a weaker execution agent; compare columns within a tier, and across tiers read only the qualitative contrast.

	Execution agent `gpt-4.1`		**Execution agent `gpt-4.1-mini`**	
Memory	*Succ*	*Deploy*	*Succ*	*Deploy*
No memory	73.6	12.4	52.9	3.2
Episode log	**87.1**	13.4	66.4	9.0
Curated skills	80.7	13.1	65.7	4.7
Curated skills+mem	75.7	13.2	55.0	5.3
Curated skills+mem (GS)	82.1	12.4	**76.4**	4.3

Table A12: Embodied setting on ScienceWorld (Section 4.2), three-chain runs, management agent and search agent on `gpt-5.4-mini` as in Table A11: **Succ**, tasks fully solved (%, 140 tasks); **Score**, the environment's task score, which credits completed subgoals (mean over tasks, clipped at 0); **Cap**, episodes ended by the 60-action cap without success. Bold: the best value per column. Per-family success in Table A26.

	Execution agent `gpt-4.1`			**Execution agent `gpt-4.1-mini`**		
Memory	*Succ*	*Score*	*Cap*	*Succ*	*Score*	*Cap*
No memory	40.7	53.3	53	22.1	35.7	89
Episode log	**52.1**	56.3	35	30.0	49.4	80
Curated skills	46.4	**58.4**	40	**40.0**	**60.4**	64
Curated skills+mem	48.6	56.8	39	27.9	42.6	81
Curated skills+mem (GS)	48.6	58.3	47	**40.0**	56.7	61

shotted after every task (Table A21). Under the stronger execution agent the verbatim Episode log leads (87.1%), with Curated skills+mem (GS) second (82.1%); under the weaker execution agent the order inverts, Curated skills+mem (GS) leading by ten points (76.4 against 66.4; Table A11). Task-paired comparisons give the two sides of the inversion different statistical support: at `gpt-4.1` the Episode log's lead over GS is suggestive at best (net $+7$ of 140 tasks, two-sided sign test $p \approx 0.23$), while at `gpt-4.1-mini` GS beats both the Episode log (net $+14$, $p \approx 0.02$) and the Curated skills memory (net $+15$, $p \approx 0.01$).

The slopes tell the mechanism: dropping the execution agent tier costs the Episode log 21 points but GS only 6, and the execution agent's invalid-action rate, measured from the run records, rises under the Episode log from 19 to 36 percent while GS rises only from 12 to 22, so **raw episode transcripts demand an execution agent strong enough to digest them, whereas guidance written for the task degrades gracefully**.

Two corollaries follow. Memory is worth more to the weaker execution agent (best lift over No memory: $+23.6$ points against the stronger tier's $+13.6$), exactly where deployment is also least expensive. And the protective strictness of Curated skills+mem is tolerable only under a strong execution agent: already a regression at `gpt-4.1` (75.7 against 80.7 for Curated skills), it collapses to two points above the no-memory floor at mini, while Curated skills, which serves files liberally, lands with the Episode log there; GS retains the largest gain over no memory at the weaker tier.

Curated skills+mem (GS) is also the least expensive memory to deploy. Deployment cost per task (Table A11) puts Curated skills+mem (GS) below every other memory on both tiers, because its compact memory keeps retrieval cheap and its guidance shortens episodes. Running with No memory still costs less per task at the mini tier (3.2 against 4.3 cents), but counted per solved task GS undercuts it (5.7 against 6.1), so **the memory costs less per solved task than no memory**. Building it is the extra cost (about 10 to 11 dollars per 140 tasks at each tier, against zero for

Table A13: The management agent backbone-capability study on one PersonaMem 128k conversation: varying only the management agent under the fixed gpt-5.4-mini search agent. *Corr* = answer correctness (%), *Attr* = attribution (%), *Cost* = search cents per query at the Table A17 rates, *TC* = total build tool calls; the memory profile follows: directories, files, markdown sections (headings of any level), size in kilobytes, and *X-ref* = in-memory `see /memories/...` cross-references.

Management agent	*Corr*	*Attr*	*Cost (¢)*	*TC*	*Dirs*	*Files*	*Sec.*	*KB*	*X-ref*
gpt-5.4-nano	73.8	96.0	3.2	2720	12	122	413	286	57
gpt-5.4-mini	66.7	93.7	2.6	1986	1	2	210	151	7
gpt-5.4	71.4	92.5	3.8	3921	4	105	561	340	233

Table A14: The search agent grid: three search agent backbones read the same three fixed memories of Table A13 (the gpt-5.4-mini row restates that table's cells). Per memory, *Corr* = answer correctness (%), *¢* = search cents per query priced at each search agent's own rates (per million uncached-input / cached-input / output tokens: gpt-5.4-nano \$0.20 / \$0.02 / \$1.25, gpt-5.4-mini \$0.75 / \$0.075 / \$4.50, gpt-5.4 \$2.50 / \$0.25 / \$15.00), *Calls* = mean tool calls per query.

	nano-built memory			**mini-built memory**			**gpt-5.4-built memory**		
Search agent	*Corr*	*¢*	*Calls*	*Corr*	*¢*	*Calls*	*Corr*	*¢*	*Calls*
gpt-5.4-nano	64.3	0.9	5.6	59.5	0.7	5.7	61.9	0.8	6.5
gpt-5.4-mini	73.8	3.2	7.3	66.7	2.6	8.2	71.4	3.8	8.0
gpt-5.4	**83.3**	18.5	9.9	81.0	10.0	7.6	71.4	15.3	10.0

the Episode log), the same curation-versus-search trade the conversational setting prices. Where the conversational setting's answer was that organization's value is conditional on the memorized content, the embodied setting's is that it is conditional on the consumer; **both halves of RQ2 refuse an unconditional answer**.

ScienceWorld repeats the two-tier pattern. On ScienceWorld (Table A12; families in Table A26) the stronger execution agent again does best with the Episode log (52.1 against 46.4, 48.6 and 48.6 for the three curated memories) and the weaker one with Curated skills and Curated skills+mem (GS) (40.0 against 30.0), with No memory floors of 40.7 and 22.1. Curated skills+mem, the weakest curated memory on ALFWorld, lands at 48.6 and 27.9 here: level with Curated skills+mem (GS) under `gpt-4.1` and below the Episode log at the mini tier. Curated skills and Curated skills+mem (GS) raise the environment's task score, which credits the subgoals an episode completes, above the Episode log's at both tiers even where their success is lower, and the curated memories change which task families get solved: under `gpt-4.1`, Curated skills and Curated skills+mem are the only memories that solve grow-a-plant often (65 and 50 against at most 35) and Curated skills+mem (GS) the only one above half on the unknown-substance conductivity family. The inversion is therefore a property of the pairing of management agent and execution agent, not of ALFWorld; with the other tested families in both memory roles, it disappears on both benchmarks (Section F).

The trajectory-origin study, which separates the effect of the execution agent that consumes a memory from that of the execution agent whose trajectories built it, is reported in Section E.2.7.

E.2.3 Backbone capability of the two agents, conversational setting: the full reading

The reading below carries the full detail of the conversational half of Section 4.2.2; the complete backbone-capability tables follow. One figure is corrected against Table A13: the nano-built PersonaMem 128k memory has 122 files.

Does memory management improve with the management agent's backbone capability? Each backbone-capability study changes one agent's backbone over three tiers, `gpt-5.4-nano`,

`gpt-5.4-mini` and `gpt-5.4`, and holds everything else fixed. We compare three management agent backbones on the PersonaMem 128k memory, adding gpt-5.4-nano and gpt-5.4 rebuilds beside the main gpt-5.4-mini build while holding the stream, the instruction, and the gpt-5.4-mini search agent fixed (Table A13). **The management agent's strength expressed itself almost entirely as organizational style and almost not at all as answer quality.**

Structurally the three memories share nothing: nano fragments into 122 files across twelve folders, mini consolidates into two files that nest 210 heading sections, and gpt-5.4 shards into 105 files nested seven levels deep and stitched by 233 cross-references, thirty times mini's linking. Yet correctness sits in a seven-point band that does not even run in one direction with model strength (73.8 for nano, 66.7 for mini, 71.4 for gpt-5.4), a spread of three questions out of 42, comparable to the one-to-two questions that re-runs move on this same conversation (Section 4.1.3). The fixed search agent, in effect, absorbed large differences in memory organization into a three-question spread in answers.

Build effort follows the same pattern, not in one direction with capability: the strongest model works the hardest (3,921 tool calls against mini's 1,986) and builds the memory that is the most expensive to search (3.8 against 2.6 cents), without out-answering the weakest. On this one conversation, then, a stronger management agent produces a more elaborate memory, not a more useful one, once the search agent is held fixed and the instruction says what structure is for.

The search agent's side of the same probe answers what the management agent backbone-capability study left open: varied over the same three fixed memories (Table A14), the search agent's effect runs in one direction where the management agent's was flat, mean correctness rising from about 62 under the nano search agent through 71 under mini to 79 under gpt-5.4. **On this benchmark backbone strength counts in the search agent, not in the management agent.** And the stronger search agent does not rescue the elaborate memory: gpt-5.4 reads the nano-built 122-file memory best (83.3) and the ornate gpt-5.4-built memory worst (71.4), so the 105-file, 233-pointer construction stays unrewarded under every search agent we tried. Under the weak search agent the three memories' scores compress into a low band (59.5 to 64.3), differences flattening exactly where capability is scarcest.

The grid's cost columns locate the price of that capability. The search agent's strength, not memory organization, is the dominant cost axis: a query costs under a cent at nano, two to four cents at mini, and ten to nineteen at gpt-5.4, while within any one search agent the memory moves the cost by at most a factor of two. That factor is still informative: **the consolidated two-file memory nearly halves the strong search agent's per-query price against the 122-file memory (10.0 against 18.5 cents) at a one-question difference in correctness (81.0 against 83.3)**, and it draws the fewest tool calls from that search agent (7.6 per query against 9.9). For the strongest search agent too, organization shows in the same quantity, search cost, that Section 4.2 found across the conversational benchmarks' variants.

All of this remains one 42-question conversation, so gaps of a few points are suggestive; but the asymmetry, **the management agent's capability decoupled from quality, the search agent's strongly coupled**, is the cleanest single fact this study has produced about the conversational setting. The asymmetry also has a measured boundary: on the 32k conversations where curation collapses through build-time defects, the same management agent upgrade does move the fixed search agent, from 12 of 32 questions to 18 of 32 (Section E.2.8). The benefit of a stronger management agent therefore differed between the two evaluations: it was larger on the 32k cases with observed build-time defects.

E.2.4 Backbone capability, embodied setting: the full reading

The reading below carries the full detail of the embodied half of Section 4.2.2; the complete single-chain table follows.

The same probe on the skill management agent finds a step at the top tier, and it shows in the memory. Varying only the management agent's backbone under a fixed execution agent and search agent (Table A15; per family in Table A21) leaves the bottom of the backbone-capability study flat, `gpt-5.4-nano` not detectably different from `gpt-5.4-mini` (net -2, $p \approx 0.85$) at a seventh of the management cost ($1.59 against $10.82 for the management agent role), while

Table A15: Single-chain runs on ALFWorld: success over 140 tasks (%), final memory size and full cell cost (management plus deployment, dollars) by memory variant, harness and backbone, all with Curated skills+mem (GS) unless stated; the Center row and the two `gpt-5.4-mini` backbone rows are the same cell as Curated skills+mem (GS) at the mini tier. Per-family success: Table A21.

	Succ	*Files*	*KB*	*$*
Execution agent `gpt-4.1`				
No memory	73.6	0	.	.
Episode log	**88.6**	140	220	18.9
Curated skills+mem (GS)	84.3	137	120	23.8
Execution agent `gpt-4.1-mini`				
No memory	52.9	0	.	.
Episode log	73.6	140	270	12.2
Curated skills+mem (GS)	76.4	45	70	17.5
Harness (execution agent `gpt-4.1-mini`)				
Center	76.4	45	70	17.5
Center+BM25	75.0	76	92	15.0
Shell	**82.9**	**36**	78	14.9
Management agent backbone (execution agent `gpt-4.1-mini`)				
`gpt-5.4-nano`-built	75.0	114	203	10.3
`gpt-5.4-mini`-built	76.4	45	70	17.5
`gpt-5.4`-built	**89.3**	**16**	101	34.6
Search agent backbone (execution agent `gpt-4.1-mini`)				
`gpt-5.4-nano` search	73.6	50	67	14.4
`gpt-5.4-mini` search	76.4	45	70	17.5
`gpt-5.4` search	80.0	62	65	22.5

the top of the backbone-capability study jumps: `gpt-5.4` adds 13 points (net $+18$, $p \approx 0.001$), concentrated in the two families that need genuinely transferred procedures (cool $+9$, heat $+5$; the easy families are saturated at every management agent model).

The memory explains it. Memory size runs inversely with capability (Figure A7): nano writes 114 small files and links them, mini consolidates to 45, and gpt-5.4 distills 16 dense, self-contained entries, with few additional files after task 35 and continued editing and expansion thereafter. The file-format audits and the outcome-gate audit measure different properties (Section B.5); some positive skills first appeared after failed attempts. The libraries also differed in procedural detail: nano dutifully filed twelve heating-related entries yet scored 6 percent on heat, while gpt-5.4's two dense heat entries carried it to 75. Notably, the gpt-5.4 management agent's chain, run entirely at the mini execution agent, reaches 89.3 against 84.3 for the `gpt-5.4-mini`-curated GS chain at the `gpt-4.1` execution agent, and one task above that tier's raw-log result of 88.6, a comparison that changes both the management agent and the execution agent.

Varying the search agent's backbone instead gives smaller successive gains, within noise at each step. With the management agent held at `gpt-5.4-mini` and only the search agent changed (Table A15, last block), success runs 73.6, 76.4 and 80.0 for `gpt-5.4-nano`, `gpt-5.4-mini` and `gpt-5.4`. Neither neighboring step is significant under the paired sign test (net -4, $p \approx 0.56$; net $+5$, $p \approx 0.46$) and the end-to-end gap is net $+9$ ($p \approx 0.09$), a modest slope against the management agent's step at the top tier. The gains sit in the two families that need transferred procedures (cool $+24$ and heat $+13$ points for `gpt-5.4` over `gpt-5.4-mini`), while the light family falls from 12 to 6 of 13 tasks under both the weaker and the stronger search agent; the memories end at 50, 45 and 62 files, the difference the changed trajectories make, since the management agent is the same. The search agent role cost $0.62, $3.66 and $9.81.

Together the two backbone-capability studies give a contrast. Where writing means recording and arranging content that already exists, as in conversation, the management agent's strength changes style, not quality; a sufficiently competent search agent absorbs any arrangement by spending effort, and that compensation is itself capability-priced, the grid of Table A14 showing the search agent's

Table A16: Harness axis (Section 4.2.3): answer quality under three harnesses, holding the memory variant (Agent-curated memory) and backbone fixed. **Corr**: answer correctness (judged, %; on PersonaMem judged option matching). **Attr**: citation support (judged, %). The three harnesses are defined in Section 3; Center is the agent-curated run of Table A9, reused. Center+BM25 and Shell report clean re-runs after two prompt-wording fixes (Section E.1); the per-benchmark orderings sit within the single-run variation measured there.

	LoCoMo		**PersonaMem 128k**	
Harness	*Corr*	*Attr*	*Corr*	*Attr*
Center (default tools)	86.1	94.4	66.7	93.7
Center+BM25	86.7	94.9	64.3	91.3
Shell (bash)	86.1	93.1	61.9	92.1

strength worth points at every tier while the management agent's is not. Where writing must transform noisy trajectories into transferable procedures, capability couples sharply, as a step at the top tier: **every management agent model meets the format; only gpt-5.4 of the backbone-capability study distills procedures that transfer** (the two replicated families do as well, Section F).

Notably, in neither setting does organization alone explain value: the conversational study varies organization wildly at flat quality, and the embodied study's bottom two models do not differ detectably across a 114-against-45-file gap, gpt-5.4 leading on what its entries say, not how many there are.

E.2.5 The harness and outcomes: the full reading

The reading below carries the full detail of Section 4.2.3, conversational then embodied; the complete harness quality table follows.

Radically different organizations answer about equally well. These memories diverge in organization but stay close in quality (Table A16). On LoCoMo the three harnesses sit within one question of each other (86.1 to 86.7% correctness) despite the gap in file count. On PersonaMem 128k correctness spans 61.9 to 66.7% with the consolidated memories ahead of the 147-file shard, but the gap is a question or two out of 42 and moved by that much between our original runs and the clean re-runs, so we read the per-benchmark orderings as within single-run variation rather than as a fragmentation penalty. The broad convergence is still not a null result: on these benchmarks several organizations serve retrieval comparably, and the search agent adapts to whatever structure it is handed. It does, though, caution against the intuition that finer structure is straightforwardly better.

The search agent co-adapts to the memory it is given. How retrieval proceeds shifts with the layout. BM25 keyword search, the tool Center+BM25 adds, is used routinely on a many-file memory (54 calls across the LoCoMo queries, where ranked lookup narrows 29 files) but almost never on a single mega-file (twice on PersonaMem 128k), where the search agent instead walks the table of contents and reads sections. **The same agent draws a different strategy from the same tools depending on how memory is laid out**, an unprompted coupling between organization and retrieval.

In the embodied setting, the same axis splits: adding tools changes behavior, replacing the harness changes outcomes. Repeating the comparison on the embodied setting (Table A15, harness block), granting both roles ranked whole-file search on top of the default tools, and the management agent two section-level write tools, changes behavior without changing outcome: the search agent adopts the new tool (219 ranked-search calls across the chain), the management agent's memory lands at 76 files instead of 45, and the result shows no detected difference from Center (net -2, $p \approx 0.85$).

Replacing the harness with a shell moves the outcome: **the shell chain is the best chain the mini management agent produced under any harness** (82.9%), ahead of Center+BM25 decisively

(net $+11$, $p \approx 0.04$) and of Center suggestively (net $+9$, $p \approx 0.06$), with the broadest family lift (heat reaches 81%, above even the gpt-5.4 management agent). The higher success accompanies a different memory: the same mini management agent, working through bash, produced the most consolidated memory of any mini-curated chain at the file level (36 files against the default's 45) while keeping its internal structure (107 sections against the default's 99; Table A6), and its schema held without any tool-side validation.

Across both settings. Adding tools changed behavior but not outcomes, and replacing the harness reshaped the memory; what flips between settings is the direction and the gain. On long dialogue Shell shards memory without a detected change in quality; in the embodied setting it consolidates memory and leads. Both directions are `gpt-5.4-mini`'s: the two replicated families of Section F consolidate PersonaMem 128k under the shell as well, and their two harnesses do not differ detectably in the embodied setting.

The harness is therefore not a neutral wrapper but a lever whose effect is mediated by what the setting rewards: fine shards for browsing long dialogue, dense procedures for guiding execution, for `gpt-5.4-mini`; which direction a family takes is its own (Section F). On the conversational side it is also a lever our quality benchmarks are largely blind to. That sharpens the paper's central question rather than settling it: if a memory's organization can vary this much with so little effect on conversational answer quality, *when* does organization matter: at a scale beyond one conversation, over the long horizons a persistent memory is meant to serve, or for sustainability, keeping a growing memory navigable instead of letting it spread? And if the harness controls organization this directly, it becomes a natural knob for *inducing* a target structure rather than merely observing one.

E.2.6 Search cost and effort per query

Table A17 reports search cost and effort per query for every variant and benchmark by token category, with tool rounds and calls; the search-economy reading of Section 4.2.1 and panel (b) of Table A10 are drawn from it.

E.2.7 The trajectory-origin study

The cells of Table A18 (Section 4.2) reuse the paper's full chains: the memory after task 70 of each chain (the curated memory reconstructed from the curation record and checked against the run records file by file, the episode log as the first 70 episodes) is loaded as a fixed library, the management agent is disabled, and tasks 71 to 140 of the same order run in five parallel groups with the search agent of record; every run record's library snapshot equals the frozen memory, and no management agent call is made. Everything else equals the full chains of record. Numbers are successes over the 70 tasks; the task-paired sign tests behind Section 4.2 compare cells task by task; Table A19 gives the same cells' steps, retrieval tokens and execution agent cost. The ScienceWorld block applies the same freeze to the ScienceWorld full chains, with one substitution: the `gpt-4.1-mini` chain's curation record could not reproduce its task-70 memory (the management agent's structured edits depend on the exact memory state, and a cache-driven replay diverged from the run records at the first task), so that memory comes from a second run of the same chain over tasks 1 to 70 under the settings of record (33 files, reconstructed and checked file by file as above). Its own-memory reference in Table A18 remains the original chain, so that one frozen-versus-growing contrast ($+21$ tasks) also carries run-to-run variance between two builds; the other three pairings compare a chain with its own memory. Task-paired sign tests on ScienceWorld: consumer contrasts $+15$, $+11$, $+10$ and -1 (p=0.001, 0.02, 0.04, 1) on the `gpt-4.1`-built log, the `gpt-4.1`-built curated memory, the `gpt-4.1-mini`-built log and the `gpt-4.1-mini`-built curated memory; management agent contrasts $+11$ (p=0.02) for the strong consumer on logs and -10 (p=0.06) for the weak consumer on curated memories, the other two within 6 tasks ($p \geq 0.2$); curated over log by $+4$, $+8$, $+13$ and $+24$ (p=0.56, 0.20, 0.02, < 0.001); every frozen memory beats no memory, seven of the eight at $p \leq 0.06$.

Frozen memories separate consumer effects from trajectory-source effects. The comparison above changes two things at once: the execution agent that consumes the memory and the execution agent whose episodes the management agent integrated, so each tier read a different memory. To separate the two we froze each full chain's memory after task 70 and ran both execution agents

Table A17: Search cost and effort per query, by token category so any pricing can be applied. *Un* = uncached input, *Ca* = cached input, *Out* = output, in thousands of tokens (*Rsn%* = share of output that is reasoning); *Rd* / *TC* = mean tool rounds and tool calls; *Cost* = cents per query at the deployed rates ($0.75 / $0.075 / $4.50 per million uncached-input / cached-input / output tokens).

Memory	*Un*	*Ca*	*Out*	*Rsn%*	*Rd*	*TC*	*Cost (¢)*
			LoCoMo				
No memory							
Closed-book	0.2	0.0	0.9	97	1.0	0.0	0.4
Chunk retrieval	13.8	5.3	0.6	74	3.6	3.0	1.4
Files							
Verbatim dump	23.8	10.8	1.1	71	4.7	6.0	2.4
Foldered sessions	22.9	12.4	1.2	71	5.0	6.4	2.4
Reorg. (preserve)	23.4	15.5	1.3	63	5.9	7.3	2.4
Reorg. (condense)	19.2	17.0	1.2	64	6.4	7.6	2.1
Agent-curated memory	23.0	19.3	1.3	62	5.6	7.0	2.4
			PersonaMem 32k				
No memory							
Closed-book	0.5	0.0	0.5	98	1.0	0.0	0.3
Chunk retrieval	8.8	2.2	1.0	80	2.6	2.9	1.1
Files							
Verbatim dump	41.7	6.3	1.8	82	3.9	4.7	4.0
Foldered sessions	26.4	9.3	2.0	82	3.8	4.7	2.9
Reorg. (preserve)	9.4	10.4	1.4	73	4.8	7.0	1.4
Reorg. (condense)	9.0	8.4	1.5	74	4.6	7.1	1.4
Agent-curated memory	19.8	10.8	2.0	77	5.4	8.9	2.5
			PersonaMem 128k				
No memory							
Closed-book	0.4	0.0	0.6	99	1.0	0.0	0.3
Chunk retrieval	10.9	1.9	1.0	79	2.6	3.5	1.3
Files							
Verbatim dump	39.8	6.1	1.8	84	3.9	4.4	3.8
Foldered sessions	35.8	6.7	1.6	80	3.9	4.9	3.5
Reorg. (preserve)	9.8	12.3	1.8	77	4.9	7.4	1.6
Reorg. (condense)	10.6	12.1	1.7	75	5.2	7.9	1.6
Agent-curated memory	23.1	14.6	1.7	74	5.3	8.2	2.6
			REALTALK				
No memory							
Closed-book	0.2	0.0	0.6	95	1.0	0.0	0.3
Chunk retrieval	11.3	3.1	0.9	74	3.3	3.4	1.3
Files							
Verbatim dump	29.6	10.9	1.9	79	5.3	6.6	3.2
Foldered sessions	34.4	12.3	2.1	77	5.6	7.3	3.6
Reorg. (preserve)	32.3	16.0	2.0	76	5.6	6.8	3.4
Reorg. (condense)	25.8	23.0	2.5	76	7.5	10.5	3.2
Agent-curated memory	27.0	21.6	2.1	74	5.6	8.8	3.1

on the same 70 held-out tasks over the same fixed memory, with the search agent of record active and no curation, crossing the memories built from `gpt-4.1`'s trajectories with those built from `gpt-4.1-mini`'s (Table A18, Section E.2.7). On ALFWorld, whose trajectories built the memory makes no detectable difference for either consumer: the four management agent contrasts move 1 to 3 of the 70 tasks (task-paired sign tests, $p \geq 0.6$). The consumer does. On the episode logs the strong execution agent solves 4 to 5 more tasks than the weak one (p=0.13 and 0.06) while on the curated memories they differ by at most two tasks. The strong execution agent does better with the log than with the curated memory whichever tier built them (by 3 and 6 tasks), whereas the weak execution agent's preference for curated skills is small on this half of the chain (+3 with the `gpt-4.1`-built memories, −1 with its own): the weak execution agent's advantage from curated skills shrinks as the Episode log accumulates, 10 points in the three-chain runs, 3 over one 140-task chain, and 3 or fewer on the held-out half at a fixed memory. Freezing the memory costs 0 to 4 tasks against the chains that

Table A18: Frozen-memory cross-evaluation (Section 4.2): success (%) over tasks 71 to 140 of the full-chain order with the memory frozen after task 70, for memories built from each execution agent's trajectories (rows) and each execution agent as the consumer (columns); the search agent of record stays active and no curation runs, so every cell reads the same fixed memory on the same 70 tasks. Reference rows: the paper's own chains, whose memory kept growing, on the same 70 tasks, and no memory. n=70 per cell. The ScienceWorld memory built from `gpt-4.1-mini`'s trajectories comes from a second run of that chain over tasks 1 to 70 (Section E.2.7).

Memory	**Built from the trajectories of**	**Consumer**	
		`gpt-4.1`	`gpt-4.1-mini`
ALFWorld, frozen after task 70			
Curated skills+mem (GS)	`gpt-4.1`, 59 files	84.3	87.1
Curated skills+mem (GS)	`gpt-4.1-mini`, 30 files	82.9	82.9
Episode log	`gpt-4.1`, 70 episodes	88.6	82.9
Episode log	`gpt-4.1-mini`, 70 episodes	91.4	84.3
ALFWorld, references on the same tasks			
Curated skills+mem (GS)	own, memory growing	88.6	87.1
Episode log	own, memory growing	94.3	84.3
No memory	–	74.3	61.4
ScienceWorld, frozen after task 70			
Curated skills+mem (GS)	`gpt-4.1`, 28 files	67.1	51.4
Curated skills+mem (GS)	`gpt-4.1-mini`, 33 files	64.3	65.7
Episode log	`gpt-4.1`, 70 episodes	61.4	40.0
Episode log	`gpt-4.1-mini`, 70 episodes	45.7	31.4
ScienceWorld, references on the same tasks			
Curated skills+mem (GS)	own, memory growing	74.3	35.7
Episode log	own, memory growing	51.4	27.1
No memory	– (three-chain cells, same tasks)	41.4	20.0

kept curating, and every frozen memory beats no memory ($p \leq 0.11$). On ScienceWorld, curated memory helps both consumers, while consumer and trajectory-source effects vary by memory: the strong execution agent solves 10 to 15 more of the 70 tasks than the weak one on three of the four memories ($p \leq 0.04$), both consumers do better with the curated memory whoever built it (by 4 to 24 tasks), and the management agent matters only where it plausibly should, the strong consumer gaining from the strong tier's own logs ($+11$, p=0.02) and the weak consumer from a curated memory distilled from its own trajectories ($+10$, p=0.06). The frozen results retain the benchmark difference: the strong execution agent does better with logs on ALFWorld, whereas both consumers do better with curated memory on ScienceWorld. In ScienceWorld, however, the trajectory source also affects some comparisons. Efficiency moves with success (Table A19): on ScienceWorld, the log incurs 2.6 to 4.7 times the curated memory's execution agent cost per task, and on ALFWorld the strong execution agent's logs take fewer steps but twice the retrieval tokens, while the weak execution agent pays less for execution with curated memory, with success differences of $+3$ and -1 tasks for the two trajectory sources.

E.2.8 Why the agent-curated memory fails on PersonaMem 32k

The notable entry in Table A9 is that agent curation, the most effortful memory to build on every benchmark (highest curation tool calls; Table A5), is the weakest memory on PersonaMem 32k (37.5% correctness, against 78.1 for the verbatim dump on identical questions). We traced every wrong answer across the three conversations to understand it.

The inspected errors point to problems in temporal representation and retrieval, even when the gold-critical facts are retained. The cells are mechanically clean (no API errors, no failed episodes, no truncation; searches use at most 12 of their 40 tool rounds), and the search agent's grounding and attribution scores stay near-perfect (0.94 to 0.98 and 0.97 to 1.0 on the judge's 0-to-1 scales) even as correctness collapses: the search agent finds and cites the right sections, then selects the wrong option. In the thirteen questions the Verbatim dump answers and curation misses, the gold-critical

Table A19: Frozen-memory cross-evaluation, efficiency counterpart of Table A18 over the same 70 held-out tasks: *Steps* = environment steps per task; *Retr. in* = search agent input tokens per task (thousands, at the `gpt-5.4-mini` rates of Table A23); *¢/task* and *¢/succ* = execution agent cost per task and per solved task at its tier's rates. Rows and columns as in Table A18.

Memory	Built from	Consumer `gpt-4.1`				Consumer `gpt-4.1-mini`			
		Steps	*Retr. in*	*¢/task*	*¢/succ*	*Steps*	*Retr. in*	*¢/task*	*¢/succ*
ALFWorld, frozen after task 70									
Curated skills+mem (GS)	`gpt-4.1`	19.6	51k	8.0	9.5	20.7	52k	1.7	2.0
Curated skills+mem (GS)	`gpt-4.1-mini`	21.1	40k	9.1	11.0	21.1	47k	1.8	2.2
Episode log	`gpt-4.1`	14.5	108k	7.5	8.4	18.6	103k	2.0	2.4
Episode log	`gpt-4.1-mini`	14.4	112k	7.4	8.1	18.3	100k	2.0	2.3
ScienceWorld, frozen after task 70									
Curated skills+mem (GS)	`gpt-4.1`	21.4	64k	7.1	10.5	33.8	63k	2.7	5.3
Curated skills+mem (GS)	`gpt-4.1-mini`	19.3	82k	6.7	10.4	26.3	79k	2.0	3.0
Episode log	`gpt-4.1`	25.0	162k	26.2	42.6	34.3	176k	7.1	17.7
Episode log	`gpt-4.1-mini`	28.3	209k	31.6	69.1	37.5	201k	9.2	29.2

fact is present in the curated memory in every case. What curation removed was not facts but three properties of the record that PersonaMem rewards. Its questions ask for the persona's *latest* state of a preference that *changed* over the conversation, with distractors drawn from the earlier state or the static profile, and curation flattens the very signals that separate them. First, it leaves superseded preferences standing as present-tense traits beside their updates (one file asserts the persona "prefers sharing music face-to-face rather than in online forums" while a later section records the same persona joining and enjoying a forum). Second, it rewrites first-person affect into neutral feature lists (an emphatic "this app has become a game-changer" becomes a bulleted capability, so a question about whether the persona appreciated the app reads as an overclaim). Third, it scatters one narrative arc across many sections and files, so the decisive updating fact is often not read alongside the rest. The damage is graded: Verbatim dump 78.1 $>$ Foldered sessions 62.5 $>$ Agent-curated memory 37.5 on the same questions, worsening with how far the memory's form departs from the chronological verbatim record.

We read this as a limitation of the backbone model rather than of the curated representation itself. The management agent's instruction already prescribes timestamped temporal reconciliation, recording an update as the current value with the superseded one preserved as a dated past entry (Section C.1), so the representation *can* carry the latest-versus-previous distinction. The management agent applied that instruction inconsistently, reconciling some updates while leaving others as live traits, and the search agent did not always retrieve every dated entry and reason over the conflict; a backbone that executed the existing instruction reliably at build time and reasoned over co-located dated facts at read time would resolve these cases. We flag this as a hypothesis rather than a demonstration: the management agent-strength backbone-capability study (Section 4.2.2) varies the management agent while holding the search agent fixed and finds flat quality there, so the two sides need separating on these conversations. The direct build-side test covers all three conversations: each rebuilt with the management agent raised to `gpt-5.4`, everything else fixed, including the search agent and the judge. The fixed search agent recovers 18 of 32 questions over the `gpt-5.4` builds against 12 of 32 over the `gpt-5.4-mini` builds of record (56.3 against 37.5); the per-question pairing gains nine and loses three (two-sided sign test p=0.146), every conversation moves in the same direction (2/10 to 5/10, 5/10 to 6/10, 5/12 to 7/12), and the three rebuilds again share no organization (8, 21, and 7 files). This is directional support for the build-side half: management agent capability moves read outcomes on precisely the benchmark where curation collapses, while the same upgrade is flat on the 128k conversation whose memory already serves its search agent (Section 4.2.2). It is support with a remainder: 56.3 still sits far below the Verbatim dump's 78.1 on the same questions, but the remaining gap could reflect residual build defects, retrieval limits, or their interaction with the curated form. A search-side intervention remains outstanding.

Table A20: LoCoMo correctness with the four defective golds removed (Section E.2.9): the headline scores over all 158 questions against the same judge outcomes restricted to the 154 sound ones. Orderings are unchanged; every memory-backed variant rises slightly because each is judged correct on only two of the four.

Memory	*Corr, 158 Q*	*Corr, 154 Q (filtered)*
Closed-book	18.4	18.2
Chunk retrieval	81.6	82.5
Verbatim dump	84.2	85.1
Foldered sessions	86.1	87.0
Reorg. (preserve)	82.9	83.8
Reorg. (condense)	79.1	79.9
Agent-curated memory	86.1	87.0

E.2.9 LoCoMo gold-defect catalog and filtered scores

Before any experiment ran, we audited all 158 evaluated LoCoMo golds against the conversation transcript. Four are materially defective (2.5%); we evaluate on the standard data anyway for comparability and catalog them here. By evaluation order: question 20 (multi-hop) asks which places Calvin visited in Tokyo, and its gold lists a car museum that appears nowhere in the transcript (the cited turn mentions a Ferrari dealership, not placed in Tokyo). Question 64 (multi-hop) asks what style of guitars Calvin owns, and its gold invents a yellow guitar (the transcript's octopus guitar and shiny purple guitar are also plausibly the same instrument). Question 112 (single-hop) asks which Disney movie Dave mentioned as a favorite, but in the transcript Dave never names one and it is Calvin who names Ratatouille. Question 138 (single-hop) asks when Calvin first got interested in cars, citing a childhood car-show memory that belongs to Dave.

Table A20 restates correctness with these four questions removed, using the same per-question judge outcomes. Orderings do not change. Every variant except the closed-book baseline rises by roughly a point because each scores below its own average on the defective set (judged correct on only two of the four); on question 138 each memory-equipped variant answered faithfully from the record and was marked wrong, while closed-book, guessing an early age with no record at all, alone matched the gold. The headline table stays unfiltered, standard data being what other work reports.

E.2.10 Embodied setting: success by goal family

Table A21 gives the single-chain runs in full, success by goal family beside the totals of Table A15: the memory variants at each execution agent tier, the harness study and the two backbone-capability studies; the family readings of Section 4.2.2 (the heat and cool families under each backbone, the light family under the search agent study) rest on it.

Table A22 breaks the three-chain matrix of Table A11 down by ALFWorld goal family, recomputed from the archived per-task trajectories (the micro column reproduces Table A11 exactly). Simple placement is near ceiling for every variant at both tiers (94.3 to 100), so the matrix is decided in the harder families. At the `gpt-4.1` tier the Episode log leads through clean (85.2) and cool (84.0), while Curated skills+mem (GS) trades those for by far the best heat performance (87.5 where the log manages 37.5). At the `gpt-4.1-mini` tier the inversion of Section 4.2 is broad rather than concentrated: Curated skills+mem (GS) leads the Episode log in five of six families (two-object $+20.8$, cool $+20.0$, heat $+18.8$, examine $+15.4$, placement $+2.8$), the log keeping only clean ($+7.4$). Heat and cool stay the weak execution agent's hardest families, heat never exceeding 37.5 in the three-chain cells.

E.2.11 Embodied setting: deployment cost by role and absolute compute

Two questions the headline tables leave open are what the pipeline costs per role, and how much computation the system truly performs once provider caching is accounted for. Table A23 splits the three-chain cost of Table A11 into its three roles, management agent (build), search agent, and execution agent, each priced at its own backbone's rates; the dollar figure is the legitimate cross-

Table A21: Embodied setting, single-chain runs. Success (%) by goal family (family sizes as in Table A22); *All*: the 140-task micro average. *Files*, *KB*: final file count and size. *$*: full cell cost (build plus deployment). Upper blocks: memory variants at each execution agent tier. Third block: the harness axis, varying both roles' harness. Lower block: the management agent backbone-capability study, varying only the management agent's backbone. Last block: the search agent backbone-capability study, varying only the search agent's backbone with the management agent on `gpt-5.4-mini`. The harness block's Center row, the study's `gpt-5.4-mini` row and the last block's `gpt-5.4-mini` row are the same cell as Curated skills+mem (GS) at the `gpt-4.1-mini` tier above.

	Place	*Two*	*Look*	*Clean*	*Heat*	*Cool*	*All*	*Files*	*KB*	*$*
Execution agent `gpt-4.1`										
No memory	100.0	95.8	92.3	37.0	50.0	60.0	73.6	0	·	·
Episode log	100.0	100.0	84.6	92.6	25.0	**100.0**	**88.6**	140	220	18.9
Curated skills+mem (GS)	100.0	95.8	92.3	70.4	68.8	72.0	84.3	137	120	23.8
Execution agent `gpt-4.1-mini`										
No memory	97.1	75.0	46.2	33.3	12.5	20.0	52.9	0	·	·
Episode log	97.1	83.3	61.5	88.9	12.5	60.0	73.6	140	270	12.2
Curated skills+mem (GS)	97.1	87.5	92.3	77.8	43.8	48.0	76.4	45	70	17.5
Harness (Curated skills+mem (GS), execution agent `gpt-4.1-mini`)										
Center	97.1	87.5	92.3	77.8	43.8	48.0	76.4	45	70	17.5
Center+BM25	97.1	91.7	92.3	55.6	43.8	60.0	75.0	76	92	15.0
Shell	97.1	91.7	84.6	81.5	**81.2**	56.0	**82.9**	**36**	78	14.9
Management agent backbone (Curated skills+mem (GS), execution agent `gpt-4.1-mini`)										
`gpt-5.4-nano`-built	94.3	91.7	84.6	77.8	6.2	68.0	75.0	114	203	10.3
`gpt-5.4-mini`-built	97.1	87.5	92.3	77.8	43.8	48.0	76.4	45	70	17.5
`gpt-5.4`-built	100.0	87.5	**100.0**	85.2	75.0	84.0	**89.3**	**16**	101	34.6
Search agent backbone (Curated skills+mem (GS), execution agent `gpt-4.1-mini`)										
`gpt-5.4-nano` search	97.1	87.5	46.2	81.5	43.8	52.0	73.6	50	67	14.4
`gpt-5.4-mini` search	97.1	87.5	92.3	77.8	43.8	48.0	76.4	45	70	17.5
`gpt-5.4` search	97.1	91.7	46.2	85.2	56.2	72.0	80.0	62	65	22.5

Table A22: Embodied setting, three-chain runs: success (%) by ALFWorld goal family. *Place*: put an object at a location (n=35); *Two*: put two objects (n=24); *Look*: examine an object under a light (n=13); *Clean*, *Heat*, *Cool*: transform an object's state, then place it (n=27, 16, 25). *All*: the 140-task micro average of Table A11.

	Place	*Two*	*Look*	*Clean*	*Heat*	*Cool*	*All*
Execution agent `gpt-4.1`							
No memory	100.0	95.8	92.3	37.0	50.0	60.0	73.6
Episode log	100.0	100.0	100.0	85.2	37.5	84.0	87.1
Curated skills	97.1	95.8	84.6	59.3	62.5	76.0	80.7
Curated skills+mem	97.1	100.0	76.9	40.7	37.5	84.0	75.7
Curated skills+mem (GS)	100.0	91.7	84.6	59.3	87.5	68.0	82.1
Execution agent `gpt-4.1-mini`							
No memory	97.1	75.0	46.2	33.3	12.5	20.0	52.9
Episode log	94.3	75.0	69.2	74.1	6.2	48.0	66.4
Curated skills	100.0	83.3	69.2	63.0	37.5	20.0	65.7
Curated skills+mem	94.3	75.0	69.2	37.0	18.8	16.0	55.0
Curated skills+mem (GS)	97.1	95.8	84.6	66.7	25.0	68.0	76.4

role aggregate, since the roles run different backbones and token counts do not compare across backbones. Two regularities organize the deployment side. At the `gpt-4.1` tier the execution agent dominates deployment cost, and memory lowers its execution component: every memory variant's execution agent undercuts the no-memory execution agent's $17.35, the floor variant being

Table A23: Embodied setting, three-chain runs: build and deployment cost by role over the 140 tasks. *Build*: management agent spend per cell ($; zero for the Episode log, whose memory is written mechanically); *Search*, *Exec*: retrieval and execution spend; *Deploy* their sum; *¢/task* and *¢/succ*: deployment cents per task and per solved task; *Succ*: success (%), repeated from Table A11 for reading the ratios. Rates per million tokens (uncached input / cached input / output): execution agent `gpt-4.1` 2.00 / 0.50 / 8.00, `gpt-4.1-mini` 0.40 / 0.10 / 1.60; retrieval `gpt-5.4-mini` 0.75 / 0.075 / 4.50.

	Build $	*Search $*	*Exec $*	*Deploy $*	*¢/task*	*¢/succ*	*Succ*
Execution agent `gpt-4.1`							
No memory	—	—	17.35	17.35	12.4	16.9	73.6
Episode log	0	6.62	12.15	18.77	13.4	15.4	87.1
Curated skills	7.28	1.66	16.61	18.27	13.1	16.2	80.7
Curated skills+mem	8.12	2.82	15.71	18.53	13.2	17.5	75.7
Curated skills+mem (GS)	9.66	3.75	13.63	17.38	12.4	15.1	82.1
Execution agent `gpt-4.1-mini`							
No memory	—	—	4.51	4.51	3.2	6.1	52.9
Episode log	0	7.88	4.67	12.54	9.0	13.5	66.4
Curated skills	7.15	2.79	3.81	6.61	4.7	7.2	65.7
Curated skills+mem	7.81	3.10	4.34	7.45	5.3	9.7	55.0
Curated skills+mem (GS)	10.56	2.97	3.10	6.07	4.3	5.7	76.4

the tier's most expensive execution agent. At the `gpt-4.1-mini` tier the execution agent costs little, so retrieval efficiency decides: the Episode log's serve-everything retrieval costs $7.88 against Curated skills+mem (GS)'s $2.97, which combined with the success gap makes Curated skills+mem (GS) the least expensive variant per solved task at both tiers, and at the mini tier less expensive per solved task than running no memory at all (5.7 against 6.1 cents). Building is where curation bills instead: 7 to 11 dollars per cell against zero for the mechanically written Episode log memory, the same curation-versus-search trade the conversational setting prices.

Table A24 then mirrors, on the execution agent role, the per-query conventions of the conversational search-cost table (Table A17): rounds and token categories per task. What memory saves the execution agent is visible within each tier as the gap to the no-memory row. At `gpt-4.1` every memory variant shortens episodes (the Episode log most, 17.1 rounds against 25.9). At the mini tier Curated skills+mem (GS) has the fewest rounds, the lowest uncached and cached input, and the lowest execution agent cost (23.6 rounds against 32.0, uncached input 13.5k against 17.8k, cached input 135k against 209k per task), while the Episode log's verbatim context raises cached input and output above even the no-memory row, the serve-everything burden the deployment column prices.

Table A25 reports execution agent-side token-accounting bounds without applying token prices, from the per-turn token usage archived with every trajectory. Three readings. First, the naive no-cache sum overstates the intrinsic cost 16 to 23 times in these cells, which is why we never compare variants on raw prompt-token sums. Second, the appending episode design realizes 94.4 to 96.3 percent of the achievable caching savings in every cell, and measured compute stays within 1.7 to 2.05 times the ideal-reuse token bound. Third, the capability inversion survives translation into intrinsic compute: per task at the mini tier, Curated skills+mem (GS) beats the Episode log on every component (first-turn context 620 against 2,684 tokens, unique prefill 5,737 against 7,824, decode 1,999 against 3,028), so its token total is roughly 29 percent lower when prefill and decode tokens receive equal weight. The direction of the reduction survives any nonnegative prefill-versus-decode weighting (with at least one positive weight); at the `gpt-4.1` tier the same verbatim context instead makes the Episode log the memory variant with the lowest intrinsic cost (6,937 tokens per task against 7,818 for Curated skills+mem and 8,452 for Curated skills), the strong execution agent converting episode dumps into short episodes.

E.2.12 SCIENCEWORLD RESULTS

Table A12 and Table A26 give the fourteen `gpt-5.4-mini` cells (ten three-chain runs and four single-chain runs). In the three-chain runs, the ALFWorld two-tier pattern reproduces: with the

Table A24: Embodied setting, three-chain runs: execution agent-side efficiency per task. *Rounds*: environment steps taken. *Un*, *Ca*, *Out*: uncached-input, provider-cached-input, and output tokens (thousands per task). ¢: execution agent cost per task at the Table A23 rates. Comparisons are within one execution agent tier; what memory saves the execution agent shows as fewer rounds and fewer tokens than the no-memory row of the same tier.

	Rounds	*Un*	*Ca*	*Out*	¢
Execution agent `gpt-4.1`					
No memory	25.9	13.9	150.3	2.6	12.4
Episode log	17.1	10.0	102.1	2.0	8.7
Curated skills	24.0	13.0	146.8	2.4	11.9
Curated skills+mem	24.0	13.1	135.2	2.3	11.2
Curated skills+mem (GS)	22.1	12.4	116.7	1.8	9.7
Execution agent `gpt-4.1-mini`					
No memory	32.0	17.8	209.0	2.6	3.2
Episode log	26.0	16.3	219.5	3.0	3.3
Curated skills	26.9	15.6	178.9	2.0	2.7
Curated skills+mem	30.7	15.8	208.0	2.4	3.1
Curated skills+mem (GS)	23.6	13.5	135.3	2.0	2.2

Table A25: Embodied setting, three-chain runs: absolute execution agent-side computation in millions of tokens, provider- and price-independent. *Perfect*: the transcript's intrinsic lower bound (each token prefilled once, each output decoded once). *Measured*: billed uncached prefill plus decode. *No-cache*: the naive upper bound that re-prefills every turn's full prefix. *Eff*: share of the achievable caching savings realized, (no-cache − measured)/(no-cache − perfect). Comparisons are within one execution agent model only.

	Perfect	*Measured*	*No-cache*	*Meas/Perf*	*Eff*
Execution agent `gpt-4.1`					
No memory	1.16	2.32	23.36	1.99×	94.8%
Episode log	0.97	1.67	15.97	1.72×	95.3%
Curated skills	1.18	2.16	22.70	1.82×	95.5%
Curated skills+mem	1.09	2.15	21.08	1.97×	94.7%
Curated skills+mem (GS)	1.01	1.99	18.32	1.97×	94.4%
Execution agent `gpt-4.1-mini`					
No memory	1.40	2.86	32.12	2.04×	95.3%
Episode log	1.52	2.71	33.44	1.78×	96.3%
Curated skills	1.30	2.45	27.49	1.89×	95.6%
Curated skills+mem	1.39	2.56	31.67	1.84×	96.2%
Curated skills+mem (GS)	1.08	2.17	21.10	2.00×	94.6%

stronger execution agent the flat memory is the best memory (52.1 against 46.4, 48.6 and 48.6 for the curated memories), with the weaker one Curated skills and Curated skills+mem (GS) are (40.0 against 30.0; Curated skills+mem 27.9), and No memory sets floors of 40.7 and 22.1. Curated skills and Curated skills+mem (GS) raise the task score above the flat memory's at both tiers even where their success is lower (Curated skills+mem only under `gpt-4.1`, 56.8 against 56.3; 42.6 against 49.4 at the mini tier), and the curated memories push more episodes to the action cap (40, 39 and 47 against 35 under `gpt-4.1`). Skills lift success mostly by converting timeouts: at the mini tier the cap terminations fall from 89 with no memory to 80, 64, 81 and 61 while the terminal failures (a wrong focus or a wrong answer box, scored −100) barely move (20, 18, 20, 20 and 23), and under `gpt-4.1` the no-memory cell ends 30 episodes in a terminal failure, more than any mini cell; without a memory the mini execution agent falls into a focus trap on the melting-point family, 15 of its 20 tasks ending that way. The families split: under `gpt-4.1` the flat memory solves the find-a-living-thing and longest-lived-organism families fully and grow-a-plant never, the Curated skills and Curated skills+mem memories are the only ones that solve grow-a-plant often (65 and 50), and Curated skills+mem (GS) is the only memory above 50 on the unknown-substance

conductivity family (60). The full chains under the mini tier land below their three-chain twins (26.4 and 35.0 against 30.0 and 40.0) with more terminal failures (28 and 29 against 18 and 23), and their memories end at 140 episode files and 40 memory files (12 skills and 28 notes) (the three-chain Curated skills and Curated skills+mem (GS) memories at the mini tier hold 25 and 53 files summed over their three chains, and the Curated skills+mem memories 62 files under `gpt-4.1` and 49 at the mini tier, 49 and 43 of them memory notes); under `gpt-4.1` the Episode log full chain lands at 50.7 against 52.1 for its three-chain twin (42 cap terminations, 27 terminal failures) while the Curated skills+mem (GS) full chain reaches 66.4 against 48.6, the largest full-chain gain on either benchmark, with 24 cap terminations and a 32-file memory. Unclipped task scores run 13 to 25 points below the clipped ones. Table A27 prices the cells by role. The run records list per-task terminations (success, action cap, terminal failure) and the execution agent's and retriever's handled tool-call errors; no task errored.

E.3 Does a memory stay useful and organized as it grows? (RQ3)

Sustainability asks three things of a growing memory: does it stay useful, does it stay affordable per use, and does it stay healthy as later writing reworks earlier memory. The two settings answer with different instruments. On the conversational side the evidence is a scale contrast and a build record. From 32k to 128k the curated memory consolidates, relocating hierarchy inside files rather than sharding (Section 4.1), and organization's search economy grows with the scale of the memorized content (Section 4.2); but the two tiers differ in conversations, not in length alone, so we read the contrast as suggestive. The per-chunk build records of the LoCoMo and PersonaMem 128k builds add a checkpointed view of the memory itself (Figure A2), while answer quality along a growing dialogue remains unmeasured, since the questions come at the end. The embodied setting answers directly: its single-chain runs follow one 140-task chain with the memory accumulating end to end and snapshotted after every task, for the three memory variants at both execution agent tiers, the two other harnesses and the two other management agent backbones. Every full-chain run records, for each task, the outcome, the steps taken, per-role token usage and a memory snapshot; the per-task memories reconstructed from the curation records were validated against those snapshots (560 states, zero mismatches) before any content-level measure was taken from them. Success is read both absolutely and against the no-memory chain at the same position, because the fixed task order gets easier along the way and lifts even the chain without memory.

A conversational memory grows inside its files, and curation is not always compression. Figure A2 follows the two deepest builds chunk by chunk. Most growth occurs inside existing files: LoCoMo holds three files from chunk 11 of 85, PersonaMem 128k a single file until chunk 189 of 271 and two thereafter, and every later curation event, 226 on LoCoMo and 283 on PersonaMem, is an edit; `gpt-5.4-mini` never moves or deletes a file (`gemma-4-31B-it` moves and deletes a handful, Figure A12). The memory therefore grows mainly as sections inside existing files, 1.4 new sections per chunk on LoCoMo and 0.8 on PersonaMem, linearly through three quarters of the stream: the relocation of hierarchy into headings that Section 4.1 found at the endpoint is how these memories grow all along, not a late reorganization. Volume splits by benchmark. **The LoCoMo memory ends thirty-five percent *larger* than its input stream**, and the ratio keeps rising (0.87, 1.05, 1.21 and 1.35 at the four quarter marks), curation as elaboration: structure, resolved references, locators; **PersonaMem 128k compresses to a steady quarter of its stream** from the first quarter on. A memory kept organized is not necessarily a memory kept small. Maintenance shows no economy of scale: build rounds per chunk rise from 9.9 in the first quarter to 12.7 in the last on LoCoMo and from 6.9 to 8.2 on PersonaMem, so each new chunk costs slightly more to integrate as the memory it joins grows. Memory health on this side holds almost by construction: the files created in the first chunks are never replaced, so the early files do not merely survive, they are the memory.

Memory keeps its advantage as it grows, in two profiles: raw logs gain late, curated memories early. Figure A3 plots each chain's running success minus the no-memory chain's at the same position. In absolute terms every chain rises from its first 35 tasks to its last 35, the no-memory chain included (40.0 to 71.4 percent at the mini execution agent, 71.4 to 80.0 at `gpt-4.1`), which is why the differenced view is the honest one. The raw Episode log has an advantage of about 11 points over no memory in the first 35 tasks and 20 points in the last 35 at both tiers. The curated

Table A26: ScienceWorld success by task family (%, n=20 each; Section B.6). *Find*: find a living thing; *Grow*: grow a plant; *Life*: longest-lived organism; *Melt*: measure a known melting point; *Cond*: test conductivity; *Cond-U*: test conductivity of an unknown substance; *Therm*: use a thermometer. *All*: success over the 140 tasks (the three-chain cells also appear in Table A12). Third and fourth blocks: single-chain runs at the two execution agent tiers. Last blocks: the cross-family replication of Section F on ScienceWorld, with `claude-sonnet-5` and `gemma-4-31B-it` as management agent and retriever at both execution agent tiers; the No memory rows use neither role, so they do not depend on the family and are not repeated.

	Find	*Grow*	*Life*	*Melt*	*Cond*	*Cond-U*	*Therm*	*All*
Execution agent `gpt-4.1`								
No memory	60.0	0.0	90.0	25.0	50.0	35.0	25.0	40.7
Episode log	100.0	0.0	100.0	30.0	50.0	30.0	55.0	52.1
Curated skills	30.0	65.0	80.0	25.0	65.0	25.0	35.0	46.4
Curated skills+mem	65.0	50.0	100.0	25.0	40.0	30.0	30.0	48.6
Curated skills+mem (GS)	45.0	35.0	100.0	30.0	30.0	60.0	40.0	48.6
Execution agent `gpt-4.1-mini`								
No memory	25.0	0.0	100.0	15.0	0.0	5.0	10.0	22.1
Episode log	30.0	0.0	100.0	20.0	5.0	30.0	25.0	30.0
Curated skills	25.0	30.0	100.0	45.0	20.0	5.0	55.0	40.0
Curated skills+mem	15.0	0.0	95.0	35.0	0.0	0.0	50.0	27.9
Curated skills+mem (GS)	45.0	20.0	95.0	35.0	0.0	30.0	55.0	40.0
Full chain, execution agent `gpt-4.1`								
Episode log	75.0	0.0	100.0	50.0	50.0	25.0	55.0	50.7
Curated skills+mem (GS)	40.0	65.0	100.0	80.0	65.0	30.0	85.0	66.4
Full chain, execution agent `gpt-4.1-mini`								
Episode log	35.0	0.0	100.0	0.0	15.0	20.0	15.0	26.4
Curated skills+mem (GS)	35.0	15.0	95.0	5.0	5.0	25.0	65.0	35.0
`claude-sonnet-5` *management agent and retriever, execution agent* `gpt-4.1`								
Episode log	100.0	0.0	100.0	35.0	45.0	30.0	50.0	51.4
Curated skills	80.0	45.0	100.0	85.0	70.0	50.0	95.0	75.0
Curated skills+mem	65.0	45.0	100.0	80.0	80.0	40.0	45.0	65.0
Curated skills+mem (GS)	80.0	95.0	100.0	70.0	70.0	35.0	95.0	77.9
`claude-sonnet-5` *management agent and retriever, execution agent* `gpt-4.1-mini`								
Episode log	20.0	0.0	95.0	15.0	10.0	0.0	35.0	25.0
Curated skills	60.0	65.0	95.0	65.0	50.0	55.0	60.0	64.3
Curated skills+mem	65.0	25.0	95.0	70.0	55.0	50.0	80.0	62.9
Curated skills+mem (GS)	85.0	20.0	95.0	60.0	55.0	25.0	65.0	57.9
`gemma-4-31B-it` *management agent and retriever, execution agent* `gpt-4.1`								
Episode log	75.0	0.0	100.0	50.0	60.0	30.0	5.0	45.7
Curated skills	85.0	35.0	100.0	20.0	45.0	20.0	20.0	46.4
Curated skills+mem	40.0	35.0	100.0	15.0	65.0	40.0	70.0	52.1
Curated skills+mem (GS)	85.0	30.0	95.0	35.0	30.0	30.0	90.0	56.4
`gemma-4-31B-it` *management agent and retriever, execution agent* `gpt-4.1-mini`								
Episode log	30.0	0.0	90.0	10.0	0.0	10.0	20.0	22.9
Curated skills	10.0	0.0	70.0	10.0	10.0	10.0	30.0	20.0
Curated skills+mem	30.0	15.0	60.0	25.0	15.0	5.0	40.0	27.1
Curated skills+mem (GS)	45.0	10.0	95.0	35.0	5.0	5.0	45.0	34.3

memories already improve success in the first 35 tasks. Most have a smaller lead in the last 35: Curated skills+mem (GS) changes from $+26$ to $+20$ at mini, Shell from $+31$ to $+23$, and the gpt-5.4-built memory from $+37$ to $+23$. GS with the stronger execution agent is an exception, increasing from about $+9$ to $+11$ points. These first-to-last comparisons do not imply a steady rise in between. Raw experience adds up late; distilled guidance lifts success from the first tasks. The compounding is steepest where the execution agent is weakest: the mini-tier Episode log solves 51.4 percent of its first 35 tasks, the no-memory rate for the whole chain (52.9), and 91.4 percent of its last 35, its invalid-action rate falling from 40 to 24 percent along the way. That 91.4 exceeds the stronger tier's

Table A27: ScienceWorld: build and deployment cost by role over the 140 tasks, the counterpart of Table A23 (*Build* = management agent dollars, zero for the mechanically written Episode log; *Search*, *Exec* = retriever and execution agent dollars; *Deploy* = Search + Exec; cents per task and per solved task; *Succ* = success (%) over the 140 tasks). `gpt-5.4-mini` blocks: priced by token count at the rates of Table A23 (these cells ran without a billed-cost log); third block: single-chain runs. `claude-sonnet-5` blocks (Section F): *Build* and *Search* are its billed dollars apportioned to the two roles by rate-card cost, as in Table A36; the No memory rows use neither role, so they do not depend on the family and are not repeated. `gemma-4-31B-it` blocks: as in Table A36, *Build* and *Search* give the self-served model's token volume in millions (input plus output, thinking included) and the dollar columns cover the execution agent only. –: no such role, or no rate.

	Build $	*Search $*	*Exec $*	*Deploy $*	*¢/task*	*¢/succ*	*Succ*
Execution agent `gpt-4.1`							
No memory	–	–	18.10	18.10	12.9	31.8	40.7
Episode log	0	11.45	32.25	43.70	31.2	59.9	52.1
Curated skills	9.17	2.74	17.79	20.52	14.7	31.6	46.4
Curated skills+mem	13.59	3.34	14.97	18.31	13.1	26.9	48.6
Curated skills+mem (GS)	14.02	4.23	16.22	20.46	14.6	30.1	48.6
Execution agent `gpt-4.1-mini`							
No memory	–	–	4.88	4.88	3.5	15.7	22.1
Episode log	0	8.61	10.17	18.78	13.4	44.7	30.0
Curated skills	9.14	2.78	4.80	7.58	5.4	13.5	40.0
Curated skills+mem	15.97	3.48	4.49	7.97	5.7	20.4	27.9
Curated skills+mem (GS)	16.51	4.62	4.13	8.75	6.2	15.6	40.0
Full chain, Execution agent `gpt-4.1`							
Episode log	0	15.57	29.52	45.08	32.2	63.5	50.7
Curated skills+mem (GS)	16.80	6.02	10.58	16.60	11.9	17.9	66.4
Full chain, Execution agent `gpt-4.1-mini`							
Episode log	0	14.71	13.68	28.39	20.3	76.7	26.4
Curated skills+mem (GS)	14.64	5.87	4.15	10.02	7.2	20.5	35.0
`claude-sonnet-5` *management agent and retriever, execution agent* `gpt-4.1`							
Episode log	0	5.49	34.74	40.23	28.7	55.9	51.4
Curated skills	8.31	2.01	14.04	16.05	11.5	15.3	75.0
Curated skills+mem	12.45	1.97	19.53	21.49	15.4	23.6	65.0
Curated skills+mem (GS)	13.13	4.28	10.14	14.42	10.3	13.2	77.9
`claude-sonnet-5` *management agent and retriever, execution agent* `gpt-4.1-mini`							
Episode log	0	7.60	16.95	24.55	17.5	70.1	25.0
Curated skills	11.59	2.46	5.90	8.36	6.0	9.3	64.3
Curated skills+mem	14.68	2.44	4.83	7.27	5.2	8.3	62.9
Curated skills+mem (GS)	18.88	6.18	3.80	9.99	7.1	12.3	57.9
`gemma-4-31B-it` *management agent and retriever, execution agent* `gpt-4.1`							
Episode log	0	5.1M	23.57	–	–	–	45.7
Curated skills	6.5M	1.6M	18.71	–	–	–	46.4
Curated skills+mem	8.1M	2.7M	13.73	–	–	–	52.1
Curated skills+mem (GS)	6.7M	1.9M	12.39	–	–	–	56.4
`gemma-4-31B-it` *management agent and retriever, execution agent* `gpt-4.1-mini`							
Episode log	0	4.5M	5.06	–	–	–	22.9
Curated skills	7.9M	1.8M	5.99	–	–	–	20.0
Curated skills+mem	8.5M	3.0M	4.00	–	–	–	27.1
Curated skills+mem (GS)	8.3M	2.5M	4.09	–	–	–	34.3

three-chain average (87.1), but the comparison uses a different task subset and protocol, so it does not isolate capability. The management agent backbone chains climb the same way, the nano chain rising from 63 percent over its first 35 tasks to 89 over its last, and the inversion of Section 4.2.1 reproduces under end-to-end accumulation (Episode log 88.6 over GS 84.3 at `gpt-4.1`, GS 76.4 over Episode log 73.6 at mini).

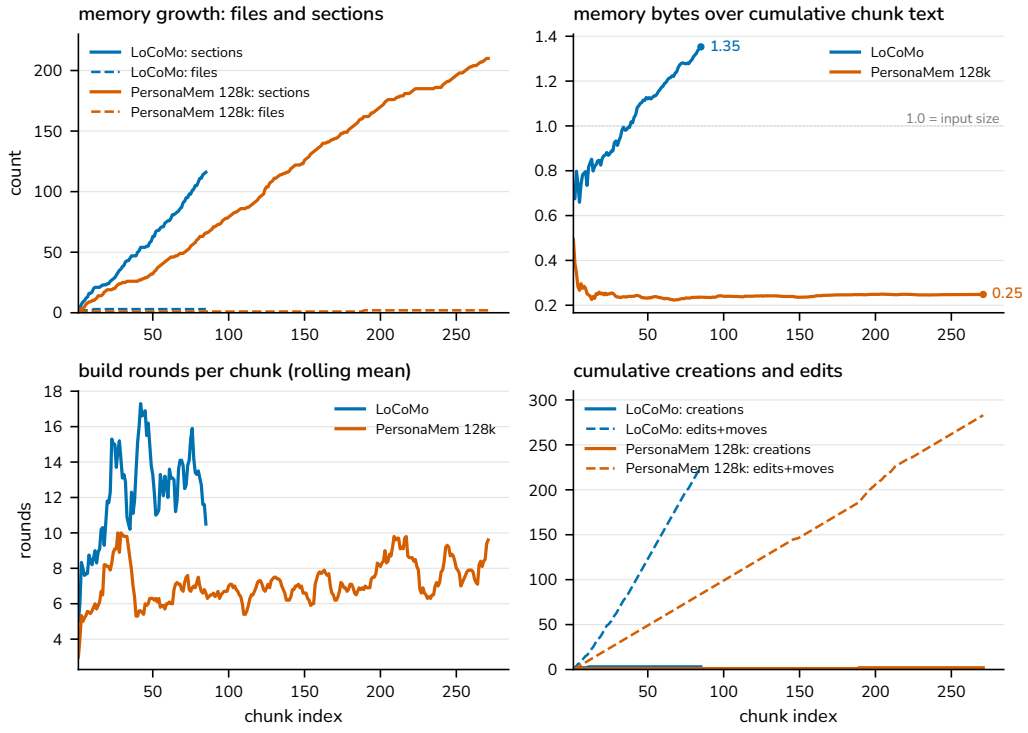


Figure A2: Conversational build curves from the per-chunk trajectories of the builds of record. Top left: LoCoMo has three files from chunk 11 onward, while PersonaMem 128k adds a second file at chunk 189. Most growth occurs as additional headings inside existing files. Top right: memory bytes over cumulative chunk text; the LoCoMo memory ends thirty-five percent *larger* than its input stream (curation as elaboration: structure, resolved references, locators), while PersonaMem 128k compresses to a steady quarter. Bottom left: build effort per chunk does not fall as the memory grows; it drifts up by a fifth to a quarter, so there is no economy of scale. Bottom right: new file creation is sparse; conversational building is edit-dominated from the start, unlike the skill management agents of Figure A9.

Memory shortens episodes more the longer it accumulates. The per-task records give a second measure of usefulness that the endpoint tables do not show: the length of the episode. Mean steps per task fall along every chain, but far more with memory than without. At the mini execution agent the no-memory chain goes from 35 steps per task over its first 35 tasks to 26 over its last, the Episode log chain from 30 to 13 and the GS chain from 28 to 18; at `gpt-4.1` the no-memory chain goes from 28 to 24 and the Episode log chain from 20 to 9. **By the last stretch of the chain the weak execution agent with the raw log finishes in half the steps of the no-memory chain, and the strong one in a third.** The execution-side saving of a memory grows along the chain, and it is largest for the raw log, whose retrieval is also the most expensive (below): the log's higher search cost comes with the larger execution saving.

The two memories teach different task families. Figure A5 splits the differenced curves by goal family at the mini execution agent. Transfer onset is family-specific, and the two memories transfer different things. The Episode log's lead over no memory is largest where the execution agent can imitate a routine from a transcript: $+56$ points on clean (from task 21) and $+40$ on cool (from task 40), above Curated skills+mem (GS)'s $+44$ and $+28$. The log and no-memory chains both solve 2 of 16 heat tasks. Curated skills+mem (GS) improves heat success by $+31$ points, with its first positive cumulative advantage at task 23, and look by $+46$ against the log's $+15$. Two-object placement changes less for either ($+8$ and $+13$, both positive from task 54) and simple placement is saturated at 97 percent for all. **Raw transcripts transfer routines; distilled guidance transfers procedures**,

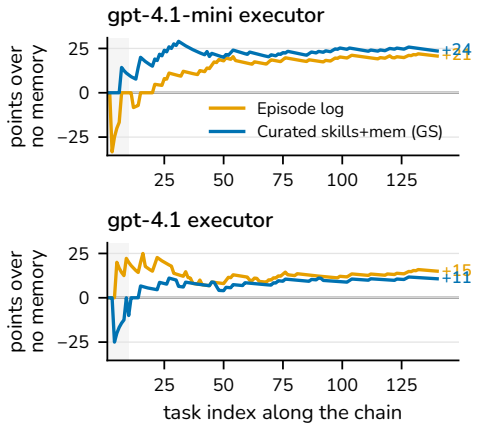


Figure A3: Memory's lead as the memory grows: running success minus the no-memory chain's running success at the same position of the fixed 140-task order (points). Subtracting the no-memory result compares the same task prefixes and accounts for their observed baseline success; it does not isolate accumulation from all effects of task order. At the weak execution agent, cumulative success exceeds the no-memory baseline from task 7 for Curated skills+mem (GS) and task 21 for Episode log; GS remains ahead of the log after task 6. At the strong tier, the log exceeds the no-memory baseline from task 5. Early spikes reflect small denominators.

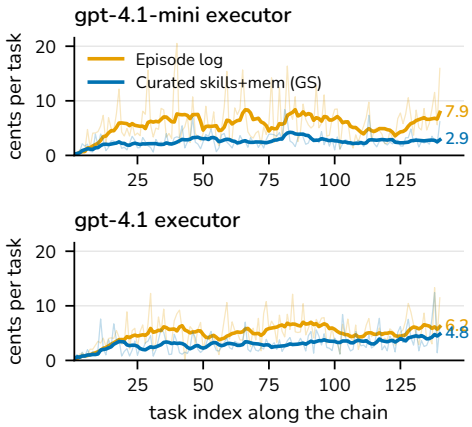


Figure A4: Search cost per task along the full chain (cents at the retrieval rate card; faint lines per task, bold lines a 10-task rolling mean). The Episode log's serve-everything retrieval climbs as its memory grows at both execution agent tiers; at the mini tier it stays a multiple of the flat Curated skills+mem (GS) price throughout, and the gap opens within the first quarter of the chain.

and the execution agent inversion of Section 4.2.1 is, family by family, a statement about which tasks a memory can teach the agent that reads it.

Accumulation itself is worth points, most where the execution agent is weak. The same variants run both ways, so the single-chain minus three-chain gap prices end-to-end accumulation against three shorter independent chains. The no-memory control does not move at all (73.6 and 52.9 in both kinds of runs, the order-invariance check), and GS is nearly unchanged (+2.1 at `gpt-4.1`, 0.0 at mini). The Episode log gains +1.4 at the strong execution agent and +7.1 at the weak one (66.4 to 73.6): **a memory of raw episodes keeps adding points in later tasks precisely for the execution agent that extracts least from each**, while curated guidance reaches its value within a third of the chain. The two kinds of runs differ in chain length and in build draws, so we read the deltas as suggestive rather than settled.

The cost of growth: search scales with the memory, curation with the episodes. Figure A4 follows search cost per task along the chain and Figure A6 the management agent's effort per episode. Over the whole chain, the Episode log's retrieval costs 2.25 times Curated skills+mem (GS)'s at the mini execution agent and 1.62 times at the stronger tier: 5.0 against 1.9 cents per task over the first quarter and 5.7 against 2.7 over the last at mini, $8.26 against $3.66 over the 140 tasks ($7.14 against $4.41 at `gpt-4.1`). It rises through the third quarter (6.6 cents) as the search agent retrieves more of a growing memory (2.5 files per task over the first 35 tasks, 3.9 over the last); the three-chain runs put the same gap at $7.88 against $2.97 (Table A23). Curated memories mature instead: the gpt-5.4 management agent adds only two net files after task 35, ending with 16, while its stored text grows from about 43 to 104 kilobytes and continues to be edited, and GS's per-task retrieval stays the lowest of all, while the Episode log adds one file per task, reaching 140 files and 270 kilobytes

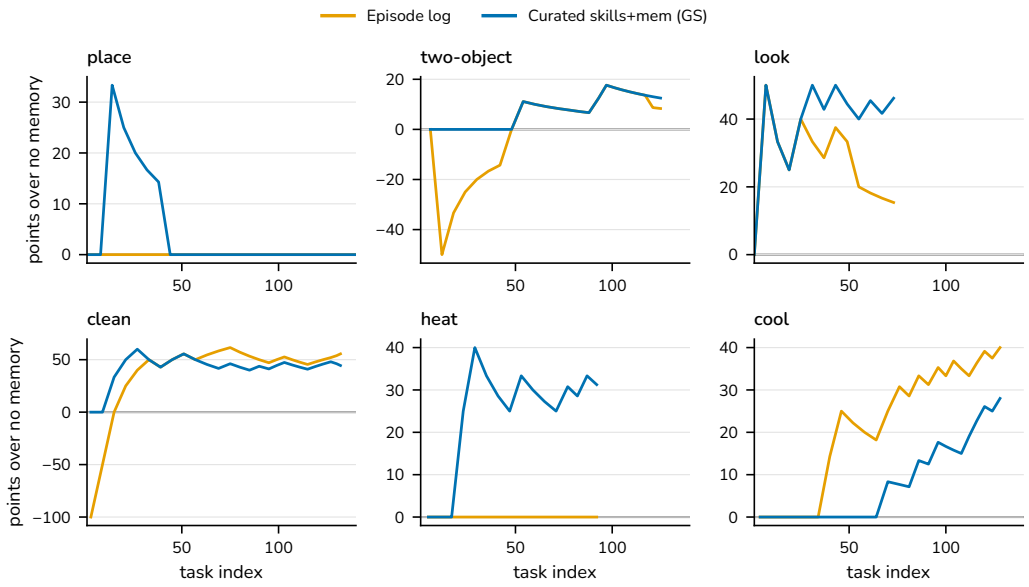


Figure A5: Per-family differenced running success at the family's interleaved positions (mini execution agent, Episode log against Curated skills+mem (GS), both minus the no-memory chain); each curve ends at its family's last task in the fixed order (look's 13 tasks all fall by position 73, heat's 16 by 92). Transfer onset is family-specific: the Episode log's clean advantage builds steadily, Curated skills+mem (GS) takes its two-object and look leads early, and heat is lifted only by Curated skills+mem (GS).

by chain's end at mini. **The cost of serving everything scales with the memory itself.** Management cost, however, scales with the episodes and never trends to zero. Smoothed management effort generally remains around six to twelve rounds per episode, with individual episodes spanning two to twenty-five rounds in these chains, $10.82 over the mini GS chain; the effort drifts down by a quarter on that chain as its memory matures (11.4 to 8.9 rounds, 8.6 to 6.3 cents per episode) and up on the `gpt-4.1` chain whose memory keeps growing (6.6 to 8.0 rounds). Even the gpt-5.4 management agent, which adds few files after task 35, averages about eight to ten rounds per episode across the four quarters on maintenance, and the nano management agent works at a similar effort while producing the spread-out memory of Figure A7. Maintenance remains substantial even where file creation slows; effort declines in some chains but does not approach zero within this horizon. Counting curation, then, **the curated memory costs more per task than the raw log for the whole chain**, about 9 to 12 cents of search plus curation against the log's 5 to 7 cents of search at mini, and at the log's own second-half trend the two lines would cross only around task 870, an extrapolation that holds the curated memory's per-task cost at its last-quarter mean; the log's cost fell in the last quarter, so no crossing is observed. In the embodied setting as in the conversational one (Section 4.2.1), organization is a long-horizon cost: within 140 tasks it is repaid in success and shorter episodes, not in dollars.

Size and compression along the chain are capability signatures. Figure A7 follows memory size along the chain. The Episode log grows linearly by construction while the curated variants grow concavely, the Shell chain consolidating hardest (18 files at task 35, 36 at the end); across management agent backbones the nano agent adds files linearly to 114, mini sublinearly to 45, and gpt-5.4 grows from fourteen files at task 35 to sixteen at the end, first reaching sixteen at task 119. Figure A8 divides memory bytes by the chain's own cumulative trajectory text. Curated skills+mem (GS) at mini compresses to about a quarter of what the execution agent experienced, and the gpt-5.4 management agent to 0.47 (measured against a smaller stream, since its frequent successes end episodes early). The nano management agent ends highest at 0.64, writing more memory bytes per experienced byte than even the Episode log retains (0.48; its step near task 30 is one runaway

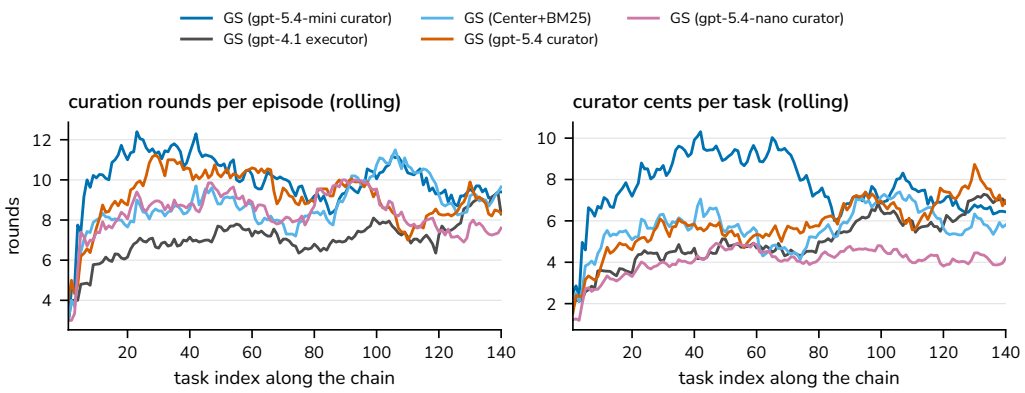


Figure A6: Curation effort per episode along the chain (20-task rolling), from the per-episode curation records; cents are priced at one shared rate card, so the panel compares effort rather than price. Effort does not trend to zero at any capability: even the gpt-5.4 management agent, whose memory stops growing a quarter of the way in, keeps spending eight to ten rounds per episode on maintenance, and the nano management agent works the chain at a similar per-episode effort while producing the spread-out memory of Figure A7. This matches the edit-dominated phase of Figure A9 and the flat per-chunk build effort of Figure A2.

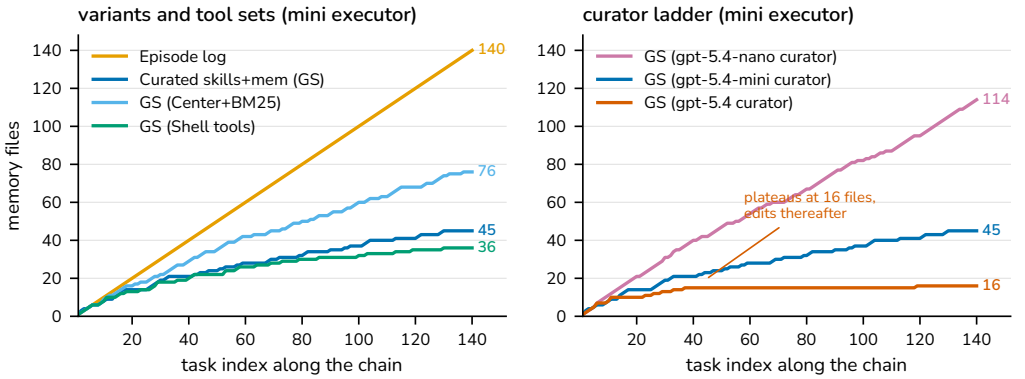


Figure A7: Memory size along the full chain (files). Left: the Episode log grows linearly by construction while curated variants grow concavely, the Shell-tools chain consolidating hardest. Right: the management agent backbone-capability study (Curated skills+mem (GS) chains varying only the management agent backbone) turns memory size into a capability signature, gpt-5.4-nano adding files linearly, mini sublinear, and gpt-5.4 adding few files after the first quarter, reaching sixteen at task 119, while editing the files it has.

300-kilobyte episode at 218 times the chain's median, after which the ratio climbs back). The final memory-to-trajectory ratios are 0.64, 0.25 and 0.47 for nano, mini and gpt-5.4, respectively. The weakest management agent retains the largest fraction of its own trajectory text, but compression does not run in one direction with backbone capability.

Early memories survive, and maintenance is where capability shows. Figure A9 reconstructs every memory's life at the path level from the per-task snapshots. Most files present at task 35 remain at the same path at task 140, although their contents are often edited; at most two early files per memory are absent at the end by both their original path and their original content hash, which does not establish whether every fact survives later edits. What distinguishes management agents is the kind of continued attention. The gpt-5.4 management agent records 20 of its 23 file creations

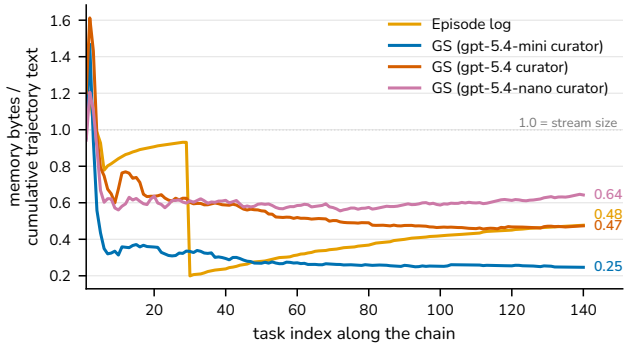


Figure A8: Memory bytes over the cell's own cumulative streamed trajectory text (approximated by the archived trajectories' text length; denominators are cell-specific streams, since execution agent behavior sets how much text each chain generates). The Episode log's step near task 30 is one runaway episode, a 300-kilobyte trajectory at 218 times the chain's median, that inflates its stream; the ratio then climbs back to 0.48 by chain's end. Curated skills+mem (GS) at mini compresses to about a quarter; the gpt-5.4-nano management agent chain ends highest (0.64), writing more memory bytes per experienced byte than even the Episode log retains; the gpt-5.4 management agent's 0.47 is measured against a smaller stream, since its frequent successes end episodes early.

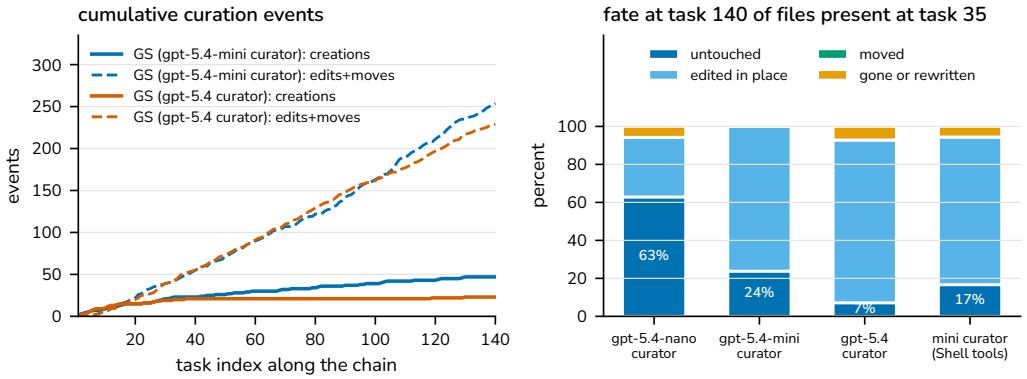


Figure A9: Memory lifecycle at the path level, from the per-task snapshots; every series is a Curated skills+mem (GS) chain, varying the management agent backbone (and, in the last bar, the harness). Left: cumulative file changes between consecutive task snapshots. The gpt-5.4 management agent records 20 of its 23 file creations by task 35, while changed-path events increase from 49 to 229. Right: endpoint status of files present at task 35: unchanged at the same path, changed at the same path, the same content at a different path, or absent by both original path and hash. These categories describe file persistence and do not measure retention of every fact.

by task 35 and three more afterwards, while its edits rise from 49 to 229, with seven deletions. It leaves only one of its fourteen early files untouched, where the mini agent leaves five of twenty-one, the Shell chain three of eighteen and nano twenty-two of thirty-five: **stronger curation shows as broader in-place maintenance, not replacement**.

Growth threatens organization in two ways; the strongest backbone's memory avoided both. Figure A10 tracks organization at the content level, the sections of each memory and the distance-mirrors-relatedness correlation (B4) at every tenth task, and Table A28 gives all four contract metrics of Section A.2 after task 40 and after task 140 for every chain whose per-task memory could be reconstructed. Sections keep growing wherever files do: the chain at the stronger execution agent grows the most internal structure (263 sections), Center+BM25 183 and the mini GS chain 99,

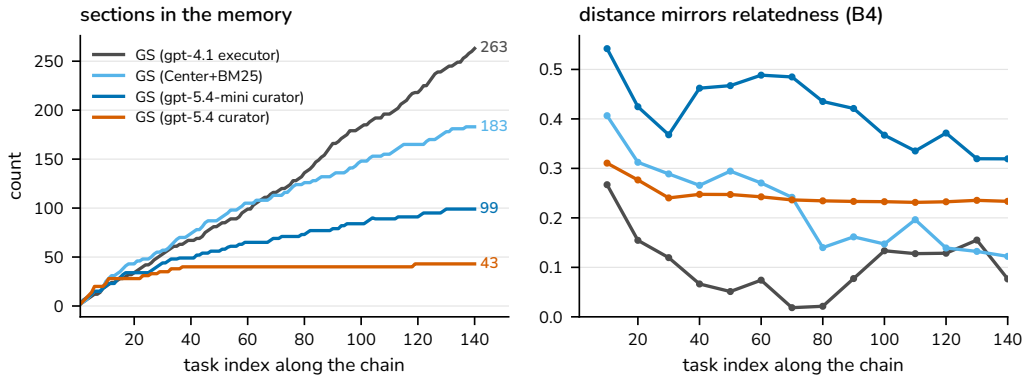


Figure A10: Content-level hierarchy along the chain, from memories reconstructed after every task and validated against the per-task snapshots (560 states, zero mismatches). Left: sections; the chain at the stronger execution agent grows the most internal structure (263 sections) while the gpt-5.4 management agent's grows from 38 sections at task 35 to 43 at the end. Right: the distance-mirrors-relatedness correlation at every tenth task; early points rest on few units, and the measure leans with granularity, but the drift is consistent: adherence to the taxonomy contract erodes in the well-organized memories that keep growing, while the strongest management agent's memory holds it roughly constant across the full chain.

while the gpt-5.4 management agent increases from 38 sections at task 35 to 43 at the end. Early B4 points rest on few sections and the measure leans with granularity, but the drift by 20-task block is consistent, and it has two faces. The first is spread: memories that never had a fit between layout and relatedness and grow without acquiring one, GS at `gpt-4.1` at .06 after task 40 and .08 after 140 with 137 files (its plotted curve falls from .21 to .02 over the first 80 tasks), the nano-built memory at .07 after task 40 and .12 after 140 with 114 files. The second is dilution: memories that start well organized and lose the fit as they keep adding files, the mini GS chain from .46 to .30 while going from 21 to 45 files, Center+BM25 from .26 to .14 while going from 29 to 76, and the ScienceWorld chain from .24 to .17 while going from 19 to 32 (Table A28). **Adherence to the taxonomy contract erodes in the well-organized memories that keep growing, while the strongest management agent's memory holds it roughly constant across the full chain** (.25 to .23), the one agent that added only two files after the first quarter and edited thereafter; the families' chains that stop adding files roughly hold theirs too (`claude-sonnet-5`'s seven files, .33 to .27). **Spread and dilution are the two ways a growing memory loses its organization; the memory that avoided both is the one whose management agent shifted early from creating files to editing them, an association these chains cannot separate from the backbone itself.**

Across families, and the horizon this does not reach. Two of these readings hold for every family we ran (Section F): the management agent's work per episode does not shrink away along the chain for `claude-sonnet-5` or `gemma-4-31B-it` either, and early files survive; `gemma-4-31B-it` is the one agent that moves and deletes a few conversational files. Within 140 tasks and one conversation length nothing here measures months-long accumulation, decay by neglect, or the point at which spread or dilution begins to cost answers; that horizon is the next thing this growth design is built to probe.

Table A28: Contract adherence along the single-chain runs: the four adherence metrics of Section A.2 (B1 sibling label distinctness, B2 sibling content cohesion, B3 scope leakage, B4 distance mirrors relatedness) and the file count of the memory after task 40 and after task 140, for every single-chain run whose per-task memory was reconstructed from its curation record and validated against the run records (zero mismatches). *Management* and *Execution* give the two agents' backbones; *Memory*: GS is the Curated skills+mem (GS) chain of that agent pairing on ALFWorld, Center+BM25 the same chain under the ranked-search harness, ScienceWorld the same variant on ScienceWorld. Sibling-label distinctness remains high and measured leakage remains low. Both sibling cohesion and the distance correlation change along several chains. The largest declines in B4 occur in selected memories that begin with higher values and continue adding files; this pattern is descriptive, not a causal test of file creation. Model names are shortened: 5.4 nano, 5.4 mini and 5.4 are `gpt-5.4-nano`, `gpt-5.4-mini` and `gpt-5.4`; 4.1 and 4.1 mini are the two execution agent tiers; sonnet 5 is `claude-sonnet-5`; gemma 31B is `gemma-4-31B-it`.

			After task 40					**After task 140**				
Management	**Execution**	**Memory**	*Files*	*B1*	*B2*	*B3*	*B4*	*Files*	*B1*	*B2*	*B3*	*B4*
5.4 mini	4.1	GS	34	0.94	3.62	0.00	0.06	137	0.90	3.77	0.01	0.08
5.4 mini	4.1 mini	GS	21	0.99	2.33	0.00	0.46	45	1.00	2.93	0.00	0.30
5.4 mini	4.1 mini	Center+BM25	29	1.00	3.29	0.00	0.26	76	0.97	2.93	0.01	0.14
5.4	4.1 mini	GS	15	0.99	1.95	0.00	0.25	16	0.99	1.98	0.00	0.23
5.4 nano	4.1 mini	GS	40	0.92	2.65	0.02	0.07	114	0.96	3.69	0.02	0.12
5.4 mini	4.1	ScienceWorld	19	1.00	4.00	0.00	0.24	32	1.00	3.35	0.00	0.17
sonnet 5	4.1	GS	7	0.96	1.80	0.00	0.33	7	0.96	1.72	0.00	0.27
sonnet 5	4.1 mini	GS	8	0.97	1.62	0.00	0.19	8	0.97	1.51	0.00	0.18
gemma 31B	4.1	GS	9	0.99	4.18	0.00	0.37	12	0.99	2.79	0.00	0.30
gemma 31B	4.1 mini	GS	13	0.99	2.50	0.00	0.27	19	0.99	2.27	0.00	0.21

Table A29: Model families as the management agent and search agent (Section F): answer correctness (judged, %; PersonaMem judged option matching) with those two roles run by three families under frozen prompts, seeds, chunking and judge (declared deltas in Section F). **5.4m**: `gpt-5.4-mini` (the default family, Table A9); **gem**: `gemma-4-31B-it`; **son**: `claude-sonnet-5`. †: the `gemma-4-31B-it` pass could not complete within its 32k serving window before making any edit, so the searched memory equals the flat dump; –: the pass made no change (no folders), so no separate search was run. Citation support and the PersonaMem 32k tier are in Table A33.

	LoCoMo			PersonaMem 128k			REALTALK		
Memory	*5.4m*	*gem*	*son*	*5.4m*	*gem*	*son*	*5.4m*	*gem*	*son*
Baselines									
Closed-book	18.4	0.0	4.4	47.6	61.9	57.1	10.6	0.0	0.0
Chunk retrieval	81.6	73.4	84.2	66.7	73.8	71.4	71.8	68.2	70.6
Filesystem memories									
Verbatim dump	84.2	82.3	78.5	69.0	66.7	73.8	77.6	64.7	67.1
Foldered sessions	86.1	–	79.1	76.2	–	76.2	77.6	68.2	68.2
Reorg. (preserve)	82.9	77.8	80.4	59.5	69.0†	66.7	77.6	71.8†	64.7
Reorg. (condense)	79.1	59.5	69.6	54.8	64.3†	66.7	41.2	22.4	63.5
Agent-curated memory	86.1	84.2	87.3	66.7	57.1	73.8	75.3	52.9	84.7

Table A30: Model families as the management agent and search agent of the skill setting (Section F), three-chain runs on ALFWorld: success over the 140 tasks (%) with those two roles run by three families, the execution agent tiers of Table A11 held fixed. **5.4m**: `gpt-5.4-mini` (the default family, Table A11); **gem**: `gemma-4-31B-it`; **son**: `claude-sonnet-5`. No memory has no management agent or retriever, so its columns differ only by run-to-run variation of the same execution agent, the noise floor for the other rows (2 to 5 points). Bold: the best memory per column.

	Execution agent `gpt-4.1`			Execution agent `gpt-4.1-mini`		
Memory	*5.4m*	*gem*	*son*	*5.4m*	*gem*	*son*
No memory	73.6	77.1	75.7	52.9	55.0	57.9
Episode log	**87.1**	90.0	88.6	66.4	64.3	71.4
Curated skills	80.7	91.4	**95.7**	65.7	74.3	87.1
Curated skills+mem	75.7	85.7	94.3	55.0	**80.0**	89.3
Curated skills+mem (GS)	82.1	**93.6**	94.3	**76.4**	76.4	**92.1**

F Generality Across Model Families (RQ4)

Every result above runs the management agent and search agent on one family, `gpt-5.4-mini`. We re-ran the conversational, harness and skill experiments of Sections 4.1, 4.1.3, 4.2 and E.3 with `gemma-4-31B-it` (served with thinking on) and `claude-sonnet-5` at medium effort in those two roles, everything else frozen up to a few declared serving differences, listed with the per-cell checks against the `gpt-5.4-mini` cells in Section B.7; the ScienceWorld cells of Section 4.2 were re-run with both families (Section B.6).

The question is not whether the numbers repeat, they cannot across models, but which of the paper's answers survive a change of the two agents that write and read the memory, and, where an answer moves, whether it moves with the family's writing, its reading, or its serving window. The No memory skill cells use neither replicated role. Their observed variation of 2 to 5 points provides context for small differences, but does not establish a statistical noise threshold for other comparisons.

Organization across families (RQ1). Table A35 repeats the build accounting of Table A5 and Table A44 the shape panel of Table A6. The effort differs less than the memories do. Curating a conversation takes `claude-sonnet-5` 33 to 63 percent fewer tool calls than `gpt-5.4-mini` (619 against 1,660 on LoCoMo, 1,336 against 1,986 on PersonaMem 128k), and `gemma-4-31B-it`

lands close to claude-sonnet-5 on every benchmark; at its rates claude-sonnet-5's builds cost less than gpt-5.4-mini's on LoCoMo ($8.2 against $12.9) and more on PersonaMem 128k and REALTALK ($17.6 against $13.4, $10.5 against $7.6).

What the families write is their own. On LoCoMo and REALTALK, both replicated families curate a conversation into two files, where gpt-5.4-mini wrote three on LoCoMo and 41 on REALTALK; on PersonaMem 128k the order reverses, claude-sonnet-5 writing 19 files and 281 kilobytes, the largest curated memory of the three, and gemma-4-31B-it 22 files, against gpt-5.4-mini's two files. gemma-4-31B-it's curated memories are the smallest by size on every benchmark (40 kilobytes against 103 and 136 on LoCoMo), and its foldering pass completes only on REALTALK, where it places the sessions in four folders as gpt-5.4-mini placed them in six and claude-sonnet-5 in five. The shape panel gives the family signature in structural terms: claude-sonnet-5's conversational memories hold two files with wide fan-out and many cross-references (45 and 34 on LoCoMo and REALTALK), where gemma-4-31B-it's hold two files with almost none (2 and 0).

The families' Curated skills+mem (GS) full-chain memories stay flat, with no directories, at a distance-mirrors-relatedness correlation of .18 to .30 around gpt-5.4-mini's .29, while the harness memories of both families add one directory and span .16 to .66.

The reorganization passes show the same dependence on the management agent. Under the Reorg. (preserve) prompt claude-sonnet-5 removes 52 to 74 percent of the characters of the LoCoMo and REALTALK memories, which gpt-5.4-mini kept at their size, and 83 to 92 percent on PersonaMem, where gpt-5.4-mini removed 89 to 96. Under the Reorg. (condense) prompt it removes 65 to 93 percent on every benchmark, where gpt-5.4-mini archived the LoCoMo sessions instead of condensing them (36 files, 114 kilobytes). The searches over claude-sonnet-5's condensed memories gain on some benchmarks and lose on others, and gemma-4-31B-it cannot complete either pass on the larger memories within its window (below).

The harness axis reverses direction with the family: claude-sonnet-5's shell wrote PersonaMem 128k into one file where its file-native harnesses wrote 12 and 19, and gemma-4-31B-it's into 6 against 22 and 25 (Table A38), so both families consolidate under the shell where gpt-5.4-mini sharded (2 to 147). The extra dollars of sharding that gpt-5.4-mini did not show appear for claude-sonnet-5 without any sharding ($25.9 for the shell build against $17.6 and $19.2; $28.2 with the 591 calls replayed from three outage-killed attempts), and its shell memory is also the largest at 329 kilobytes. Per chunk, claude-sonnet-5 builds with 6 to 7 rounds and 5 to 9 tool calls under every harness, against 7 to 15 rounds for gpt-5.4-mini (Section E.1), and gemma-4-31B-it with 5 to 7 rounds.

Memory organization is a signature of the organizing model, and so is the direction of every organization-level effect: with more content the replicated families shard where gpt-5.4-mini consolidated, and under the shell they consolidate where it sharded. What is invariant is that scale and harness change the organization, not which way, so the organization-level claims of Sections 4.1 and 4.1.3 describe the organizing model rather than filesystem memory in general. claude-sonnet-5 uses a third to two thirds as many build tool calls as gpt-5.4-mini. Its curated LoCoMo and REALTALK memories have fewer files and many cross-references, although its PersonaMem 128k memory is larger. In the embodied full-chain GS runs, claude-sonnet-5 writes 7 to 8 files and gemma-4-31B-it 12 to 19, against 45 to 137 for gpt-5.4-mini (Table A39).

Answer quality across families (RQ2, conversational setting). Table A29 repeats the comparison of Table A9 with gemma-4-31B-it and claude-sonnet-5 as the management agent and search agent, Table A33 gives correctness and citation support for every memory, benchmark and family, and Table A46 the question-paired sign tests behind the readings. The rows that involve no writing move little: Chunk retrieval lands within 9 points of its gpt-5.4-mini counterpart everywhere, and the Verbatim dump within 6 points on LoCoMo and PersonaMem 128k but 10 to 13 points below on REALTALK. Without a memory both replicated families abstain on LoCoMo and REALTALK (0.0 to 4.4 correct) and score above gpt-5.4-mini on PersonaMem (61.9 and 57.1 against 47.6), a closed-book caveat treated with its shuffle control in Section B.7.

As with gpt-5.4-mini, no memory variant is consistently the most accurate within any family, and the agent-curated memory is where the families part. It is the best or tied-best memory on LoCoMo for all three (86.1, 84.2 and 87.3). For claude-sonnet-5 it is the best memory on REALTALK (84.7; 18 wins against 3 losses over its own dump, $p < 0.01$) and ties claude-sonnet-5's dump on PersonaMem 128k (73.8) behind the foldered memory (76.2). For gemma-4-31B-it it is the weakest memory on PersonaMem 128k (57.1) and ahead only of the condensing reorganizer on REALTALK (52.9; 2 wins against 21 losses over gpt-5.4-mini's curated memory, $p < 0.01$). The PersonaMem 32k collapse of Section 4.2.1 narrows without vanishing: the curated memory trails the dump by 6 points for claude-sonnet-5 (65.6 against 71.9) and 13 for gemma-4-31B-it (56.2 against 68.8) against 41 for gpt-5.4-mini, and neither gap is significant.

Citation support of the memory-backed rows stays above 82 for claude-sonnet-5 but falls to 77 to 80 on PersonaMem 128k and, apart from its curated memory (85.9), to 75 to 81 on REALTALK for gemma-4-31B-it. Removing the four defective LoCoMo golds (Table A43) raises every memory-backed row of both families by up to 1.5 points and changes no ordering except that claude-sonnet-5's foldered memory comes to tie its dump.

The harness ordering of Section 4.2.3 moves with the family too (Table A37): with claude-sonnet-5 the shell harness matches or beats Center+BM25 on correctness (LoCoMo 89.9 against 88.0, PersonaMem 128k 73.8 against 66.7) at markedly lower citation support (77.6 and 83.3 for the shell, versus 93.1 and 92.1 for gpt-5.4-mini's shell on the same benchmarks). Its two nondefault harnesses produce one to twelve files, where gpt-5.4-mini's produce up to 147; with gemma-4-31B-it the shell harness trails on both benchmarks (74.7 against 76.6 and 50.0 against 69.0).

Set side by side, the three families disagree least about the memories nobody wrote and most about the memories somebody did. Across families the Verbatim dump spans 6 to 13 points per benchmark (84.2, 82.3 and 78.5 on LoCoMo; 69.0, 66.7 and 73.8 on PersonaMem 128k; 77.6, 64.7 and 67.1 on REALTALK). The curated memory spans 3 points on LoCoMo but 17 on PersonaMem 128k, 28 on PersonaMem 32k and 32 on REALTALK, and the condensing reorganizer 41 on REALTALK (41.2, 22.4 and 63.5), where gemma-4-31B-it's pass removed 77 to 96 percent of the characters and its correctness fell below its own dump on every memory it managed to condense (LoCoMo 59.5).

No organization is most accurate on every benchmark for any family. Filesystem memories make citation support measurable, but support scores vary across families and harnesses. On PersonaMem and REALTALK, curated accuracy varies more across families than dump accuracy; on LoCoMo, its range is smaller. The condensing pass has the widest REALTALK range. Because both memory agents change, this comparison does not isolate writing from retrieval.

With claude-sonnet-5 in both management and search, the curated memory is best on LoCoMo and REALTALK, and its gap to the dump on PersonaMem 32k narrows to six points without reaching significance. With gemma-4-31B-it in both roles, it is the weakest memory on PersonaMem 128k. The smaller gaps suggest that the severity of the collapse in Section 4.2.1 depends on the memory agents, consistent with the gpt-5.4 rebuild in Section 4.2.2. Curation still trails the dump on these 32k slices for all three families.

Search economy across families (RQ2, cost). Table A34 repeats the per-query accounting of Table A17 for the two families (the claude-sonnet-5 cents are billed figures, up to 19 percent above the nominal rate-card figure because nearly all uncached input is written to the prompt cache at 1.25 times the input rate).

The economy that organization brings with gpt-5.4-mini recurs with claude-sonnet-5: its reorganized memories search for about half the price of the dump on both PersonaMem tiers (1.9 to 2.2 cents against 3.9 and 4.1) and for 27 to 38 percent less on LoCoMo, and the saving comes from smaller reads over the condensed memories rather than from fewer rounds, which stay at two to four. The curated memory is less expensive than the dump on PersonaMem for claude-sonnet-5 as well (2.7 against 4.1 and 3.2 against 3.9 cents) but the most expensive memory on REALTALK (6.4 cents against 3.3), where claude-sonnet-5 wrote two files that its search agent reads whole.

gemma-4-31B-it makes 2.1 to 6.4 tool calls per query on the memories against 4.4 to 10.5 for gpt-5.4-mini, and its input volume per query stays between 12 and 30 thousand tokens:

lowest for its curated LoCoMo memory (12.5 against the dump's 19.0 thousand) and highest over its condensed REALTALK memory (29.6 thousand at 6.7 rounds), the memory on which its correctness also collapsed. The reorganized memories it could not edit remain the dump and search like it.

Organization's search economy recurs with a stronger search agent, and partly with `gemma-4-31B-it`, by the same mechanism, smaller reads rather than fewer rounds; and a stronger search agent also reads less: `claude-sonnet-5` answers in two to four rounds whatever the memory, where `gpt-5.4-mini` takes four to eight, so at rates about three times higher its queries cost about what `gpt-5.4-mini`'s do. The largest cost increase occurs for `claude-sonnet-5`'s two-file curated REALTALK memory, which its searcher reads whole on questions spanning the session. Curated LoCoMo and both reorganized REALTALK memories also cost more to search than their dumps.

Embodied value across families (RQ2, embodied setting). In Table A30, the No memory and Episode log rows differ from their `gpt-5.4-mini` counterparts by about 2 to 5 points. The executor is unchanged, but the episode-log retriever changes family, so these differences do not isolate executor variation. The curated rows do not. With `claude-sonnet-5` curating, every curated memory beats the Episode log at both tiers (95.7, 94.3 and 94.3 against 88.6 under `gpt-4.1`, by 6 to 7 points; 87.1, 89.3 and 92.1 against 71.4 under `gpt-4.1-mini`, by 16 to 21). With `gemma-4-31B-it` curating the same holds at the mini tier (74.3, 80.0 and 76.4 against 64.3, by 10 to 16 points, $p \leq 0.02$ on task-paired sign tests over the 140 tasks), while at the strong tier none of the three task-paired comparisons with the log is statistically significant (91.4, 85.7 and 93.6 against 90.0; the largest gap, six tasks, has $p = 0.29$).

`claude-sonnet-5`'s curated memories are less expensive to deploy than its episode log. `gemma-4-31B-it`'s execution costs can be compared, but its self-served management and search roles have no dollar rate (Table A36, the counterpart of Table A23). With `gpt-5.4-mini`'s curated memories the `gpt-4.1` execution agent spent as much as with no memory ($13.6 to $16.6 against $17.4); with `claude-sonnet-5`'s it spends less than half ($7.1 to $8.0), and with `gemma-4-31B-it`'s $8.6 to $12.7, so that search plus execution over a `claude-sonnet-5`-curated memory costs 6 to 8 cents per task against 12 cents with no memory, while solving 26 to 28 more of the 140 tasks. At the `gpt-4.1-mini` tier the per-task price lands on both sides of the no-memory cell (2.7 to 3.6 cents against 3.1, at 41 to 48 more tasks), and the Episode log remains the most expensive memory option to deploy for both priced families.

Curating cost `claude-sonnet-5` $4.5 to $7.8 per cell against `gpt-5.4-mini`'s $7.2 to $10.6, at rates about three times higher. On the execution agent side (Table A40) the memories written by `claude-sonnet-5` lower the intrinsic cost of the transcript at both tiers (0.66 to 0.85 million tokens of perfect-cache compute at `gpt-4.1` against 1.01 to 1.18 with `gpt-5.4-mini`'s curated memories), and cache efficiency stays between 92 and 97 percent for every family, so the appending design of the execution agent conversation holds regardless of who wrote the memory.

Per task (Table A42), the execution agent finishes in 12 to 13 environment steps with `claude-sonnet-5`'s curated memories at the `gpt-4.1` tier against 25 with no memory and 22 to 24 with `gpt-5.4-mini`'s, and the success changes differ by goal family (Table A41): in the three-chain runs at that tier the heat family rises from 56 and 62 with no memory to 75 to 88 with `claude-sonnet-5`'s curated memories and 44 to 81 with `gemma-4-31B-it`'s, and the full chains end it at 94 and 88. The libraries behind these numbers are thin, 7 to 19 files in the full chains against 45 to 137 for `gpt-5.4-mini` (Table A39).

The harness differences are smaller: the two harnesses have equal aggregate success for `gemma-4-31B-it` (84.3 each, 118 of 140 with 14 and 16 files, below its curated chain at 87.9; its Center+BM25 curation was cut short by the window from task 100 on); for `claude-sonnet-5` the two chains at the design tier solve 134 and 136 of 140 (95.7 and 97.1, with ten and seven files, a two-task difference), and an extra pair at the `gpt-4.1` tier, outside the design of Section 4.1.3, ties at 136 (97.1, six files each), so the shell's lead of Section 4.2.3 is `gpt-5.4-mini`'s.

ScienceWorld repeats this with the same execution agents (Table A26; the `claude-sonnet-5` blocks give its eight cells and the `gemma-4-31B-it` blocks its eight, the gated cells included). The Episode log cells, whose management agent does not exist, land within 1 and 5 points of their `gpt-5.4-mini` counterparts (51.4 against 52.1 and 25.0 against 30.0), differences that combine

a search agent change with run variation. With claude-sonnet-5 curating, Curated skills and Curated skills+mem (GS) solve 75.0 and 77.9 percent of the tasks under gpt-4.1, where gpt-5.4-mini's memories solved 46.4 and 48.6 and trailed the flat memory (52.1), and 64.3 and 57.9 at the mini tier against 40.0 for both of gpt-5.4-mini's and 25.0 for the flat memory; Curated skills+mem follows, at 65.0 and 62.9 against 48.6 and 27.9 with gpt-5.4-mini curating, with 28 and 31 cap terminations and clipped task scores of 76 and 73.

The lift comes from the same place as on ALFWorld: cap terminations fall to 18 and 12 for Curated skills and Curated skills+mem (GS), respectively, under gpt-4.1 (34 with the flat memory, 40 and 47 with gpt-5.4-mini's corresponding memories), and to 27 and 36 at the mini tier (86 with the flat memory, 64 and 61 with gpt-5.4-mini's). Terminal failures range from 17 to 23. In the same variant order, clipped task scores are 80 and 82 under gpt-4.1, and 74 and 68 at the mini tier, against 57 to 60 for gpt-5.4-mini's corresponding two variants; its mini-tier gated variant scores 42.6.

By family, claude-sonnet-5's memories lift the procedures that gpt-5.4-mini's did not: grow-a-plant rises from 0 with the flat memory to 45 and 95 under gpt-4.1, the melting-point family from 35 to 85 and 70 (80 with the gated memory), and the thermometer family from 50 to 95 with both; the unknown-substance conductivity family is the one where gpt-5.4-mini's Curated skills+mem (GS) memory (60) stays ahead of claude-sonnet-5's Curated skills and Curated skills+mem (GS) memories (50 and 35, respectively).

The memories are thinner, 15 to 21 files summed over the three chains against 20 to 57 for gpt-5.4-mini, and less expensive to use (Table A27): search plus execution over the three claude-sonnet-5-curated variants costs 10.3 to 15.4 cents per task under gpt-4.1 (12.9 with no memory, 13.1 to 14.7 with gpt-5.4-mini's curated variants) and 5.2 to 7.1 at the mini tier (3.5 with no memory, 5.4 to 6.2 with gpt-5.4-mini's). Deployment cost per solved task ranges from 8.3 to 23.6 cents across the two tiers, against 15.7 to 31.8 with no memory; curating cost claude-sonnet-5 $8.3 to $18.9 per cell against gpt-5.4-mini's $9.1 to $16.5, at rates about three times higher.

The gemma-4-31B-it blocks differ. Under gpt-4.1, gemma-4-31B-it's Episode log trails its gpt-5.4-mini counterpart by 6.4 points (45.7 against 52.1), while Curated skills has the same aggregate success (46.4). At the mini tier, the gaps are larger: Episode log scores 22.9 against 30.0, and Curated skills 20.0 against 40.0. Within gemma-4-31B-it, Curated skills is close to the log at both tiers; its Curated skills+mem (GS) memory does lift, by 10 and 11 points over its own flat memory, to 56.4 under gpt-4.1 (above gpt-5.4-mini's 48.6) and 34.3 at the mini tier (below gpt-5.4-mini's 40.0), and both stay 21 to 24 points below claude-sonnet-5's; its Curated skills+mem memory lands at 52.1 under gpt-4.1 and 27.1 at the mini tier, above its Curated skills and Episode log at both tiers (46.4 and 45.7; 20.0 and 22.9) and below its Curated skills+mem (GS) (56.4; 34.3), against 48.6 and 27.9 with gpt-5.4-mini curating.

Two mechanisms show in the records. The first is the window: on the mini tier the raw episode logs of the conductivity families run to tens of thousands of tokens, and the retriever's reading of a few of them overflowed gemma-4-31B-it's 32,768-token window on 10 of the 140 tasks (8 in the two conductivity families, 1 each in the melting-point and grow-a-plant families), which count as failures; under gpt-4.1, whose episodes are shorter, no call overflowed; among the curated memories only the Curated skills+mem memory at the mini tier had a management agent overflow, once, after a conductivity task whose execution had already failed, so its recorded outcome remains a failure.

The second is the curated memory itself: gemma-4-31B-it's memories are larger than claude-sonnet-5's (39 to 54 files summed over the three chains against 15 to 21) and cap terminations remain frequent. Under gpt-4.1, however, all three curated variants hit the cap less often than no memory (33 to 42 versus 53); at the mini tier, Curated skills hits it more often (98 versus 89), whereas Curated skills+mem (GS) hits it less often (68) (42 cap terminations at the flat and Curated skills cells under gpt-4.1 and 38 at the Curated skills+mem (GS) cell, against 35 to 47 with gpt-5.4-mini's memories; 98 and 68 at the mini-tier Curated skills and Curated skills+mem (GS) cells against 61 and 64), the failure that claude-sonnet-5's curation removes; gemma-4-31B-it's curation passes are also the longest of any family, up to 28 thousand output

tokens on one task, which is what stalled two of its chains until the read window was widened (Section B.7).

The clipped task score follows: 51.0, 53.9 and 64.4 under `gpt-4.1`, 36.9, 34.8 and 48.2 at the mini tier. `gemma-4-31B-it`'s two roles carry no rate (Table A27: 1.6 to 5.1 million retriever tokens and 6.5 to 8.5 million management agent tokens per cell, thinking included); the execution agent alone costs $12.4 to $23.6 per cell under `gpt-4.1` and $4.0 to $6.0 at the mini tier.

The executor-tier ordering in Section 4.2.1 changes with the memory-agent family in these three-chain runs. With a stronger management agent the curated memory dominates the raw log for both execution agents on both benchmarks, by 6 to 21 points on ALFWorld and 14 to 39 on ScienceWorld, from compact libraries. On ALFWorld, `claude-sonnet-5`'s strong executor uses less than half the no-memory execution cost and about half as many steps. This cost reduction is not uniform on ScienceWorld: the gated variant's execution cost is $19.53, versus $18.10 without memory. The procedural gains also depend on the variant: `claude-sonnet-5` improves ALFWorld heat and ScienceWorld melting-point performance, while its GS memory is particularly effective for growing a plant. These results are consistent with better distilled guidance helping both executors, but the family swap changes both writing and retrieval. The small ALFWorld harness gaps do not establish that the harness is irrelevant. `gemma-4-31B-it` produces more ScienceWorld files than `claude-sonnet-5` (39 to 54 versus 15 to 21) and has more cap terminations than `claude-sonnet-5` in the compared Curated-skills and GS cells; its comparison with no memory varies by tier and variant.

Growth across families (RQ3). The growth atlas of Section E.3 was drawn from `gpt-5.4-mini`'s chains; Figure A11 redraws its skill panels for the three families, Figure A12 its conversational build panels, Figure A13 search cost along the chain, Figure A14 memory bytes over streamed text, Figure A15 the per-goal-family curves and Figure A16 its content-level hierarchy panel for both families, from memories reconstructed after every task out of identity replays of their chains and validated against the per-task snapshots (840 states, zero mismatches).

Two of its readings hold for every family. The management agent's work per episode does not shrink away along the chain for any family (`claude-sonnet-5`'s falls to about 0.7 of its first-twenty mean by task 30 and moves between 0.45 and 0.7 afterwards, `gemma-4-31B-it`'s drifts down in the last quarter, `gpt-5.4-mini`'s rises; per quarter of the chain, the management agent's output per episode in the last quarter is 0.7 to 1.0 of the first for `claude-sonnet-5` and `gemma-4-31B-it`, so the cumulative management cost is linear in the task index, with $R^2 \geq 0.99$ for all six curated chains), and conversational organization grows inside files, with sections rising roughly linearly while file counts stay small (LoCoMo at two or three files for every family, with 116, 73 and 23 sections for `gpt-5.4-mini`, `claude-sonnet-5` and `gemma-4-31B-it`; PersonaMem 128k at 210, 211 and 125 sections), though on PersonaMem 128k both replicated families add files in steps while editing the early ones, `claude-sonnet-5`'s memory there growing fastest (281 kilobytes against 151 and 135); tool calls per chunk stay flat for every family after the first chunks.

Two change with the family. The stronger management agents write far fewer files and edit them from the first tasks (7 and 12 files created over 140 tasks against `gpt-5.4-mini`'s 147), so the phase in which creations stop and edits continue, which `gpt-5.4-mini` reaches only with the strongest management agent of the backbone-capability study, is where `gemma-4-31B-it` and `claude-sonnet-5` start. The early files survive for every family: of the 76 files present at task 35 across the eight full chains of the two replicated families, 75 are still in the memory at task 140 and 68 have been edited in place (`gpt-5.4-mini`'s mini-tier Curated skills+mem (GS) chain: 21 of 21, 16 edited), so maintenance rather than replacement is the norm wherever curation is strong.

At the content level (Figure A16) the three plotted `claude-sonnet-5` chains end with 16 to 20 sections and the `gemma-4-31B-it` chains with 29 to 84, against 99 to 263 for `gpt-5.4-mini`. Section counts stabilize early in the strong-tier GS chains (last increases at tasks 11 and 82), but smaller additions continue as late as tasks 114 and 133 in the other plotted chains. Correlations also vary: after task 47, `claude-sonnet-5` spans .13 to .32 across the three chains and `gemma-4-31B-it` .21 to .69; `gemma-4-31B-it` ends at .30, .21 and .66. These memories remain compact, but do not share a single point after which their structure is fixed.

And the curated chain's lead over the fixed `gpt-5.4-mini` no-memory reference chain at the same executor tier, which every family opens within the first fifteen tasks, is twice as large for the replicated families at the strong tier (21 to 26 points against 11, where `gpt-5.4-mini`'s curated chain trails its own episode log at 15) and larger at the mini tier (35 and 41 points against 24).

Along the chain (Figure A13) the Episode log's serve-everything retrieval repeats the `gpt-5.4-mini` pattern for `gemma-4-31B-it`, climbing with the memory to 2.4 times the Curated skills+mem (GS) input-token volume at the `gpt-4.1` tier (33 against 14 thousand tokens) and 1.4 times at mini (27 against 19), while `claude-sonnet-5`'s two `gpt-4.1`-tier chains end at similar ten-task mean retrieval costs (4.5 and 4.8 cents), the log's retrieval rising and the curated memory's staying flat until the last quarter, where its retriever begins to read the seven files more fully.

`gemma-4-31B-it`'s curated memories compress the experience to 15 and 17 percent at the two tiers against `gpt-5.4-mini`'s 47 and 25, while the replicated families' computable episode-log ratios stay near the stream size (95 to 105 percent), while `gpt-5.4-mini`'s mini-tier log is at 48 percent because a runaway episode inflates its stream; Figure A14; the `claude-sonnet-5` chains of record were resumed from their run records after the outages and carry the trajectory text of the live tasks only, so their ratio is not computable, and its uninterrupted mini chains end at 0.41 for GS and 0.95 for the log).

Per goal family (Figure A15) transfer is family-specific for every management agent: place gains at most three points, clean 41 to 63 points for every memory, and heat separates the management agents, ending 69 and 88 points over no memory with `gemma-4-31B-it`'s and `claude-sonnet-5`'s curated memories at the mini tier and 38 to 44 with the `gpt-4.1` execution agent, where the Episode log ends at 13 (`claude-sonnet-5`) and -19 (`gemma-4-31B-it`) and, at the mini tier, at -6 (`claude-sonnet-5`) and 31 (`gemma-4-31B-it`).

Running one long chain instead of three short ones also has a different effect: `gemma-4-31B-it`'s log gains 9 points at mini, compared with 7 for `gpt-5.4-mini`, but its curated chain gains 12 (87.9 against 76.4); `claude-sonnet-5`'s log loses one (70.7 against 71.4) and its curated chain gains two (94.3 against 92.1).

The replicated families build compact memories and edit early files throughout the chain. Their GS chains establish and retain a larger lead over the fixed no-memory reference, but section counts and correlations do not freeze uniformly. The observed curves suggest less structural dilution in these runs; they do not establish that dilution is confined to weaker models. Full-chain gains also depend on the executor: they are larger for GS than for logs at the mini tier for both replicated families, while `claude-sonnet-5`'s strong-tier log and GS gains are similar.

The serving window as a factor. `gemma-4-31B-it` is served with a 32,768-token window against about 400k for `gpt-5.4-mini`, with an 8,192-token output reservation (Section B.7 lists every consequence per cell). Its foldering pass produced no memory on LoCoMo, PersonaMem 128k and the 32k slices, and its preserving reorganization left the REALTALK, PersonaMem 128k and 32k memories equal to the dump. Its curated builds lost chunks (2 of 85 on LoCoMo, 10 of 93 on REALTALK, 1 of 271 on PersonaMem 128k, up to 40 of 271 in the harness builds). Its management agent overflowed once on the Curated skills+mem (GS) chain and 17 times on the Center+BM25 chain, its ScienceWorld retriever overflowed on 10 of 140 mini-tier log tasks, and the output cap cut 21 build turns.

Where its condensing pass did complete it removed 77 to 96 percent of the characters and its searches fell below its own dump, with larger accuracy losses than `gpt-5.4-mini`'s condensing at its larger serving window on LoCoMo and REALTALK (LoCoMo 59.5 against 79.1 for `gpt-5.4-mini` and 84.2 for `gemma-4-31B-it`'s own curated memory; REALTALK 22.4 against 41.2).

The observed failures cluster around operations that read or rewrite large amounts of text at once. Chunkwise curation often completes, but it also loses some chunks, and search and skill writing can overflow. The 32k serving configuration therefore limits these implementations; it does not establish an inherent local/global boundary for filesystem operations. On LoCoMo and REALTALK, the completed condensing passes reduce accuracy more than the unchanged memories left by failed passes. This comparison does not isolate context length from model behavior. The model's native window is reported as 256k in Section B.7.

Table A31: Which findings of Sections 4.1, 4.1.3, 4.2 and E.3 on conversational memory survive a change of the management agent and search agent family. *Holds*: the same reading follows from the family's own cells; *deviates*: a different reading, summarized in the last column with its table or figure. **gem**: `gemma-4-31B-it`; **son**: `claude-sonnet-5`. The embodied and cross-setting findings follow in Table A32.

Finding (section)	gem	son	Deviation and reading
			Conversational memory
With more memorized content the curated memory consolidates rather than shards (4.1)	deviates	deviates	The opposite trend: two files on LoCoMo and REALTALK, then 22 (`gemma-4-31B-it`) and 19 (`claude-sonnet-5`) topic files on PersonaMem 128k, where `gpt-5.4-mini` went from 41 REALTALK files to two (Table A35).
No memory variant is the most accurate on every benchmark (4.2)	holds	holds	The most accurate memory changes with the benchmark for every family (Table A29).
Organizing the memory lowers the cost of searching it (4.2)	partly	holds	`claude-sonnet-5`: reorganized and curated memories read less than the dump on PersonaMem, the curated REALTALK memory the most; `gemma-4-31B-it`: least over its curated LoCoMo memory, more than the dump over its curated PersonaMem 128k and REALTALK memories (26k and 21k against 21k and 18k tokens); the memories its window kept it from editing search like the dump (Table A34).
On the short PersonaMem 32k slices the curated memory falls far below the dump (4.2)	partly	partly	The gap narrows without vanishing: 13 (`gemma-4-31B-it`) and 6 (`claude-sonnet-5`) points against 41 for `gpt-5.4-mini` (Table A33).
Memories grow inside their files and keep the early files (E.3)	partly	partly	Holds on LoCoMo; on PersonaMem 128k both families add files in steps while editing the early ones (Figure A12).
Under the shell harness the memory splits into many files and answer quality does not change (4.1.3)	deviates	deviates	Both families consolidate PersonaMem 128k under the shell (19 to 1 and 22 to 6 files); `claude-sonnet-5`'s shell matches or beats its extended harness on correctness, `gemma-4-31B-it`'s trails (Tables A37 and A38).
Under the shell, sharding the large memory costs extra bytes and build effort but not dollars (4.1.3)	deviates	deviates	Neither family shards, yet the shell build of PersonaMem 128k is the most effortful for both (`claude-sonnet-5` $28 against $18, `gemma-4-31B-it` 19.6M against 10.8M input tokens); `claude-sonnet-5`'s one-file shell memory is the largest (329 KB), `gemma-4-31B-it`'s six-file memory the smallest (54 KB) (Table A38).

What generalizes and what does not. Tables A31 and A32 set the paper's findings against the two families. Within each family the paper's answers survive in outline: no memory organization leads correctness everywhere, organization's measured gain is search cost, the curated chain leads from the first tasks on and the management agent's work per task remains substantial, curated skills are the least expensive memory to use at test time for the priced family, early files survive, and adding tools changes behavior more than outcomes.

The deviations concentrate in two places and each has one reading. First, what a family writes when asked to organize is its own: `gemma-4-31B-it` and `claude-sonnet-5` shard PersonaMem 128k where `gpt-5.4-mini` consolidated, and consolidate under the shell where it sharded, so the organization-level claims of Sections 4.1 and 4.1.3 describe the organizing model, not memory in general.

Second, the execution agent inversion and the shell's lead in the embodied setting are the default family's: under `claude-sonnet-5` as both management agent and search agent, curated skills also help the strong execution agent in both environments, and the tested embodied harnesses do not differ detectably.

`gemma-4-31B-it` adds a third factor, its 32k serving window: it cannot fold or reorganize the large memories at all, and where its condensing pass completed on LoCoMo and REALTALK, its accuracy fell farther below its own dump than `gpt-5.4-mini`'s did.

Several patterns recur: no conversational organization is the most accurate on every benchmark, smaller reads can lower search cost, and early skill files often survive. Other outcomes depend on the model family and setting. These include answer quality, citation support, the cost of a curated REALTALK memory, and the ordering of curated skills and raw episodes. Because writing and retrieval change together, the family comparison does not assign each difference to one role.

Table A32: Continuation of Table A31: the embodied and cross-setting findings under the two families, same columns and reading.

Finding (section)	gem	son	Deviation and reading
Embodied tasks			
A strong execution agent does better with raw episodes, a weak one with curated skills (4.2)	deviates	deviates	The inversion belongs to the weakest management agent: `claude-sonnet-5`'s curated memories beat the episode log at both tiers, `gemma-4-31B-it`'s show no significant difference at `gpt-4.1` and lead at mini (Table A30); on ScienceWorld `claude-sonnet-5`'s lead the log by 14 to 39 points and `gemma-4-31B-it`'s Curated skills+mem (GS) by 10 to 11 (Table A26).
Curated skills are the least expensive memory to use at test time (retrieval plus execution per task) (4.2)	unpriced	holds	Supported for `claude-sonnet-5`; `gemma-4-31B-it` has no deployment dollar rate: the strong execution agent spends about half of its no-memory cost over `claude-sonnet-5`'s memories; on ScienceWorld they are the least expensive per solved task at both tiers (Tables A27 and A36).
Along a chain of tasks, the curated memory beats no memory from the first tasks on and keeps its lead (E.3)	holds	holds	The lead appears within the first fifteen tasks for every family and ends larger, 21 to 26 points at `gpt-4.1` against 11 (Figure A11).
Running one long chain instead of three short ones helps the episode log most at the weak execution agent (E.3)	partly	deviates	`gemma-4-31B-it`'s log gains 9 points at mini, compared with 7 for `gpt-5.4-mini`, but its curated chain gains 12; `claude-sonnet-5`'s log gains 5 at `gpt-4.1` and nothing at mini, its curated chain 5 and 2 (Table A39); as in the paper, the two kinds of runs also differ in build draws.
Early skill files survive to the end of the chain; stronger curation shows as editing in place (E.3)	holds	holds	75 of 76 files present at task 35 survive to the end of the eight replicated chains, 68 edited in place; their final section counts are much smaller than `gpt-5.4-mini`'s, although some chains continue adding sections late and their B4 correlations vary (Figure A16).
Under the shell harness the skill memory consolidates and success rises (4.1.3)	deviates	deviates	The two harnesses have equal aggregate success for `gemma-4-31B-it` (84.3 each) and differ by two tasks for `claude-sonnet-5` (95.7 against 97.1); these runs show smaller harness differences than the `gpt-5.4-mini` comparison (Table A39).
Both settings			
Memory organization is a signature of the model (4.1)	holds	holds	`claude-sonnet-5`'s two conversational files are heavily cross-linked (45 and 34 references), `gemma-4-31B-it`'s two carry almost none; both shard PersonaMem 128k where `gpt-5.4-mini` consolidates; skill libraries of 7 to 19 files against 45 to 137 (Tables A39 and A44).
For the agent-curated memory, the management agent's work per episode does not shrink as the memory grows, so its cumulative cost is linear in the number of tasks (E.3)	holds	holds	On the Curated skills+mem (GS) chains the management agent's output per episode in the last quarter is 0.7 to 1.0 of the first quarter for `claude-sonnet-5` and `gemma-4-31B-it` (1.3 and 0.8 for `gpt-5.4-mini`), and a linear fit of its cumulative cost has $R^2 \geq 0.99$ for all six chains; in the conversational builds its tool calls per chunk stay flat for every family (Figures A11 and A12).

Table A33: Full cross-family memory results (Section F): answer correctness (**Corr**) and citation support (**Attr**), judged %, for the default family (`gpt-5.4-mini`, rows as in Table A9), `gemma-4-31B-it` and `claude-sonnet-5` in the management agent and search agent roles. PersonaMem 32k pools the three terminal slices question-weighted (n=32). †: `gemma-4-31B-it` pass overflowed its 32k window before any edit, so the memory equals the flat dump (‡: two of the three 32k slices); –: the pass made no change (no folders), so no separate search was run. The `gemma-4-31B-it` Reorg. (preserve) and Reorg. (condense) PersonaMem 128k searches and its Reorg. (preserve) 32k slice-4 search each left one question unanswered within the window, counted as wrong in Corr and as zero in Attr.

		LoCoMo		**PersonaMem 32k**		**PersonaMem 128k**		**REALTALK**	
Memory	**Family**	*Corr*	*Attr*	*Corr*	*Attr*	*Corr*	*Attr*	*Corr*	*Attr*
Closed-book	`gpt-5.4-mini`	18.4	10.1	34.4	2.1	47.6	1.6	10.6	12.9
	`gemma-4-31B-it`	0.0	7.8	68.8	2.1	61.9	0.8	0.0	8.4
	`claude-sonnet-5`	4.4	8.0	71.9	4.2	57.1	4.4	0.0	12.2
Chunk retrieval	`gpt-5.4-mini`	81.6	96.0	71.9	93.8	66.7	97.2	71.8	89.4
	`gemma-4-31B-it`	73.4	93.2	75.0	90.6	73.8	94.4	68.2	91.6
	`claude-sonnet-5`	84.2	92.8	75.0	96.4	71.4	92.1	70.6	88.4
Verbatim dump	`gpt-5.4-mini`	84.2	91.2	78.1	95.8	69.0	90.9	77.6	89.8
	`gemma-4-31B-it`	82.3	87.7	68.8	90.1	66.7	77.8	64.7	79.8
	`claude-sonnet-5`	78.5	88.2	71.9	83.3	73.8	90.5	67.1	86.3
Foldered sessions	`gpt-5.4-mini`	86.1	91.1	62.5	96.9	76.2	94.8	77.6	84.3
	`gemma-4-31B-it`	–	–	–	–	–	–	68.2	80.8
	`claude-sonnet-5`	79.1	88.6	62.5	90.1	76.2	96.8	68.2	86.9
Reorg. (preserve)	`gpt-5.4-mini`	82.9	92.8	56.2	93.2	59.5	92.5	77.6	88.6
	`gemma-4-31B-it`	77.8	87.7	68.8†	88.0†	69.0†	79.0†	71.8†	75.3†
	`claude-sonnet-5`	80.4	90.7	71.9	91.1	66.7	93.7	64.7	87.5
Reorg. (condense)	`gpt-5.4-mini`	79.1	90.2	68.8	89.6	54.8	88.9	41.2	82.0
	`gemma-4-31B-it`	59.5	85.1	62.5‡	81.8‡	64.3†	76.6†	22.4	75.1
	`claude-sonnet-5`	69.6	90.2	56.2	89.6	66.7	97.6	63.5	84.9
Agent-curated memory	`gpt-5.4-mini`	86.1	94.4	37.5	97.9	66.7	93.7	75.3	85.1
	`gemma-4-31B-it`	84.2	92.8	56.2	80.7	57.1	79.8	52.9	85.9
	`claude-sonnet-5`	87.3	92.9	65.6	96.9	73.8	93.3	84.7	90.0

Full results. The counterparts of the main tables: Table A33 (correctness and citation support for every memory, benchmark and family); Table A34 (search cost and effort per query, the counterpart of Table A17); Table A35 (management cost, effort and memory profiles, of Table A5); Table A36 (embodied cost by role, of Table A23); Table A40 (execution agent-side absolute computation, of Table A25); Table A37 and Table A38 (the harness axis, its cost and memory organization, of Tables A7 and A16); Table A41 and Table A42 (success by goal family and execution agent-side efficiency per task, of Tables A22 and A24); Table A43 (LoCoMo correctness with the four defective golds removed, of Table A20); Table A44 and Table A45 (the hierarchy shape panel and adherence metrics over the memories each family built, of Tables A6 and A8); Table A39 (every skill cell including the full-chain and harness chains with the final memory file counts); Table A46 (the question-paired sign tests behind the memory readings; the skill readings cite goal-paired sign tests in the text); and the six growth figures named above.

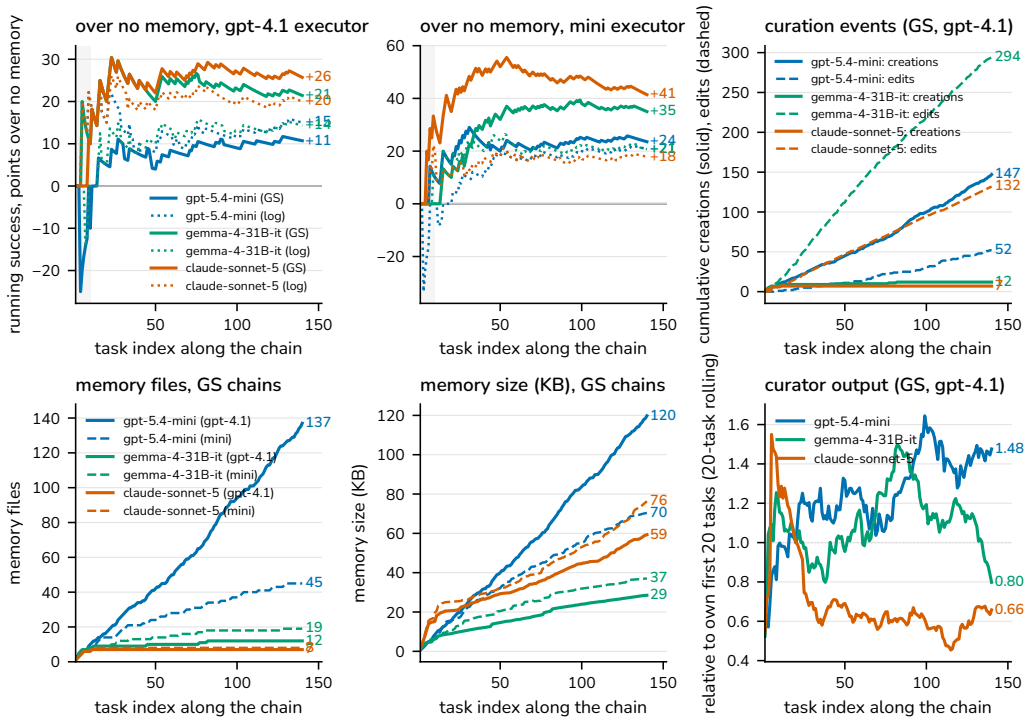


Figure A11: Skill chains across model families (full-chain runs; GS = Curated skills+mem (GS), log = Episode log), the counterpart of Figures A3, A6, A7 and A9 from the run records of the chains of record. Top left and middle: running success minus the original `gpt-5.4-mini` no-memory reference chain's running success at the same position of the fixed order, per execution agent tier. Every GS chain takes its lead within the first fifteen tasks; with `gemma-4-31B-it` or `claude-sonnet-5` curating it holds 21 to 26 points at the `gpt-4.1` tier, where `gpt-5.4-mini`'s GS chain trails its own episode log (11 against 15), and at the mini tier the GS chains end 35 (`gemma-4-31B-it`) and 41 (`claude-sonnet-5`) points over no memory against 24 for `gpt-5.4-mini`. Top right: cumulative curation events of the `gpt-4.1`-tier GS chains; `gpt-5.4-mini` creates 147 files and edits 52 times, `gemma-4-31B-it` creates 12 and edits 294 times, `claude-sonnet-5` creates 7 and edits 132 times, so the stronger management agents are edit-dominated from the first tasks. Bottom left and middle: memory files and size; `gpt-5.4-mini`'s memories grow to 137 and 45 files while `gemma-4-31B-it`'s and `claude-sonnet-5`'s stop at 7 to 19 files and keep growing in bytes. Bottom right: management agent output per episode relative to the family's own first twenty tasks (20-task rolling; token counts are not compared across families): `claude-sonnet-5`'s falls to about 0.7 of its first-twenty mean by task 30 and moves between 0.45 and 0.7 afterwards, `gemma-4-31B-it`'s drifts down in the last quarter, `gpt-5.4-mini`'s rises; for no family does the effort per episode shrink away away.

Table A34: Search cost and effort per query across model families, the counterpart of Table A17 for `claude-sonnet-5` and `gemma-4-31B-it` as the search agent over the memories each family built. `claude-sonnet-5`: *Un* / *Ca* / *Out* = uncached-input / cached-input / output (reasoning included) tokens in thousands, *Rsn%* = share of output that is reasoning, *Rd* / *TC* = mean tool rounds and tool calls, *Cost* = billed cents per query as logged by the serving proxy at the deployed rates ($2.20 / $0.22 / $11.00 per million uncached-input / cached-input / output tokens, and $2.75 per million for the uncached input written to the prompt cache, which is nearly all of it). `gemma-4-31B-it`: served locally, so no rate applies and no cached-input sub-count is reported; *In* = all input tokens and *Out* = output tokens including thinking, both in thousands. Token counts are not comparable across families (Section F). PersonaMem 32k rows pool the three slices question-weighted. †: the `gemma-4-31B-it` pass overflowed its window before any edit, so the searched memory is the flat dump (‡: two of the three 32k slices); –: no memory produced.

	`claude-sonnet-5`							**`gemma-4-31B-it`**			
Memory	*Un*	*Ca*	*Out*	*Rsn%*	*Rd*	*TC*	*Cost (¢)*	*In*	*Out*	*Rd*	*TC*
LoCoMo											
No memory											
Closed-book	0.2	0.0	0.1	0	1.0	0.0	0.1	0.2	0.3	1.0	0.0
Chunk retrieval	4.3	7.9	0.3	15	2.4	2.0	1.7	6.9	0.7	2.5	1.5
Files											
Verbatim dump	7.0	19.6	0.5	23	3.3	3.2	2.9	19.0	1.2	3.8	2.9
Foldered sessions	6.9	24.2	0.6	23	3.7	3.0	3.0	–	–	–	–
Reorg. (preserve)	4.4	18.5	0.5	23	3.3	2.6	2.1	19.2	1.2	4.0	3.4
Reorg. (condense)	3.4	15.5	0.5	20	3.3	2.6	1.8	17.3	1.4	5.2	4.3
Agent-curated memory	8.5	19.9	0.5	21	3.1	2.2	3.3	12.5	1.0	3.7	2.7
PersonaMem 32k											
No memory											
Closed-book	0.8	0.0	0.0	0	1.0	0.0	0.2	0.5	1.1	1.0	0.0
Chunk retrieval	7.4	7.1	0.6	31	2.3	2.3	2.8	5.3	1.5	2.0	1.0
Files											
Verbatim dump	11.6	13.0	0.7	40	2.7	2.1	4.1	16.8	2.1	3.1	2.1
Foldered sessions	11.3	10.4	0.7	43	2.2	1.3	4.0	–	–	–	–
Reorg. (preserve)	4.5	11.6	0.7	42	2.7	1.9	2.2	17.1†	2.0	3.1	2.1
Reorg. (condense)	3.7	12.1	0.6	37	2.8	1.9	2.0	18.4‡	2.5	4.0	3.2
Agent-curated memory	6.0	13.2	0.7	39	2.8	1.9	2.7	15.4	2.2	3.6	2.8
PersonaMem 128k											
No memory											
Closed-book	0.7	0.0	0.0	0	1.0	0.0	0.2	0.5	1.1	1.0	0.0
Chunk retrieval	5.9	6.7	0.5	29	2.2	2.0	2.4	5.2	1.4	2.0	1.0
Files											
Verbatim dump	10.8	12.4	0.6	35	2.6	2.0	3.9	20.9	2.2	3.3	2.3
Foldered sessions	10.9	12.3	0.6	36	2.2	1.3	3.9	–	–	–	–
Reorg. (preserve)	3.6	11.6	0.6	35	2.7	1.7	1.9	19.9†	2.3	3.3	2.3
Reorg. (condense)	3.0	12.6	0.7	43	2.8	1.9	1.9	19.6†	2.1	3.3	2.3
Agent-curated memory	8.5	12.5	0.6	38	2.3	1.4	3.2	25.8	3.0	4.5	3.8
REALTALK											
No memory											
Closed-book	0.2	0.0	0.1	0	1.0	0.0	0.2	0.2	0.2	1.0	0.0
Chunk retrieval	5.0	9.6	0.6	17	2.6	2.5	2.3	6.2	0.8	2.4	1.5
Files											
Verbatim dump	7.1	20.3	0.8	18	3.6	3.8	3.3	17.9	1.6	3.9	3.0
Foldered sessions	6.4	24.7	0.8	19	3.9	3.5	3.2	18.8	1.6	4.1	3.3
Reorg. (preserve)	7.9	24.0	0.8	16	3.6	2.8	3.6	16.1†	1.4	3.9	2.9
Reorg. (condense)	8.0	29.0	0.9	18	4.3	3.7	3.9	29.6	2.8	6.7	6.4
Agent-curated memory	18.0	26.7	0.8	15	3.1	2.4	6.4	20.6	2.2	5.1	4.6

Table A35: Management cost and effort per conversation across model families, the counterpart of Table A5: *Un* / *Ca* / *Out* in millions of tokens, total tool calls (*TC*), dollars (*Cost*) and the memory profile (directories holding files, files, markdown sections, kilobytes) of the memory each family built. `claude-sonnet-5`: output including reasoning, billed dollars at its deployed rates; `gemma-4-31B-it`: all input tokens under *Un* (no cached-input sub-count), output including thinking, no rate. Token columns count each pass conversation once plus the compaction summarizer's calls. The Verbatim dump rows of Table A5 apply unchanged (no agent builds them). PersonaMem 32k values are means over its three conversations. †: every pass overflowed the `gemma-4-31B-it` window before an edit, so the memory equals the flat dump (‡: two of the three 32k conversations); –: not reported (`gemma-4-31B-it`'s cached-input sub-count and rate) or no memory produced (`gemma-4-31B-it` foldering completes only on REALTALK).

Memory	**Family**	*Un*	*Ca*	*Out*	*TC*	*Cost ($)*	*Dirs*	*Files*	*Sec.*	*KB*
					LoCoMo					
Foldered sessions	`claude-sonnet-5`	0.06	0.24	0.01	62	0.34	4	30	30	102
	`gemma-4-31B-it`	–	–	–	–	–	–	–	–	–
Reorg. (preserve)	`claude-sonnet-5`	0.13	6.50	0.06	121	2.50	3	12	51	27
	`gemma-4-31B-it`	0.09	–	0.01	31	–	3	37	42	105
Reorg. (condense)	`claude-sonnet-5`	0.11	4.57	0.04	92	1.80	2	5	33	16
	`gemma-4-31B-it`	1.68	–	0.04	153	–	2	3	10	11
Agent-curated memory	`claude-sonnet-5`	1.41	8.95	0.22	619	8.23	1	2	73	103
	`gemma-4-31B-it`	3.94	–	0.36	543	–	1	2	23	40
					PersonaMem 32k					
Foldered sessions	`claude-sonnet-5`	0.05	0.06	0.00	10	0.16	2.3	4.7	5	137
	`gemma-4-31B-it`	–	–	–	–	–	–	–	–	–
Reorg. (preserve)	`claude-sonnet-5`	0.09	2.59	0.04	56	1.24	1.3	7.7	38	20
	`gemma-4-31B-it`	0.03	–	0.00	5	–	0†	4.7	5	137
Reorg. (condense)	`claude-sonnet-5`	0.08	1.91	0.03	55	0.97	1.3	7	36	16
	`gemma-4-31B-it`	0.19	–	0.01	19	–	0.7‡	6.3	6	95
Agent-curated memory	`claude-sonnet-5`	0.43	3.05	0.08	295	2.78	1	11.3	54	60
	`gemma-4-31B-it`	2.54	–	0.25	314	–	1	3.7	19	25
					PersonaMem 128k					
Foldered sessions	`claude-sonnet-5`	0.09	0.30	0.01	52	0.40	10	20	114	612
	`gemma-4-31B-it`	–	–	–	–	–	–	–	–	–
Reorg. (preserve)	`claude-sonnet-5`	0.30	21.48	0.07	114	6.34	2	13	66	51
	`gemma-4-31B-it`	0.01	–	0.00	6	–	0†	20	114	612
Reorg. (condense)	`claude-sonnet-5`	0.36	11.01	0.07	131	4.15	2	15	70	41
	`gemma-4-31B-it`	0.06	–	0.01	5	–	0†	20	114	612
Agent-curated memory	`claude-sonnet-5`	3.33	19.18	0.38	1336	17.61	1	19	211	281
	`gemma-4-31B-it`	10.80	–	1.05	1427	–	5	22	125	135
					REALTALK					
Foldered sessions	`claude-sonnet-5`	0.05	0.22	0.01	42	0.29	5	20	20	105
	`gemma-4-31B-it`	0.06	–	0.00	41	–	4	20	20	105
Reorg. (preserve)	`claude-sonnet-5`	0.14	3.94	0.05	67	1.74	1	8	50	51
	`gemma-4-31B-it`	0.31	–	0.01	21	–	0†	20	20	105
Reorg. (condense)	`claude-sonnet-5`	0.14	4.71	0.05	102	1.95	3	10	59	37
	`gemma-4-31B-it`	1.81	–	0.07	129	–	2	8	23	24
Agent-curated memory	`claude-sonnet-5`	1.96	11.65	0.23	744	10.50	0	2	51	125
	`gemma-4-31B-it`	4.00	–	0.44	600	–	1	2	16	58

Table A36: Embodied setting, three-chain runs, across model families: build and deployment cost by role over the 140 tasks, the counterpart of Table A23. `claude-sonnet-5` rows: *Build* (management agent) and *Search* (retriever) are its billed dollars, apportioned to the two roles by their rate-card cost; *Exec* is the execution agent's dollars at the rates of Table A23; *Deploy* = Search + Exec; cents per task and per solved task; *Succ* = success (%) over 140 tasks as in Table A30. `gemma-4-31B-it` rows: the management agent and search agent run on a self-served model with no rate, so *Build* and *Search* give their token volume in millions (input plus output, thinking included) and the dollar columns cover the execution agent only. –: no such role (No memory) or no rate (`gemma-4-31B-it`'s deployment dollars).

	Family	*Build*	*Search*	*Exec $*	*Deploy $*	*¢/task*	*¢/succ*	*Succ*
Execution agent `gpt-4.1`								
No memory	`claude-sonnet-5`	–	–	16.86	16.86	12.0	15.9	75.7
	`gemma-4-31B-it`	–	–	16.57	16.57	11.8	15.3	77.1
Episode log	`claude-sonnet-5`	0	3.79	11.70	15.49	11.1	12.5	88.6
	`gemma-4-31B-it`	0	2.4M	10.89	–	–	–	90.0
Curated skills	`claude-sonnet-5`	4.54	0.87	7.80	8.67	6.2	6.5	95.7
	`gemma-4-31B-it`	4.8M	1.2M	11.38	–	–	–	91.4
Curated skills+mem	`claude-sonnet-5`	5.33	1.17	7.96	9.14	6.5	6.9	94.3
	`gemma-4-31B-it`	5.6M	1.9M	12.67	–	–	–	85.7
Curated skills+mem (GS)	`claude-sonnet-5`	6.74	3.51	7.12	10.63	7.6	8.1	94.3
	`gemma-4-31B-it`	4.7M	1.5M	8.63	–	–	–	93.6
Execution agent `gpt-4.1-mini`								
No memory	`claude-sonnet-5`	–	–	4.35	4.35	3.1	5.4	57.9
	`gemma-4-31B-it`	–	–	4.43	4.43	3.2	5.8	55.0
Episode log	`claude-sonnet-5`	0	4.45	4.12	8.57	6.1	8.6	71.4
	`gemma-4-31B-it`	0	3.0M	4.15	–	–	–	64.3
Curated skills	`claude-sonnet-5`	6.24	1.62	2.69	4.31	3.1	3.5	87.1
	`gemma-4-31B-it`	5.4M	1.3M	3.45	–	–	–	74.3
Curated skills+mem	`claude-sonnet-5`	7.76	1.76	2.05	3.81	2.7	3.1	89.3
	`gemma-4-31B-it`	6.3M	2.2M	3.31	–	–	–	80.0
Curated skills+mem (GS)	`claude-sonnet-5`	7.25	3.64	1.44	5.08	3.6	3.9	92.1
	`gemma-4-31B-it`	6.5M	1.8M	2.96	–	–	–	76.4

Table A37: Harness axis across model families (Section F): answer quality of the Agent-curated memory under the three harnesses of Table A16, with the management agent and search agent run by each family. Corr and Attr as in Table A9. The `gemma-4-31B-it` builds behind these searches lost chunks to its serving window (counts in the caption of Table A38).

		LoCoMo		**PersonaMem 128k**	
Harness	**Family**	*Corr*	*Attr*	*Corr*	*Attr*
Center (default tools)	`gpt-5.4-mini`	86.1	94.4	66.7	93.7
	`gemma-4-31B-it`	84.2	92.8	57.1	79.8
	`claude-sonnet-5`	87.3	92.9	73.8	93.3
Center+BM25	`gpt-5.4-mini`	86.7	94.9	64.3	91.3
	`gemma-4-31B-it`	76.6	93.6	69.0	87.7
	`claude-sonnet-5`	88.0	83.9	66.7	88.1
Shell (bash)	`gpt-5.4-mini`	86.1	93.1	61.9	92.1
	`gemma-4-31B-it`	74.7	90.9	50.0	84.1
	`claude-sonnet-5`	89.9	77.6	73.8	83.3

Table A38: Harness axis across model families, the counterpart of Table A7: search cost and effort per query, management cost and the organization of the memory each harness builds, with the management agent and search agent run by `claude-sonnet-5` or `gemma-4-31B-it` (answer quality in Table A37). Search columns as in Table A34 (`claude-sonnet-5`: billed cents; `gemma-4-31B-it`: all input under *Un*, output including thinking, no rate); *Build* = `claude-sonnet-5`'s billed dollars, or `gemma-4-31B-it`'s input tokens in millions, and *TC* = total build tool calls; *Files* / *KB* = the memory's file count and size. The Center rows are the Agent-curated memory cells of Tables A34 and A35. `gemma-4-31B-it` builds that lost chunks to its window: Center LoCoMo 83 of 85 and PersonaMem 128k 270 of 271, Center+BM25 LoCoMo 84 of 85 and PersonaMem 128k 260 of 271, Shell LoCoMo 82 of 85 and PersonaMem 128k 231 of 271. –: not reported for `gemma-4-31B-it` (cached-input sub-count, reasoning share, rate). The `claude-sonnet-5` management cost include replayed calls for Shell PersonaMem 128k (its bundle bills the live calls only, $25.9; the 591 calls replayed from three outage-killed attempts add $2.4 at the card).

		Search / query							**Build**		**Memory**	
Harness	**Family**	*Un*	*Ca*	*Out*	*Rsn%*	*Rd*	*TC*	*¢*	*$ or M*	*TC*	*Files*	*KB*
					LoCoMo							
Center	`claude-sonnet-5`	8.5	19.9	0.5	21	3.1	2.2	3.3	8.2	619	2	103
	`gemma-4-31B-it`	12.5	–	1.0	–	3.7	2.7	–	3.9M	543	2	40
Center+BM25	`claude-sonnet-5`	9.6	18.5	0.4	18	2.8	1.9	3.5	8.6	758	2	117
	`gemma-4-31B-it`	14.7	–	1.0	–	3.9	2.9	–	3.8M	547	3	34
Shell	`claude-sonnet-5`	7.3	10.0	0.4	13	2.8	1.8	2.7	7.8	525	2	83
	`gemma-4-31B-it`	16.8	–	1.2	–	3.9	2.9	–	5.0M	471	2	30
					PersonaMem 128k							
Center	`claude-sonnet-5`	8.5	12.5	0.6	38	2.3	1.4	3.2	17.6	1336	19	281
	`gemma-4-31B-it`	25.8	–	3.0	–	4.5	3.8	–	10.8M	1427	22	135
Center+BM25	`claude-sonnet-5`	14.7	15.1	0.7	43	2.4	1.6	5.1	19.2	1437	12	269
	`gemma-4-31B-it`	19.6	–	2.5	–	3.7	3.0	–	14.3M	1490	25	174
Shell	`claude-sonnet-5`	17.1	18.7	0.7	34	3.0	2.0	5.9	28.2	1656	1	329
	`gemma-4-31B-it`	24.3	–	2.8	–	3.9	2.9	–	19.6M	1546	6	54

Table A39: Full cross-family skill results on ALFWorld (Section F): success over the 140 tasks (%) for the default family (`gpt-5.4-mini`), `gemma-4-31B-it` and `claude-sonnet-5` as management agent and retriever; execution agent tiers as in Table A11; **5.4m**: `gpt-5.4-mini`, **gem**: `gemma-4-31B-it`, **son**: `claude-sonnet-5`. Upper block: three-chain runs. Lower block: single-chain runs and the harness axis (Curated skills+mem (GS), execution agent `gpt-4.1-mini`), with the final memory file count in parentheses where recorded. –: not run at that tier. The last block reports two cells outside the design of Section 4.1.3, the harness chains run with the `gpt-4.1` execution agent.

	Execution agent `gpt-4.1`			**Execution agent `gpt-4.1-mini`**		
Memory	*5.4m*	*gem*	*son*	*5.4m*	*gem*	*son*
Three-chain runs						
No memory	73.6	77.1	75.7	52.9	55.0	57.9
Episode log	87.1	90.0	88.6	66.4	64.3	71.4
Curated skills	80.7	91.4	95.7	65.7	74.3	87.1
Curated skills+mem	75.7	85.7	94.3	55.0	80.0	89.3
Curated skills+mem (GS)	82.1	93.6	94.3	76.4	76.4	92.1
Single-chain runs						
Episode log	88.6 (140)	87.9 (140)	93.6 (140)	73.6 (140)	73.6 (140)	70.7 (140)
Curated skills+mem (GS)	84.3 (137)	95.0 (12)	99.3 (7)	76.4 (45)	87.9 (19)	94.3 (8)
Harness (Curated skills+mem (GS), full chain, execution agent `gpt-4.1-mini`)						
Center+BM25	–	–	–	75.0 (76)	84.3 (14)	95.7 (10)
Shell	–	–	–	82.9 (36)	84.3 (16)	97.1 (7)
Harness, full chain, execution agent `gpt-4.1` (extra, `claude-sonnet-5` only)						
Center+BM25	–	–	97.1 (6)	–	–	–
Shell	–	–	97.1 (6)	–	–	–

Table A40: Embodied setting, three-chain runs, across model families: absolute execution agent-side computation in millions of tokens, the counterpart of Table A25 (same definitions: *Perfect* = each token prefilled once and each output decoded once, *Measured* = billed uncached prefill plus decode, *No-cache* = every turn's full prefix re-prefilled, *Eff* = share of the achievable caching savings realized). The execution agent is the same model in every row of a tier; only the memory it reads, written by the named family, differs. Comparisons are within one execution agent. –: a resumed cell whose replayed tasks carry no per-turn usage.

	Family	*Perfect*	*Measured*	*No-cache*	*Meas/Perf*	*Eff*
Execution agent `gpt-4.1`						
No memory	`claude-sonnet-5`	–	–	–	–	–
	`gemma-4-31B-it`	1.13	2.37	21.87	2.10×	94.0%
Episode log	`claude-sonnet-5`	0.97	1.70	15.15	1.75×	94.9%
	`gemma-4-31B-it`	0.90	1.56	13.99	1.73×	95.0%
Curated skills	`claude-sonnet-5`	0.85	1.33	9.53	1.55×	94.6%
	`gemma-4-31B-it`	0.96	1.67	14.93	1.75×	94.9%
Curated skills+mem	`claude-sonnet-5`	0.78	1.33	9.64	1.71×	93.8%
	`gemma-4-31B-it`	1.00	1.72	17.09	1.72×	95.5%
Curated skills+mem (GS)	`claude-sonnet-5`	0.66	1.22	8.59	1.84×	93.0%
	`gemma-4-31B-it`	0.73	1.44	10.50	1.96×	92.8%
Execution agent `gpt-4.1-mini`						
No memory	`claude-sonnet-5`	1.38	2.79	30.92	2.02×	95.2%
	`gemma-4-31B-it`	1.42	2.63	32.10	1.86×	96.0%
Episode log	`claude-sonnet-5`	1.39	2.38	30.07	1.70×	96.6%
	`gemma-4-31B-it`	1.33	2.98	27.87	2.24×	93.8%
Curated skills	`claude-sonnet-5`	1.13	1.80	18.78	1.59×	96.2%
	`gemma-4-31B-it`	1.24	2.32	23.00	1.87×	95.0%
Curated skills+mem	`claude-sonnet-5`	0.92	1.57	13.61	1.72×	94.8%
	`gemma-4-31B-it`	1.20	2.27	22.29	1.89×	94.9%
Curated skills+mem (GS)	`claude-sonnet-5`	0.67	1.21	9.01	1.81×	93.5%
	`gemma-4-31B-it`	1.04	1.97	20.47	1.88×	95.2%

Table A41: Embodied setting across model families: success (%) by ALFWorld goal family, the counterpart of Table A22 (three-chain runs; family sizes as there; *All* = the 140-task average of Table A30) and of the family columns of Table A21 (single-chain runs and harness chains). *Files* = final file count, summed over the three chains in the three-chain rows. The No memory rows contain no replicated role and show the run-to-run floor (–: no memory).

	Family	*Place*	*Two*	*Look*	*Clean*	*Heat*	*Cool*	*All*	*Files*
Three-chain runs, execution agent `gpt-4.1`									
No memory	`claude-sonnet-5`	100.0	91.7	92.3	37.0	56.2	72.0	75.7	–
	`gemma-4-31B-it`	100.0	95.8	92.3	40.7	62.5	68.0	77.1	–
Episode log	`claude-sonnet-5`	100.0	100.0	100.0	92.6	31.2	88.0	88.6	140
	`gemma-4-31B-it`	100.0	100.0	92.3	81.5	62.5	92.0	90.0	140
Curated skills	`claude-sonnet-5`	100.0	95.8	100.0	96.3	87.5	92.0	95.7	9
	`gemma-4-31B-it`	100.0	95.8	100.0	88.9	68.8	88.0	91.4	20
Curated skills+mem	`claude-sonnet-5`	100.0	95.8	100.0	88.9	87.5	92.0	94.3	10
	`gemma-4-31B-it`	100.0	95.8	92.3	81.5	43.8	84.0	85.7	24
Curated skills+mem (GS)	`claude-sonnet-5`	100.0	100.0	100.0	96.3	75.0	88.0	94.3	14
	`gemma-4-31B-it`	100.0	95.8	100.0	96.3	81.2	84.0	93.6	23
Three-chain runs, execution agent `gpt-4.1-mini`									
No memory	`claude-sonnet-5`	100.0	87.5	61.5	40.7	12.5	16.0	57.9	–
	`gemma-4-31B-it`	94.3	79.2	61.5	37.0	18.8	16.0	55.0	–
Episode log	`claude-sonnet-5`	91.4	75.0	46.2	77.8	31.2	72.0	71.4	140
	`gemma-4-31B-it`	97.1	66.7	46.2	77.8	37.5	28.0	64.3	140
Curated skills	`claude-sonnet-5`	100.0	87.5	76.9	100.0	62.5	76.0	87.1	17
	`gemma-4-31B-it`	97.1	87.5	69.2	88.9	18.8	52.0	74.3	29
Curated skills+mem	`claude-sonnet-5`	94.3	95.8	69.2	96.3	68.8	92.0	89.3	13
	`gemma-4-31B-it`	94.3	87.5	76.9	77.8	43.8	80.0	80.0	42
Curated skills+mem (GS)	`claude-sonnet-5`	100.0	91.7	100.0	96.3	75.0	84.0	92.1	14
	`gemma-4-31B-it`	100.0	91.7	84.6	88.9	25.0	44.0	76.4	29
Single-chain runs, execution agent `gpt-4.1`									
Episode log	`claude-sonnet-5`	100.0	100.0	100.0	92.6	62.5	96.0	93.6	140
	`gemma-4-31B-it`	100.0	100.0	84.6	92.6	31.2	92.0	87.9	140
Curated skills+mem (GS)	`claude-sonnet-5`	100.0	100.0	100.0	100.0	93.8	100.0	99.3	7
	`gemma-4-31B-it`	100.0	100.0	100.0	88.9	87.5	92.0	95.0	12
Single-chain runs, execution agent `gpt-4.1-mini`									
Episode log	`claude-sonnet-5`	97.1	75.0	76.9	88.9	6.2	48.0	70.7	140
	`gemma-4-31B-it`	97.1	79.2	53.8	74.1	43.8	64.0	73.6	140
Curated skills+mem (GS)	`claude-sonnet-5`	100.0	95.8	100.0	92.6	100.0	80.0	94.3	8
	`gemma-4-31B-it`	97.1	91.7	92.3	96.3	81.2	64.0	87.9	19
Harness chains (Curated skills+mem (GS), full chain, execution agent `gpt-4.1-mini`*)*									
Center+BM25	`claude-sonnet-5`	100.0	95.8	92.3	96.3	87.5	96.0	95.7	10
	`gemma-4-31B-it`	97.1	95.8	76.9	92.6	50.0	72.0	84.3	14
Shell	`claude-sonnet-5`	100.0	95.8	92.3	100.0	93.8	96.0	97.1	7
	`gemma-4-31B-it`	100.0	91.7	92.3	85.2	62.5	64.0	84.3	16

Table A42: Embodied setting, three-chain runs, across model families: execution agent-side efficiency per task, the counterpart of Table A24 (*Rounds* = environment steps; *Un*, *Ca*, *Out* = uncached-input, cached-input and output tokens in thousands per task; ¢ = execution agent cost per task at the Table A23 rates). The execution agent is the same model within a tier; the memory it reads was written by the named family.

	Family	*Rounds*	*Un*	*Ca*	*Out*	¢
Execution agent `gpt-4.1`						
No memory	`claude-sonnet-5`	24.8	14.8	141.1	2.5	12.0
	`gemma-4-31B-it`	24.8	14.5	139.2	2.5	11.8
Episode log	`claude-sonnet-5`	16.1	10.3	96.0	1.9	8.4
	`gemma-4-31B-it`	17.4	9.3	88.8	1.9	7.8
Curated skills	`claude-sonnet-5`	12.1	8.2	58.6	1.2	5.6
	`gemma-4-31B-it`	17.6	10.2	94.7	1.7	8.1
Curated skills+mem	`claude-sonnet-5`	13.3	8.1	59.4	1.4	5.7
	`gemma-4-31B-it`	19.3	10.5	109.8	1.8	9.0
Curated skills+mem (GS)	`claude-sonnet-5`	12.7	7.5	52.7	1.2	5.1
	`gemma-4-31B-it`	15.4	8.8	64.7	1.4	6.2
Execution agent `gpt-4.1-mini`						
No memory	`claude-sonnet-5`	31.1	17.5	200.9	2.5	3.1
	`gemma-4-31B-it`	31.5	16.3	210.5	2.5	3.2
Episode log	`claude-sonnet-5`	24.3	14.6	197.8	2.4	2.9
	`gemma-4-31B-it`	25.9	18.5	177.8	2.8	3.0
Curated skills	`claude-sonnet-5`	18.3	11.2	121.3	1.6	1.9
	`gemma-4-31B-it`	23.6	13.9	147.8	2.7	2.5
Curated skills+mem	`claude-sonnet-5`	15.9	9.9	86.0	1.3	1.5
	`gemma-4-31B-it`	22.1	13.8	143.0	2.4	2.4
Curated skills+mem (GS)	`claude-sonnet-5`	13.4	7.6	55.7	1.0	1.0
	`gemma-4-31B-it`	22.1	12.1	132.1	1.9	2.1

Table A43: LoCoMo correctness across model families with the four defective golds removed (Section E.2.9), the counterpart of Table A20: the headline score over all 158 questions against the same judge outcomes restricted to the 154 sound ones. –: no memory produced.

	`claude-sonnet-5`		**`gemma-4-31B-it`**	
Memory	*158 Q*	*154 Q*	*158 Q*	*154 Q*
Closed-book	4.4	4.5	0.0	0.0
Chunk retrieval	84.2	85.1	73.4	74.7
Verbatim dump	78.5	79.2	82.3	83.1
Foldered sessions	79.1	79.2	–	–
Reorg. (preserve)	80.4	81.8	77.8	78.6
Reorg. (condense)	69.6	70.8	59.5	61.0
Agent-curated memory	87.3	88.3	84.2	85.7

Table A44: Hierarchy shape panel across model families over the memories of record, the counterpart of Table A6 (same metrics: directories, files, sections; mean/max leaf depth; mean/max fan-out; in-memory cross-references; size coefficient of variation; *B4*, the distance-mirrors-relatedness correlation). Skill memories are the full-chain Curated skills+mem (GS) memories and the harness chains' memories. *X-ref* counts the lowercase `see /memories/` pointers, as in Table A6. *Dirs* counts every directory node of the memory, where Table A35 counts directories holding files (they differ by one for the `gemma-4-31B-it` PersonaMem 128k memory, which has one directory holding only subdirectories).

Memory	**Family**	*Dirs*	*Files*	*Secs*	*Depth*	*Fan*	*X-ref*	*CV*	*B4*
Conversational (Agent-curated memory)									
LoCoMo	`claude-sonnet-5`	1	2	73	4.2/5	10.9/32	45	0.64	.03
LoCoMo	`gemma-4-31B-it`	1	2	23	3.7/4	3.7/6	2	1.37	.04
REALTALK	`claude-sonnet-5`	0	2	51	2.3/3	6.6/20	34	1.15	-.01
REALTALK	`gemma-4-31B-it`	1	2	16	3.0/3	4.8/8	0	2.03	.02
PersonaMem 128k	`claude-sonnet-5`	1	19	211	2.6/4	9.6/38	7	0.72	.21
PersonaMem 128k	`gemma-4-31B-it`	6	22	125	4.8/6	3.1/15	18	0.95	.40
Skill memories (single-chain runs)									
GS, `gpt-4.1` exec.	`claude-sonnet-5`	0	7	16	3.0/3	2.7/7	10	0.87	.27
GS, `mini` exec.	`claude-sonnet-5`	0	8	18	3.0/3	2.7/8	9	1.20	.18
GS, `gpt-4.1` exec.	`gemma-4-31B-it`	0	12	29	3.1/4	2.8/12	3	1.73	.30
GS, `mini` exec.	`gemma-4-31B-it`	0	19	50	3.0/3	3.3/19	7	1.14	.21
Center+BM25 (mini)	`claude-sonnet-5`	1	10	20	4.0/4	2.5/10	6	1.30	.23
Shell (mini)	`claude-sonnet-5`	1	7	48	4.3/5	5.2/20	6	0.60	.29
Center+BM25 (mini)	`gemma-4-31B-it`	1	14	84	4.0/4	5.6/51	2	0.76	.66
Shell (mini)	`gemma-4-31B-it`	1	16	42	4.5/5	2.1/14	1	0.72	.16

Table A45: Full hierarchy panel across model families, the counterpart of Table A8: content units, bytes and the taxonomy-contract adherence metrics of Section A.2 (*B1* sibling label distance mean/min, *B2* sibling content cohesion lift, *B3* leak fraction, *B4* as in Table A44) over the memories of Table A44. A dash would mark an undefined value (too few units or groups).

Memory	**Family**	*Units*	*Bytes*	*B1 mean/min*	*B2 lift*	*B3 leak*	*B4*
Conversational (Agent-curated memory)							
LoCoMo	`claude-sonnet-5`	70	105k	.78/.23	0.90	.043	.031
LoCoMo	`gemma-4-31B-it`	20	41k	.93/.59	0.78	.000	.036
REALTALK	`claude-sonnet-5`	46	128k	.91/.49	1.22	.000	-.007
REALTALK	`gemma-4-31B-it`	16	59k	.89/.68	0.87	.000	.017
PersonaMem 128k	`claude-sonnet-5`	208	288k	.91/.36	2.51	.005	.205
PersonaMem 128k	`gemma-4-31B-it`	104	138k	.96/.52	3.08	.000	.401
Skill memories (single-chain runs)							
GS, `gpt-4.1` exec.	`claude-sonnet-5`	16	61k	.96/.33	1.74	.000	.274
GS, `mini` exec.	`claude-sonnet-5`	18	78k	.97/.10	1.59	.000	.185
GS, `gpt-4.1` exec.	`gemma-4-31B-it`	28	29k	.99/.60	2.99	.000	.302
GS, `mini` exec.	`gemma-4-31B-it`	50	38k	.99/.17	2.04	.000	.214
Center+BM25 (mini)	`claude-sonnet-5`	20	73k	.98/.29	1.57	.000	.227
Shell (mini)	`claude-sonnet-5`	47	68k	.94/.33	1.52	.000	.294
Center+BM25 (mini)	`gemma-4-31B-it`	83	42k	.99/.50	2.80	.000	.657
Shell (mini)	`gemma-4-31B-it`	33	39k	.96/.48	2.59	.000	.163

Table A46: Question-paired sign tests behind the cross-family memory readings (Section F): wins/losses/ties on per-question correctness and the two-sided exact binomial p (ties dropped). Left: the Agent-curated memory against the Verbatim dump within the same family. Right: the family against `gpt-5.4-mini` on the same memory. PersonaMem 32k pools the three slices. **mini**: `gpt-5.4-mini`.

Benchmark	**Family**	*Curated vs dump W/L/T (p)*	*Dump vs mini W/L/T (p)*	*Curated vs mini W/L/T (p)*
LoCoMo	`claude-sonnet-5`	20/6/132 (0.01)	7/16/135 (0.09)	11/9/138 (0.82)
	`gemma-4-31B-it`	12/9/137 (0.66)	8/11/139 (0.65)	10/13/135 (0.68)
PersonaMem 32k	`claude-sonnet-5`	2/4/26 (0.69)	2/4/26 (0.69)	10/1/21 (0.01)
	`gemma-4-31B-it`	1/5/26 (0.22)	2/5/25 (0.45)	11/5/16 (0.21)
PersonaMem 128k	`claude-sonnet-5`	4/4/34 (1.00)	6/4/32 (0.75)	6/3/33 (0.51)
	`gemma-4-31B-it`	5/9/28 (0.42)	2/3/37 (1.00)	7/11/24 (0.48)
REALTALK	`claude-sonnet-5`	18/3/64 (<0.01)	5/14/66 (0.06)	9/1/75 (0.02)
	`gemma-4-31B-it`	10/20/55 (0.10)	3/14/68 (0.01)	2/21/62 (<0.01)

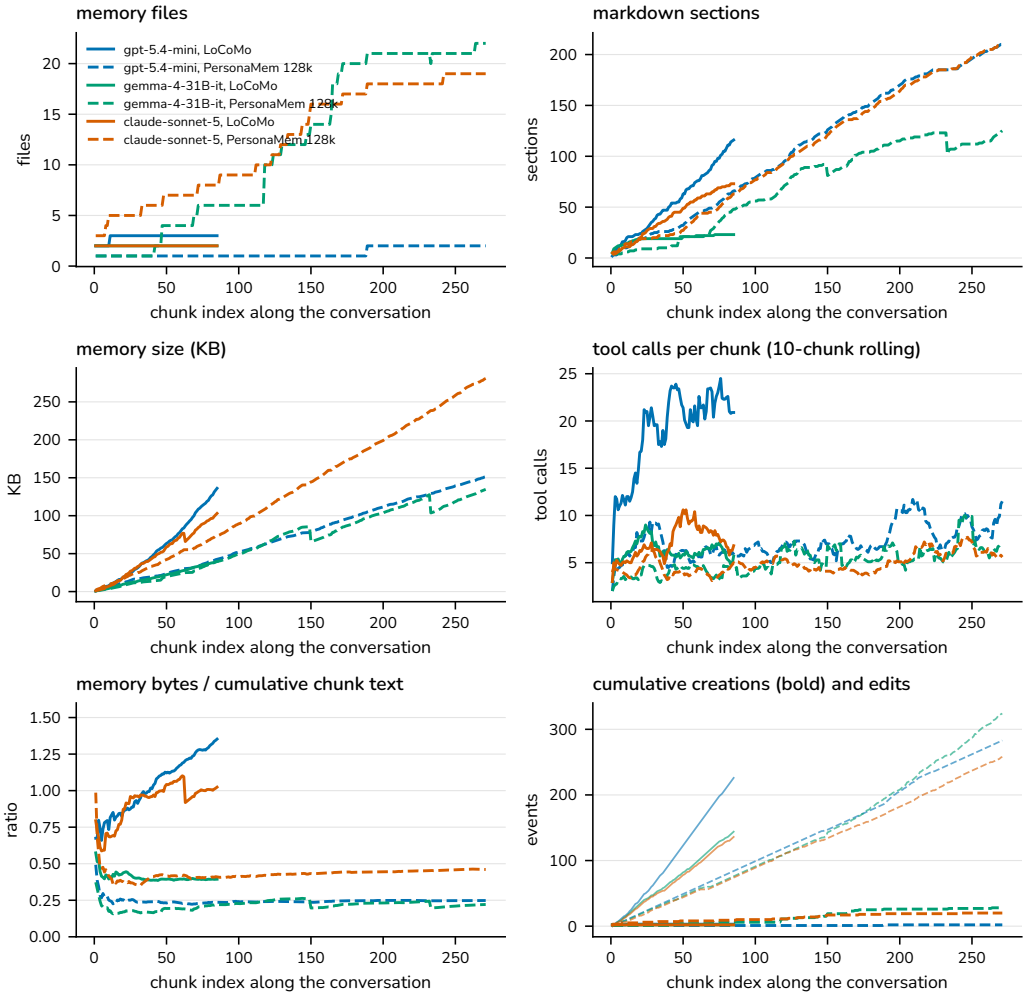


Figure A12: Conversational build curves across model families, the counterpart of Figure A2, from the per-chunk trajectories of the Agent-curated memory builds of record (LoCoMo solid, PersonaMem 128k dashed), with the two panels of Figure A2 that need per-chunk memory states added at the bottom. Files: every family keeps LoCoMo at two or three files; on PersonaMem 128k `gpt-5.4-mini` stays at one or two files while `gemma-4-31B-it` and `claude-sonnet-5` grow to 22 and 19 files in steps. Sections grow roughly linearly with the conversation for every family (LoCoMo: 116 for `gpt-5.4-mini`, 73 for `claude-sonnet-5`, 23 for `gemma-4-31B-it`; PersonaMem 128k: 210, 211 and 125). Size: `claude-sonnet-5`'s PersonaMem memory grows fastest (281 kilobytes against 151 and 135). Tool calls per chunk stay flat for every family after the first chunks, with `gpt-5.4-mini` working LoCoMo at about twenty calls per chunk against five to ten for the other two. Bottom: memory bytes over the cumulative chunk text and cumulative creations against edits; `claude-sonnet-5`'s LoCoMo memory ends at the size of its input stream (1.02) and its PersonaMem memory at 0.46, `gemma-4-31B-it`'s at 0.39 and 0.22, against `gpt-5.4-mini`'s 1.35 and 0.25, and every family's build is edit-dominated from the first chunks (2 to 28 creations against 136 to 318 edits).

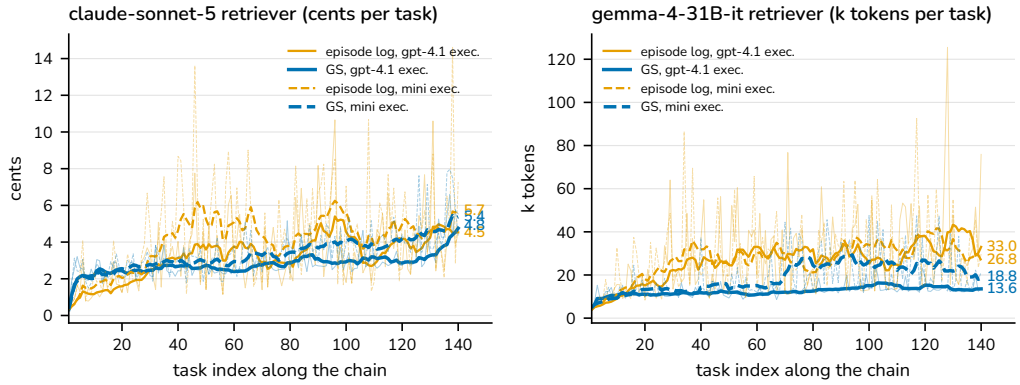


Figure A13: Search cost per task along the full chain across model families, the counterpart of Figure A4 (faint lines per task, bold lines a 10-task rolling mean; Episode log orange, Curated skills+mem (GS) blue; solid `gpt-4.1` execution agent, dashed `gpt-4.1-mini`). Left: `claude-sonnet-5`'s retriever in billed cents; right: `gemma-4-31B-it`'s in its own tokens. `gemma-4-31B-it` repeats the `gpt-5.4-mini` pattern, the Episode log's serve-everything retrieval climbing with the memory to 2.4 times the Curated skills+mem (GS) price at the `gpt-4.1` tier (33 against 14 thousand tokens) and 1.4 times at mini (27 against 19). `claude-sonnet-5`'s two `gpt-4.1`-tier chains end at the same price (4.5 and 4.8 cents), the Episode log's retrieval rising and the Curated skills+mem (GS) memory's staying flat until the last quarter, where its retriever begins to read the seven files more fully; at the mini tier its two chains end at 5.4 (GS, eight files) and 5.7 (episode log) cents per task.

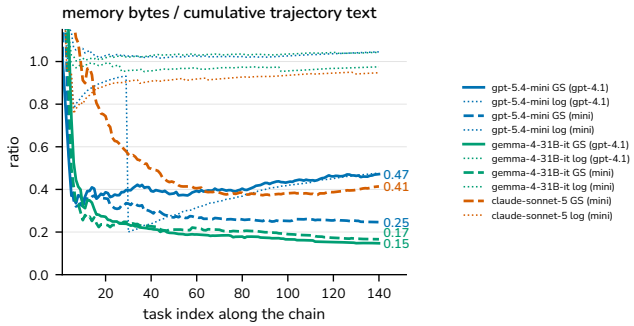


Figure A14: Memory bytes over the cell's own cumulative streamed trajectory text along the chain, the counterpart of Figure A8, for `gpt-5.4-mini`, `gemma-4-31B-it` and `claude-sonnet-5`'s mini chains (Curated skills+mem (GS) solid or dashed by tier, Episode log dotted). `gemma-4-31B-it`'s curated memories compress the experience to 15 and 17 percent at the two tiers against `gpt-5.4-mini`'s 47 and 25, while the episode logs stay near or above their stream (95 to 105 percent), except `gpt-5.4-mini`'s mini log at 48, where one runaway episode inflates the stream. The `claude-sonnet-5` chains of record were resumed from their run records after the outages, and their archives carry the trajectory text of the live tasks only, so their ratio is not computable; its mini chains, run without interruption, end at 0.41 (GS) and 0.95 (episode log).

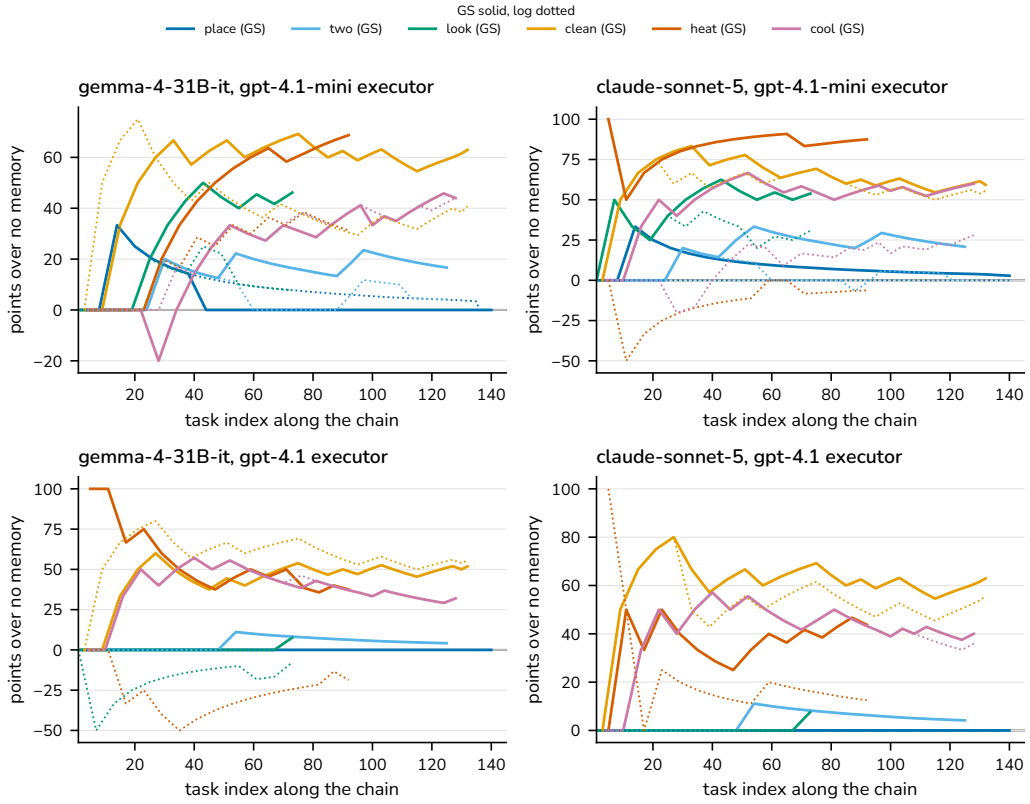


Figure A15: Per-goal-family differenced running success at the family's interleaved positions across model families, the counterpart of Figure A5 (Curated skills+mem (GS) solid, Episode log dotted, both minus the no-memory chain of the tier; each curve ends at its family's last task in the fixed order). Transfer is family-specific for every management agent: the place family gains at most three points (the no-memory chain already solves it), clean gains 41 to 63 points for every memory, and heat separates the management agents, ending 69 and 88 points over no memory with `gemma-4-31B-it`'s and `claude-sonnet-5`'s curated memories at the mini tier and 38 to 44 with the `gpt-4.1` execution agent for both replicated families, where the Episode log ends at 13 (`claude-sonnet-5`) and −19 (`gemma-4-31B-it`), and at the mini tier at −6 (`claude-sonnet-5`) and 31 (`gemma-4-31B-it`).

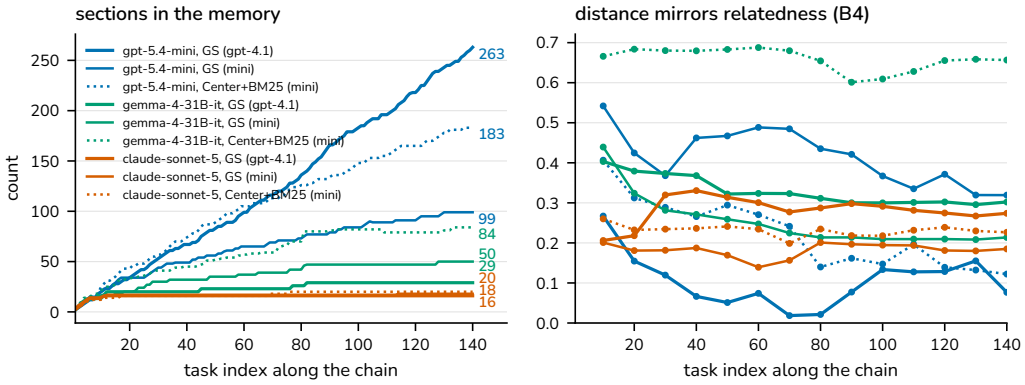


Figure A16: Content-level hierarchy along the chain across model families, the counterpart of Figure A10: the three full chains of `gemma-4-31B-it` and of `claude-sonnet-5` beside the three `gpt-5.4-mini` chains of that figure (solid: Curated skills+mem (GS) at the `gpt-4.1` tier; dashed: Curated skills+mem (GS) at the mini tier; dotted: Center+BM25 at the mini tier). The `gemma-4-31B-it` and `claude-sonnet-5` memories are reconstructed after every task from identity replays of their chains and validated against the per-task snapshots of the run records (840 states, zero mismatches). Left: sections. `claude-sonnet-5`'s memories end at 16 to 20 sections and `gemma-4-31B-it`'s at 29 to 84, where `gpt-5.4-mini`'s reach 99 to 263. The strong-tier GS chains stop adding sections at tasks 11 and 82; smaller additions continue later in the mini-tier and Center+BM25 chains. Right: the distance-mirrors-relatedness correlation per snapshot. `gpt-5.4-mini`'s erodes as its memories grow (.26 to .07, .40 to .12 and .54 to .31 from task 10 to task 140); the replicated families' correlations remain positive after task 47, but do not remain constant. Over that interval, `claude-sonnet-5` ranges from .13 to .32 and `gemma-4-31B-it` from .21 to .69 across the plotted chains; `gemma-4-31B-it` ends at .30, .21 and .66.

G USE OF AI ASSISTANTS FOR ILLUSTRATIVE FIGURES

AI assistants helped with proofreading, editorial suggestions, schematic illustrations, and plotting code. The authors reviewed the final text, analyses, and figures. Plotted observations come from the recorded experiments; AI assistance was used to present and check those results, not to generate experimental observations.