MemCodex: Self-Programming Hierarchical Memory for Language Agents

Anonymous authors
Paper under double-blind review

Abstract

Agent memory faces heterogeneous access needs: a single-hop question may require one piece of evidence, whereas a multi-hop question must combine evidence from multiple sources. Predefined memory workflows cannot adapt to these varying needs. Recent adaptive methods search or learn over memory components and their compositions, but the design space itself remains predefined. We introduce **MemCodex**, a self-evolving hierarchical memory system that organizes experience into executable memory programs for summaries, relational knowledge, reusable skills, and latent memory. Open-ended program evolution searches the open design space of layer programs by rewriting how each layer is constructed, indexed, retrieved, and routed, thereby adapting both within-layer implementations and cross-layer composition. At query time, reads traverse the hierarchy from coarse to fine and stop once sufficient evidence is found, descending to the original history when needed. We further develop MemArena, a unified runtime that places heterogeneous data and memory systems behind a common interface. `MemCodex` improves average task success by 10.1% relative to the strongest adaptive-memory baseline, with $3.4\times$ fewer context tokens and $2.1\times$ faster inference.

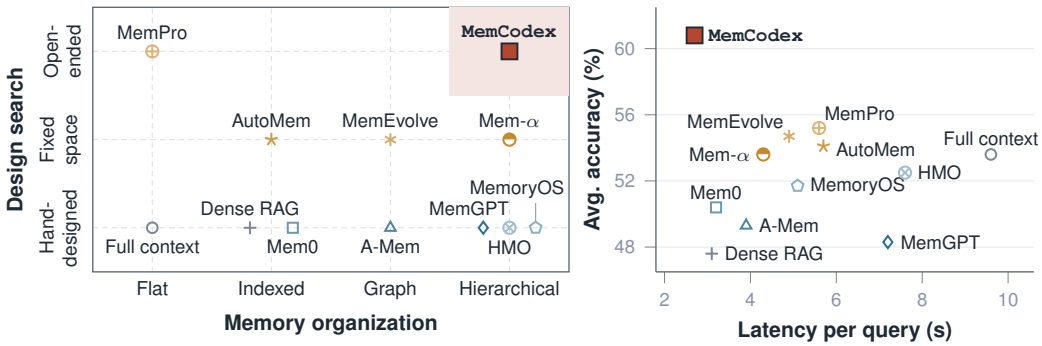

Figure 1: Memory systems by memory organization and design search (left), and average test accuracy against latency per query (right).

1 Introduction

Language agents (Yao et al., 2022; Wang et al., 2024a) increasingly act over long horizons in web navigation (Zhou et al., 2023), software engineering (Jimenez et al., 2023; Hong et al., 2023) and computer use (Xie et al., 2024), and self-evolve from experience (Tao et al., 2024) through reflection on failures (Shinn et al., 2023), skill accumulation (Wang et al., 2023; Zhao et al., 2024; Ouyang et al., 2025) and code rewriting (Zhang et al., 2025b). Across sessions, much of this experience survives only in memory. Hand-designed pipelines organize it through rolling summaries (Zhong et al., 2024; Packer et al., 2023), facts and entity graphs (Xu et al., 2025; Rasmussen et al., 2025; Edge et al., 2024), skill stores (Zhao et al., 2024; Ouyang et al., 2025; Wang et al., 2023), or memory written into parameters or latent state (Wang et al., 2024b; Behrouz et al., 2024; Li et al., 2025a). Yet they face *access heterogeneity*: the memory organization a query needs depends on the query, the record and the task (Xu et al., 2026; Lu et al., 2026; Huang et al., 2026). Single-hop questions need raw turns (Han et al., 2026), multi-hop questions need cross-session links (Gutiérrez et al., 2024), temporal questions

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.

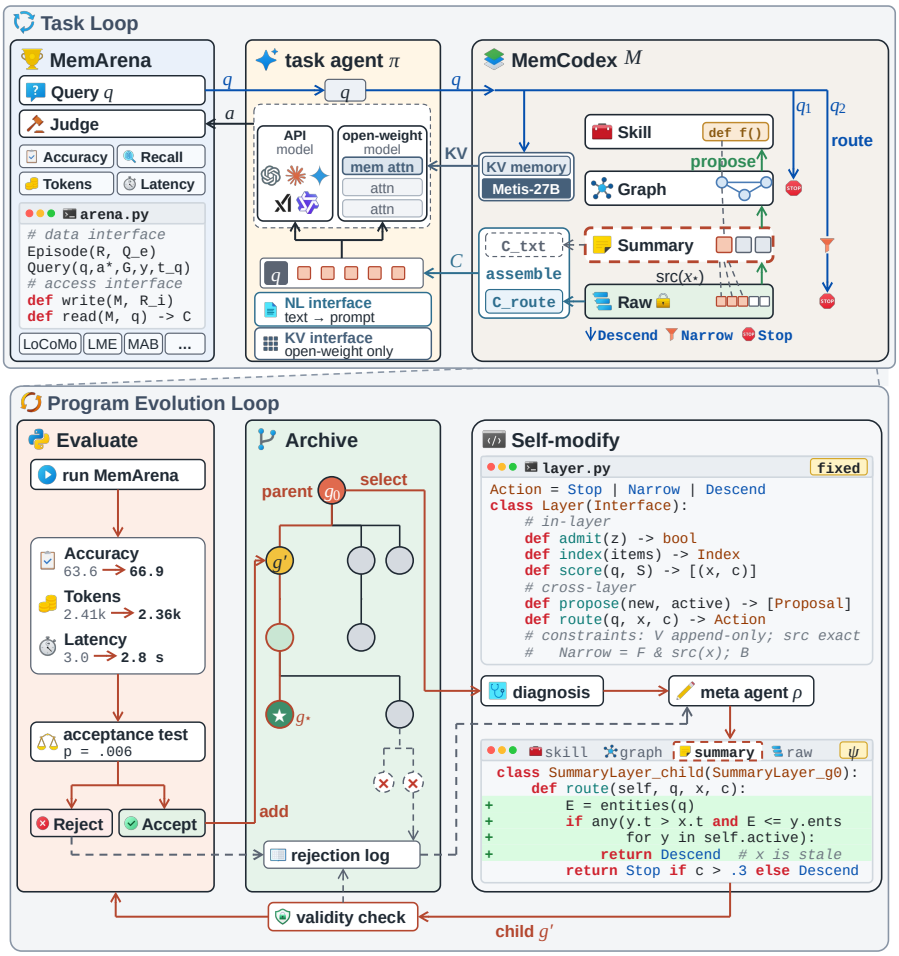

Figure 2: MemCodex workflow. In the task loop (upper), MEMARENA writes the record and poses each query q to the task agent, which reads a context C from the layer hierarchy, stopping at the coarsest sufficient layer or descending to source turns, and answers a. In the program evolution loop (lower), a parent $g = (\lambda, \psi)$ is sampled from the archive, meta agent ρ rewrites one method of one layer program ψ_ℓ, and validity checks and the acceptance test admit the child to the archive.

need dated events (Ge et al., 2025), and repeated procedures need skills (Wang et al., 2025b). A hand-designed pipeline hard-codes which organizations exist and the rule that chooses among them, and a study of fourteen memory systems (Zhou et al., 2026) finds that no architecture dominates across tasks.

Adaptive memory follows two directions. One searches a predefined space by composing modules for encoding, storage, retrieval, and management (Du et al., 2026; Lu et al., 2026; Zhang et al., 2025a). The other learns when and how to write, update, and retrieve memory (Wang et al., 2025a; Sun et al., 2026; Wang et al., 2026). Despite their different forms, these methods must jointly determine *how* experience is organized into memory and *how* queries access it, since retrieval can only use structures created during writing. As shown in Figure 1, memory systems therefore differ along two axes: memory organization and design search. Among prior systems, only MemPro (Liu et al., 2026d) opens the design itself to program search, but evolves a single monolithic program over flat memory, without a hierarchy whose components can evolve and be accessed independently.

We therefore introduce MemCodex, a hierarchical memory system whose layers are executable memory programs behind a fixed interface and runtime. Program evolution adapts the write and read code of individual layers as well as their cross-layer composition, that is, which layers are active and how items and reads pass between them. MemCodex organizes SKILL, GRAPH, and SUMMARY above an append-only raw layer, with KV MEMORY providing latent memory.

Inspired by the Darwin Gödel Machine (Zhang et al., 2025b) and HyperAgents (Zhang et al., 2026b), which evolve an agent by rewriting its code, we apply program evolution to memory. Each round, a

meta agent draws a parent from the archive and rewrites one method of one layer or toggles one layer. One change per child makes its effect attributable. An acceptance test keeps changes that improve accuracy or token use without degrading the other, and accuracy-only gains must pass a one-sided McNemar test (McNemar, 1947). Validity checks $\mathcal{K}$ reject changes that break the memory constraints.

Each layer exposes five methods for memory writing and reading. On the write side, admit filters candidates, index bounds the active set, and propose derives items for the layer above. On the read side, score ranks candidates and route directs the search (schema in Appendix B.4), moving from coarse to fine layers by STOP, NARROW, or DESCEND until evidence suffices or the raw history is reached. Every item keeps its exact source turns, so evidence survives summarization.

To support systematic evaluation across diverse sources, interaction formats, and protocols, we develop MEMARENA, a heterogeneous agent-source evaluation suite with a unified data interface, access interface, and metric suite covering LoCoMo (Maharana et al., 2024), LongMemEval (LME) (Wu et al., 2025), and MemoryAgentBench (MAB) (Hu et al., 2025b). Under one protocol, MEMARENA spans seven sources, eleven baseline systems and two backbones. `MemCodex` gains 5.6 average accuracy points over the strongest baseline with Qwen3.5-27B (Qwen Team, 2026) and 3.6 with the API model GPT-5.4-mini, while reading $3.4\times$ fewer tokens and answering $2.1\times$ faster.

2 MEMCODEX: HIERARCHICAL MEMORY AS EVOLVING PROGRAMS

We model agent memory as a hierarchy of executable memory programs whose implementations and composition evolve with the task (Figure 2). A fixed five-method layer interface and runtime constrain how these programs interact while leaving their internal behavior open to evolution.

A task agent π answers questions about a record R that arrives as write chunks $R_1, \ldots, R_n$ of timestamped turns v, and each question q is posed once the record up to its time is written. A memory system with architecture g keeps a store $\mathcal{M}$ and serves π through two operations:

$$\mathcal{M} \leftarrow \texttt{write}_g(\mathcal{M}, R_i), \qquad C = \texttt{read}_g(\mathcal{M}, q), \qquad a = \pi(q, C). \tag{1}$$

The write operation folds each chunk into the store, the read operation returns a context C of at most B tokens, and π answers from that context. The store contains an append-only *raw layer* $\mathcal{V}$ of all turns and lossy *derived items* x above it, each retaining its exact *source turns* $\mathrm{src}(x) \subseteq \mathcal{V}$.

An *architecture* $g = (\lambda, \psi)$ has two parts. The layer programs ψ implement the write through admit, index and propose and the read through score and route, five methods per layer. The layer mask λ toggles derived layers. Each query q of a task's questions $\mathcal{Q}$ has *gold turns* $G(q) \subseteq \mathcal{V}$. The accuracy of g is $\mathrm{acc}(g) = \mathbb{E}_{q\sim\mathcal{Q}}\,\mathbf{1}[\pi(q, C_g(q)) \text{ correct}]$ and its cost is $\mathrm{tok/q}(g) = \mathbb{E}_{q\sim\mathcal{Q}}|C_g(q)|$. We seek the most accurate architecture at no higher token cost than g_0. Program evolution, defined after ψ, changes g one method rewrite or one layer toggle at a time and archives accepted architectures.

2.1 LAYER PROGRAMS ψ

Layers. Queries require different levels of abstraction, but every derived item must stay connected to its raw evidence. We therefore order the four text layers $\mathcal{L} = (\text{SKILL}, \text{GRAPH}, \text{SUMMARY}, \text{RAW})$ from coarse to fine, as in RAPTOR (Sarthi et al., 2024), with store $\mathcal{M} = \mathcal{V} \cup \bigcup_{\ell \neq \text{RAW}} \mathcal{M}_\ell$, where $\mathcal{M}_\ell$ holds the derived items of layer ℓ. The RAW layer keeps every raw turn $v \in \mathcal{V}$ unchanged. The SUMMARY layer condenses each closed segment of raw turns into a compact account. The GRAPH layer stores timed assertions $x = (h, r, t, \tau)$ extracted from summaries, where entities h and t are linked by relation r over the time τ, so that a newer assertion can supersede an older one about the same entities. The SKILL layer stores executable procedures distilled from assertions recorded in several sessions. A skill is a program rather than a text note because a running procedure can be checked against the sessions that demonstrate it. Each derived item x is built from inputs $\mathrm{in}(x)$ in the layer directly below, and its exact source turns $\mathrm{src}(x)$ resolve recursively,

$$\mathrm{src}(v) = \{v\} \;\; (v \in \mathcal{V}), \qquad \mathrm{src}(x) = \textstyle\bigcup_{y \in \mathrm{in}(x)} \mathrm{src}(y) \;\; (x \in \mathcal{M} \setminus \mathcal{V}), \tag{2}$$

and Appendix B.5 traces an example record through every layer. A read can therefore reach the raw turns underlying an assertion or a skill directly. The set $\mathrm{src}(x)$ records an item's exact derivation rather than a minimal evidence set. A summary and its extracted assertions share the segment's turns, which NARROW and the raw-layer read subsequently refine.

Layer interface. Local program evolution requires every method rewrite to stay within its layer. We therefore expose each text layer through five methods. Within a layer, admit_ℓ filters proposals, while index_ℓ retains at most K_ℓ items in the active index $\mathcal{I}_\ell$. Evicted items remain in the store. The method $\mathrm{score}_\ell(q, S)$ ranks the items S allowed by the *search scope* $F \subseteq \mathcal{V}$ and returns the top candidate $x^\star$ with confidence c. Across layers, $\mathrm{propose}_\ell$ sends derived items upward, while $\mathrm{route}_\ell(q, x^\star, c)$ returns STOP, NARROW or DESCEND. The runtime owns `assemble`, which packs C under B. In every architecture, turns are appended only to the raw layer and never changed. Every derived item names the exact raw turns used to build it. NARROW can only restrict the search scope to the top candidate's source turns, $F \cap \mathrm{src}(x^\star)$. Every context must also fit within B tokens. These four *memory constraints* ensure that any read can trace a derived item back to raw turns, however lossy the item is.

Content channel and routing channel. A derived item serves a query through one of two channels, and the query-adaptive read uses the routing channel. The channel is an architecture setting, not a per-query choice, so only the read depth adapts. In the *content channel* $\mathsf{C}_{\mathrm{txt}}$, the top-scoring items enter C as text in place of their evidence, as in summary, profile and skill memories (Packer et al., 2023; Chhikara et al., 2025; Zhao et al., 2024). In the *routing channel* $\mathsf{C}_{\mathrm{route}}$, a retrieval hit leads to its source turns, as in HippoRAG (Gutiérrez et al., 2024). Figure 3 contrasts the two channels on one question. *Evidence preservation*, retaining every derived item's exact source turns, enables this route.

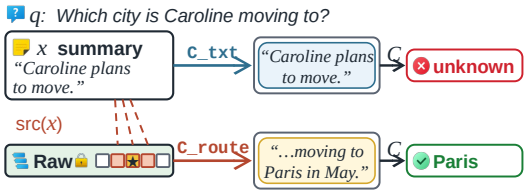

Figure 3: Content and routing channels of `MemCodex` for a question about Caroline. $\mathsf{C}_{\mathrm{txt}}$ places summary x in C, omitting Paris, while $\mathsf{C}_{\mathrm{route}}$ follows $\mathrm{src}(x)$ to the raw turn naming Paris.

Query-adaptive read. A fixed read depth can either skip useful abstractions or retrieve unnecessary detail. The $\mathtt{read}_g$ operation in Eq. (1) therefore visits layers from coarse to fine (Algorithm 1) and lets each route determine whether traversal should continue. Only STOP ends the read and adds the in-scope source turns $\mathrm{src}(x^\star) \cap F$ to C in temporal order through the routing channel. The NARROW action restricts the search scope to those source turns, while DESCEND keeps it, and both continue to the next layer. At the raw layer, the k highest-scoring in-scope turns enter C. The routing channel ensures that the task agent answers from raw turns, even when the read stops at a skill, whose source turns are its demonstrating sessions. Each query incurs only the depth it needs.

Write path. Derived layers depend on lower-layer content, so the $\mathtt{write}_g$ operation in Eq. (1) processes each appended chunk from bottom to top through propose, admit and index. Each proposal to a layer combines the items newly admitted one layer below for the current chunk with the active items of that lower layer. The runtime records every item read to form the proposal as an input. At SKILL, admit accepts a procedure only when replay on every demonstration session reproduces the recorded outcome. The task agent's backbone writes the textual proposals. In the initial architecture g_0, index retains items according to the access-and-recency heat of MemoryOS (Kang et al., 2025), score uses cosine similarity and route stops or narrows above per-layer thresholds.

Latent memory. Some associations in the record have no representation in any text layer. We feed each write chunk into Metis-27B (Zhang et al., 2026d), a memory foundation model with stateful fast weights, and use the state of its memory blocks as KV MEMORY, a fixed-size key-value memory. KV MEMORY is write-only, exposes no score or route, is built on the write path, and is not evolved. Two interfaces provide memory to the task agent. The natural language (NL) interface places the text context C in the prompt and supports any agent. The KV interface injects KV MEMORY into an open-weight agent's memory attention. Because an API agent exposes neither weights nor activations, it uses only the NL interface and has no latent memory.

2.2 EVOLVING THE ARCHITECTURE g

Architecture. Joint evolution of layer behavior and composition requires edits local to one component. The layer mask $\lambda \in \{0, 1\}^{|\mathcal{L}|-1}$ covers the derived layers, the raw layer staying active. Reads visit only active layers. Writes traverse every layer up to the highest active one, so every proposal keeps its type and derivation even when an intermediate layer is off for reading. The mask and cross-layer meth-

Algorithm 1 QUERYADAPTIVEREAD

Input: query q, store $\mathcal{M}$ with an active index $\mathcal{I}_\ell$ per layer, architecture g with active layers $\mathcal{L}_g$ ($\lambda = 1$), budget B, channel $\chi \in \{\mathsf{C}_{\text{txt}}, \mathsf{C}_{\text{route}}\}$, and top_k the first k leaves by $\text{score}_{\text{RAW}}$
Output: context C

$F \leftarrow \mathcal{V}$ ▷ every raw turn
for $\ell \in \mathcal{L}_g \setminus \{\text{RAW}\}$, coarsest first **do**
  $S \leftarrow \{x \in \mathcal{I}_\ell : \text{src}(x) \cap F \neq \varnothing\}$
  $X \leftarrow \text{score}_\ell(q, S)$
  if $X = \varnothing$ **then continue**
  $(x^\star, c) \leftarrow X_1$ ▷ top candidate
  $o \leftarrow \text{route}_\ell(q, x^\star, c)$ ▷ searched method
  if $o =$ STOP **then**
    if $\chi = \mathsf{C}_{\text{txt}}$ **then** $Y \leftarrow \text{text}(X_{1..10})$
    else $Y \leftarrow \text{src}(x^\star) \cap F$ in time order
    return $\text{assemble}_B(Y)$
  else if $o =$ NARROW **then**
    $F \leftarrow F \cap \text{src}(x^\star)$
return $\text{assemble}_B(\text{top}_k(q, F))$ ▷ STOP at RAW

Algorithm 2 PROGRAMEVOLUTION

Input: initial architecture g_0, evolve questions Q, meta agent ρ, validity checks $\mathcal{K}$, rounds T, parents P, children per parent H
Output: archive $\mathcal{A}$ and the evolved architecture $g^\star$

$\mathcal{A} \leftarrow \{g_0\}$, $E_{g_0} \leftarrow \text{EVALUATE}(g_0, Q)$
for $r = 1, \ldots, T$ **do**
  for all $n \sim w$ by Eq. (4), $j \leq H$, **in parallel do**
    $(\ell, u) \leftarrow$ next target of $\mathcal{D}(g_n)$
    if $u =$ toggle **then** $g' \leftarrow g_n$, flip λ_ℓ
    else $g' \leftarrow \rho(g_n, \mathcal{D}(g_n), \ell, u, \mathcal{J}_n)$
    if some $\kappa \in \mathcal{K}$ fails on g' **then**
      $\mathcal{J}_n \leftarrow \mathcal{J}_n \cup \{(g', \kappa)\}$; **continue**
    $E_{g'} \leftarrow \text{EVALUATE}(g', Q)$
    if $\mathsf{Accept}(g_n \to g')$ **then** add g' under n
    else $\mathcal{J}_n \leftarrow \mathcal{J}_n \cup \{(g', E_{g'})\}$
return $\mathcal{A}$, $g^\star \leftarrow \arg\max_{g \in \mathcal{A}} \text{acc}_Q(g)$

ods set composition, the within-layer methods implementation. From g_0, program evolution combines evaluator-guided program search (Hu et al., 2025a; Romera-Paredes et al., 2024; Novikov et al., 2025), the open-ended archive search of the Darwin Gödel Machine (Zhang et al., 2025b) and the Hyper-Agents meta agent (Zhang et al., 2026b). Meta agent ρ shares the task agent's backbone, so the agent programs its own memory. *Attributable edits* limit each child to one method rewrite of one layer or one layer toggle. The acceptance test then admits only improvements that trade no accuracy for tokens.

Archive. A single chain of edits cannot escape a dead end. Following MemPro (Liu et al., 2026d), we store accepted architectures in the tree-shaped archive $\mathcal{A}$ and rejected children in the parent's rejection log $\mathcal{J}_n$. Each node stores acc_Q, tokens per question and verdicts $\mathbf{v}_g \in \{0,1\}^{|Q|}$. Its parent weight $w(n)$ favors accurate, rarely expanded nodes on the Pareto front. Any archived node can be a parent and a child may hold any code meeting the layer interface, so evolution is open-ended.

Self-modification. Each local edit should address an observed failure. The parent's diagnosis $\mathcal{D}(g)$ therefore examines failed traces and refused or evicted items, then assigns each failure to the first method that loses a gold turn. Children target successive methods or layer toggles according to fault count, so siblings make distinct edits. For a method rewrite, meta agent ρ reads the target layer's code, diagnosis and rejection log, then returns a subclass that overrides only that method (Algorithm 2). Validity checks $\mathcal{K}$ reject any child that violates a memory constraint on fixture queries.

Acceptance test. Validity checks determine whether a child preserves memory constraints, but not whether it trades accuracy for tokens. The acceptance test compares child and parent on the same evolve questions with task agent and judge frozen, accepting the child exactly when

$$\mathsf{Accept}(g \to g') \iff \Delta_{\text{tok}} \leq 0 \ \wedge \ \Delta_{\text{acc}} \geq 0 \ \wedge \ \big[(\Delta_{\text{acc}} > 0 \ \wedge \ p < \alpha) \ \vee \ \Delta_{\text{tok}} < 0\big], \quad (3)$$

where Δ_{acc} and Δ_{tok} are child-minus-parent changes in accuracy and tokens per question, p is the one-sided exact McNemar p-value (McNemar, 1947) on discordant pairs of $\mathbf{v}_g$ and $\mathbf{v}_{g'}$, and $\alpha = 0.05$. The evolved architecture $g^\star$ is the archived architecture with the highest evolve-split accuracy. Appendix B specifies the write path, g_0, the graph and latent layers, parent selection and the meta-agent prompt.

3 MEMARENA: A HETEROGENEOUS AGENT-SOURCE EVALUATION SUITE

Results for memory systems are reported across sources with different evidence formats, time representations and access protocols, making them difficult to compare. MEMARENA places heterogeneous sources and memory systems behind one data interface, one access interface and one metric suite, allowing any memory module to be evaluated with any task agent under a fixed backbone and judge. It currently spans seven sources, eleven baseline systems and two backbones. Adding a source requires only one adapter. We release the suite with its adapters and wrappers.

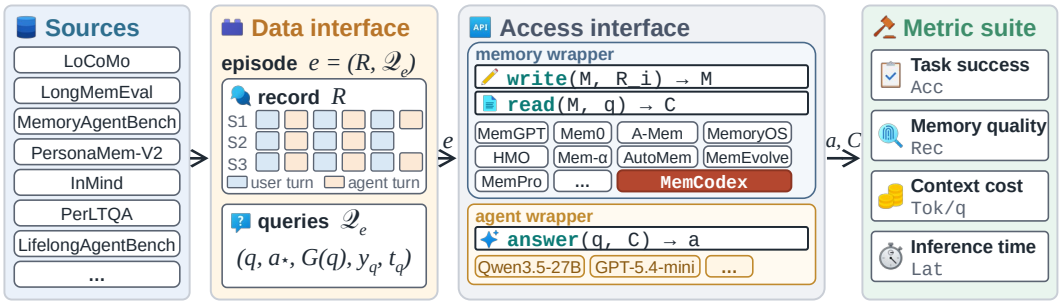

Figure 4: MEMARENA places heterogeneous sources behind unified data, access and metric interfaces.

Data interface. Source adapters map corpora with different evidence and time representations into an episode $e = (R, \mathcal{Q}_e)$ (Figure 4). The record R orders sessions of timestamped, speaker-tagged turns, while each tuple $(q, a^\star, G(q), y_q, t_q) \in \mathcal{Q}_e$ specifies a question, reference answer, gold turns, question type and question time, posed after the record through t_q is written. Adapters obtain $G(q)$ from evidence annotations for LoCoMo and LongMemEval and from task construction for PersonaMind and MAB. They cover seven sources (Maharana et al., 2024; Wu et al., 2025; Hu et al., 2025b; Jiang et al., 2025; Li et al., 2025b; Du et al., 2024; Zheng et al., 2025). PersonaMind, our fourth benchmark, poses five preference or procedure questions for each of 300 PersonaMem-V2 and InMind records.

Access interface. To separate memory from task-agent quality, MEMARENA wraps memory systems and task agents independently. A memory wrapper exposes $\texttt{write}(\mathcal{M}, R_i) \to \mathcal{M}'$ and $\texttt{read}(\mathcal{M}, q) \to C$, where C may carry KV MEMORY. An agent wrapper exposes $\texttt{answer}(q, C) \to a$ for π, using the NL interface for text and the KV interface for KV MEMORY on open-weight agents. The harness writes each record into a fresh store in 20-turn chunks and poses each question after the last chunk written before t_q, so stores are never shared across records or users and only g is shared.

Metric suite. MEMARENA reports accuracy under the Mem0 rubric (Chhikara et al., 2025), evidence recall where gold turns are annotated, $\mathrm{tok/q}$ and latency, each defined in Appendix C.3.

4 EXPERIMENTS

Benchmarks and splits. We evaluate every system through MEMARENA on LoCoMo (Maharana et al., 2024), LongMemEval (S split) (Wu et al., 2025), MemoryAgentBench (MAB) (Hu et al., 2025b) and PersonaMind. Following MemEvolve (Zhang et al., 2025a) and MemPro (Liu et al., 2026d), which evolve on a held-out slice of each benchmark, we take 20 percent of each benchmark's questions as the evolve split Q. We stratify Q by question category, such as single-hop, multi-hop, temporal and open-domain questions on LoCoMo, the five abilities of LongMemEval, the four competencies of MAB, and preference against procedure questions on PersonaMind. We never read the test split during search. Appendix C.2 repeats the comparison using record-disjoint splits.

Baselines. Unstructured baselines are 32k-token full context and dense RAG with BGE (Xiao et al., 2024) over 512-token chunks under $B = 4{,}096$. Hand-designed baselines are MemGPT (Packer et al., 2023), A-Mem (Xu et al., 2025), Mem0 (Chhikara et al., 2025), MemoryOS (Kang et al., 2025) and HMO (Liu et al., 2026c). Adaptive or evolved baselines are Mem-α (Wang et al., 2025a), AutoMem (Du et al., 2026), MemEvolve (Zhang et al., 2025a) and MemPro (Liu et al., 2026d). Search-based baselines share the evolve split, meta agent and 64-child budget.

Models and search. We evaluate open-weight Qwen3.5-27B (Qwen Team, 2026) and API model GPT-5.4-mini. Per run, one backbone serves every system as task agent π, writes derived items, and serves as meta agent ρ at temperature 0.2 with one call per child. Qwen3.5-27B reads text through the NL interface and Metis-27B (Zhang et al., 2026d) state KV MEMORY through the KV interface, while GPT-5.4-mini uses only the NL interface. Metis-27B inserts fast-weight memory blocks into frozen Qwen3.5-27B and updates them by test-time training with a gated delta rule, as in Titans (Behrouz et al., 2024). Per record, we run Metis-27B and extract its key-value state. The shared backbone makes injection native. GPT-4o-mini (Hurst et al., 2024) judges. Evolution uses $T = 8$ rounds, $P = 4$ parents and $H = 2$ children per parent. Appendix C lists benchmarks, baselines and constants.

Table 1: Results of MemCodex with Qwen3.5-27B and GPT-5.4-mini. **Bold** marks the column best, underline the strongest baseline, and parentheses the margin. Tok/q, LME, MAB and PMind denote thousands of prompt tokens per question, LongMemEval, MemoryAgentBench and PersonaMind.

	Qwen3.5-27B					GPT-5.4-mini					
Method	LoCoMo	LME	MAB	PMind	Avg.	LoCoMo	LME	MAB	PMind	Avg.	Tok/q
Full context	61.8	57.6	43.9	51.2	53.6	64.1	59.8	46.2	53.3	55.9	21.4
Dense RAG	48.9	55.1	40.7	45.8	47.6	51.2	57.3	43.0	48.1	49.9	3.6
MemGPT	55.4	50.9	39.8	47.1	48.3	58.0	52.9	41.9	49.5	50.6	8.9
A-Mem	54.6	53.2	41.5	47.9	49.3	56.8	55.6	43.4	50.1	51.5	5.1
Mem0	56.4	54.1	42.2	48.8	50.4	58.7	56.4	44.5	51.1	52.7	4.2
MemoryOS	58.0	54.8	43.6	50.3	51.7	60.3	57.0	45.9	52.6	54.0	6.3
HMO	58.7	55.9	44.1	51.3	52.5	60.9	58.3	46.2	53.4	54.7	7.4
Mem-α	59.3	56.8	45.7	52.5	53.6	61.5	58.9	47.9	54.6	55.7	5.8
AutoMem	60.1	57.3	46.0	52.9	54.1	62.4	59.6	48.3	55.2	56.4	6.1
MemEvolve	60.6	58.0	46.6	53.5	54.7	62.9	60.3	48.9	55.8	57.0	6.6
MemPro	61.2	58.4	47.1	54.0	55.2	63.5	60.7	49.4	56.2	57.5	6.8
Initial g_0	63.4	59.7	48.3	55.7	56.8	65.3	61.6	50.1	57.1	58.5	2.4
MemCodex	**67.1**	**62.9**	**52.3**	**60.9**	**60.8** (+5.6)	**68.2**	**63.9**	**52.8**	**59.4**	**61.1** (+3.6)	**2.0**

Table 2: Ablations of MemCodex with one change per row at a matched budget. Avg. averages four benchmarks, Rec. is LoCoMo recall, and parentheses give changes from MemCodex.

Method	Avg. (%)	Rec. (%)	Tok/q (k)	Method	Avg. (%)	Rec. (%)	Tok/q (k)
MemCodex	60.8	88.4	2.0	*Evolution*			
Hierarchy depth				No evolution	56.8 (−4.0)	82.6 (−5.8)	2.4
1 layer (raw)	54.1 (−6.7)	71.2 (−17.2)	1.8	Module selection	58.3 (−2.5)	84.9 (−3.5)	2.3
2 layers (+summary)	56.5 (−4.3)	78.9 (−9.5)	2.3	Layer rewrites	57.2 (−3.6)	80.9 (−7.5)	2.9
3 layers (+graph)	59.0 (−1.8)	84.6 (−3.8)	2.1	Program rewrites	56.7 (−4.1)	79.8 (−8.6)	3.2
Evidence channel				No acceptance test	58.1 (−2.7)	84.7 (−3.7)	2.6
Content channel	55.4 (−5.4)	63.0 (−25.4)	3.9	No archive	59.2 (−1.6)	86.8 (−1.6)	2.1
Content + routing	59.6 (−1.2)	88.7 (+0.3)	4.6				

4.1 Main Results

MemCodex is the most accurate system on all four benchmarks and reads the fewest tokens. It averages 60.8 percent, 5.6 points above MemPro and 7.2 above full context, while reading 2.0k tokens per question against 6.8k and 21.4k (Table 1). It answers in 2.7 seconds, 2.1× faster than MemPro and 3.6× faster than full context (Figure 1, right). Seed standard deviations are at most 0.8 points (Appendix D.3). Margins over MemPro peak on PersonaMind (+6.9) and LoCoMo (+5.9), which we attribute to latent memory carrying recurring preferences and routes linking cross-session evidence. The margin is smallest on LongMemEval (+4.5), whose records already isolate evidence.

Evidence preservation puts g_0 ahead, and evolution supplies most of the margin. The initial architecture g_0 averages 56.8 percent at 2.4k tokens, 1.6 points above MemPro, which we attribute to derived items routing reads to their exact source turns. Program evolution adds 4.0 points while reducing tokens by 17 percent. LoCoMo evidence recall rises from MemPro's 79.3 percent to 88.4 percent (Appendix D.1). With GPT-5.4-mini, which evolves its own architecture through the NL interface alone, MemCodex leads MemPro by 3.6 points, matching the margin of the Qwen3.5-27B variant without KV MEMORY, which averages 58.8 (Appendix C.5) and leads MemPro by 3.6, so latent memory contributes 2.0 points under Qwen3.5-27B.

4.2 Ablation

Table 2 tests the two claims behind MemCodex, the hierarchy with routed evidence and per-method evolution, with variants evolved at the same budget. Hierarchy depth uses one layer (raw), two (with SUMMARY), three (with GRAPH), or four (with SKILL). Evidence-channel variants place derived-item text in C instead of source turns, or alongside them. Evolution variants keep four layers and change how programs are obtained. No evolution keeps g_0. Module selection picks each method from

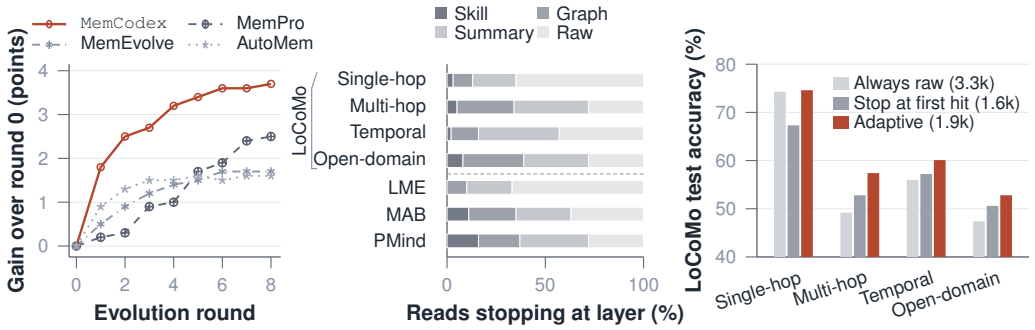

Figure 5: Evolution dynamics of MemCodex, as gain over round 0 on LoCoMo for the search-based systems (left), the fraction of test reads stopping at each layer of $g^\star$ by question type and benchmark (middle), and LoCoMo test accuracy by question type under three read policies (right).

a predefined menu as AutoMem does. Layer rewrites change one layer program per child as the slot-wise evolver of MemEvolve does, and program rewrites change every layer as MemPro does. No acceptance test archives every child passing the validity checks, and no archive follows one greedy chain. Appendix C.5 gives details and per-benchmark results.

RQ1: Does the hierarchy matter? Each added layer improves accuracy while context remains near 2k tokens. One to four layers score 54.1 (−6.7), 56.5 (−4.3), 59.0 (−1.8), and 60.8, while Tok/q remains between 1.8k and 2.3k, so we attribute the gains to reads that stop at the right level. GRAPH contributes the largest gain, +2.5 points, which we attribute to its cross-session links.

Routing to source turns preserves evidence more efficiently than reading derived text. The content channel reaches 55.4 (−5.4) with 63.0 Rec. and 3.9k Tok/q, and content plus routing restores Rec. to 88.7 but only 59.6 (−1.2) at 4.6k Tok/q, so derived text doubles context and dilutes evidence.

RQ2: Does open-ended per-method evolution beat the prior adaptive lines? Per-method evolution outperforms matched variants of the three prior search lines on the same four-layer hierarchy. MemCodex scores 60.8, compared with 56.8 (−4.0) for g_0, 58.3 (−2.5) for module selection, 57.2 (−3.6) for layer rewrites, and 56.7 (−4.1) for program rewrites. Menus remain limited to implementations anticipated by the designer. Bundled rewrites remain near g_0, with validity checks rejecting 41 of 64 layer-rewrite children against 9 of 64 per-method children.

The acceptance test prevents evolve-split overfitting, and the archive beats a single greedy chain. Without the acceptance test, accuracy falls to 58.1 (−2.7) because noisy children enter the archive. Without the archive, a greedy chain reaches 59.2 (−1.6) because it cannot revisit a useful ancestor.

4.3 Evolution Dynamics

Setting. These analyses use Qwen3.5-27B and three seeds per benchmark, with 64 children per search. Each search runs $T = 8$ rounds of eight children, and per-round test accuracy is computed after the search from archived checkpoints of the best evolve-split architecture and averaged over seeds. Retention divides the test-split gain of $g^\star$ over g_0 by its evolve-split gain. Transfer runs one benchmark's $g^\star$ unchanged on every test split.

Table 3: Accepted edits from g_0 to $g^\star$ on LoCoMo (median seed), with evolve-split and test accuracy and Tok/q after each edit.

Round	Edit	Evolve	Test	Tok/q (k)
0	g_0	63.6	63.4	2.31
1	SUMMARY.route	66.9	65.2	2.26
2	GRAPH.admit	68.2	65.9	2.21
3	RAW.score	68.5	66.1	2.13
5	SUMMARY.index	69.5	66.8	2.02
7	SKILL.route	69.9	67.0	1.94
8	GRAPH.propose	70.1	67.1	1.93

RQ3: How does accuracy change across rounds? MemCodex gains more and keeps gaining, while the three search-based baselines gain less and flatten. MemCodex gains 3.7 points, from 63.4 to 67.1, with 1.8 in the first round from a route rewrite. MemPro gains 2.5 (58.7 to 61.2), mostly after round four. MemEvolve gains 1.7 (58.9 to 60.6) and flattens after round six. AutoMem gains 1.6 (58.5 to 60.1) and plateaus from round three. Per-method evolution keeps finding edits because each child changes one method, so a small gain is attributable and passes the test.

Table 3 traces the accepted edits from g_0 to $g^\star$ on LoCoMo. The first edit rewrites the route method of SUMMARY so that a stale summary cannot stop the read, raising evolve-split accuracy from 63.6 to 66.9. Later edits each add 0.2 to 1.3 evolve-split points, while Tok/q falls from 2.31k to 1.93k. The round-5 child descends from the round-3 node rather than the round-4 leader, because the archive keeps every accepted architecture as a parent. Every accepted edit names one method of one layer.

RQ4: Do accepted gains hold on the test split? The acceptance test rejects improvements that primarily fit evolve-split noise. The test split keeps 53 to 78 percent of the evolve-split gain (3.7 of 5.8 points on LoCoMo). Accepting every child passing $\mathcal{K}$ admits 53 to 56 of 64 children and adds 1.3 to 2.0 evolve-split points but keeps 10 to 24 percent. LongMemEval, with 100 evolve questions, retains least. Under record-disjoint splits, the margin over MemPro is 5.0 points and the evolution gain 3.4, so at most 0.6 points are record-specific (Appendix C.2).

RQ5: Does the read adapt to the question? Adaptive reads match the best fixed depth on every question type at a fraction of the tokens. No fixed policy wins everywhere. Always raw leads on single-hop questions (74.3), and stop at first hit on the other three. The adaptive read scores 74.6, 57.4, 60.1 and 52.8 on single-hop, multi-hop, temporal and open-domain questions, matching or exceeding the best fixed policy on every type and leading by 4.6 points on multi-hop, while reading 1.9k tokens against 3.3k for always raw (Figure 5, right). Under $g^\star$, 65 percent of single-hop reads reach the raw layer, whereas 72 percent of multi-hop reads stop above it.

RQ6: Are evolved architectures task-specific? Each task evolves a distinct architecture, although useful structure transfers. Each native $g^\star$ beats the best transfer by 1.5 to 2.8 points, while transfers retain 0.6 to 2.4 points over g_0, a third of the native gain (Appendix D.5).

5 Related Work

Agent memory evolution. Agent memory has moved from fixed stores toward task-adaptive structures (Hu et al., 2025c; Wu et al., 2026b). One line hand-designs paged tiers, fact stores, graph indexes, and latent memory (Packer et al., 2023; Zhong et al., 2024; Chhikara et al., 2025; Xu et al., 2025; Rasmussen et al., 2025; Kang et al., 2025; Liu et al., 2026c; Sarthi et al., 2024; Gutiérrez et al., 2024; Wu et al., 2026c; Chen et al., 2026a; Hsu et al., 2026; Wang et al., 2024b; Behrouz et al., 2024; Li et al., 2025a; Zhang et al., 2026d). Beyond hand design, a second line instead learns what to write, forget, or retrieve (Wang et al., 2025a; Sun et al., 2026; Wang et al., 2026; Jiang et al., 2026; Zhang et al., 2026a; Li et al., 2026; Chen et al., 2026b; Lin et al., 2026b; Ye et al., 2026). More recently, memory design itself is also searched (Du et al., 2026; Lu et al., 2026; Zhang et al., 2025a; Xiong et al., 2026; Liu et al., 2026b). Closest to our setting, MemPro (Liu et al., 2026d) evolves the pipeline as one program over a version tree. By contrast, `MemCodex` evolves layer programs one method at a time and keeps every derived item's exact source turns.

Program search. More broadly, program search automates agent and harness design (Hu et al., 2025a; Lee et al., 2026; Lin et al., 2026a; Zhang et al., 2026c; Wu et al., 2026a). Building on this approach, agents rewrite their own code (Schmidhuber, 2007) and keep variants by validation rather than proof (Zhang et al., 2025b; 2026b; Iacob et al., 2026; Liu et al., 2026a; Srikanth et al., 2026; Shawn, 2026). By contrast, `MemCodex` searches one method of one layer behind a fixed interface, and a child is archived only after validity checks and an acceptance test.

6 Conclusion

We propose `MemCodex`, a self-evolving hierarchical memory system that organizes experience into executable layer programs for summaries, relations, skills and latent memory. It evolves each layer through attributable per-method edits behind fixed interfaces and keeps every derived item's exact source turns. Reads traverse the hierarchy from coarse to fine and descend to source turns as needed. Through MemArena, it outperforms hand-designed and adaptive memory systems across benchmarks and backbones at lower cost. The experiments credit evidence preservation with the initial lead and per-method evolution with most of the rest, while the acceptance test carries evolve-split gains to the test split and evolved architectures remain task-specific yet transfer in part. Next, we will evolve the runtime and the layer set themselves under record-disjoint deployment.

AI USE STATEMENT

We used generative AI tools for literature search and grammar suggestions, reviewed their output carefully, and all edits were made by the authors. The authors designed, ran, and checked all experiments, configurations, and measurements.

REFERENCES

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

Han Chen, Zining Zhang, Wenqi Pei, Bingsheng He, Ming Wu, Jason Zeng, Michael Heinrich, Wei Wu, and Hongbao Zhang. MemForest: an efficient agent memory system with hierarchical temporal indexing. *arXiv preprint arXiv:2605.23986*, 2026a.

Xingyu Chen, Rui Wang, Zhaopeng Tu, and Liefeng Bo. AdaMem: learning what to remember with adaptive memory policies for personalized agents. *arXiv preprint arXiv:2606.21144*, 2026b.

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.

Lin Du, Jie Zhou, Yuxuan Cai, Kai Chen, Qin Chen, Xin Li, Bo Zhang, Wei Li, and Liang He. Automem: A text-gradient recursive self-improvement framework for automated memory architectures search. *arXiv preprint arXiv:2608.14621*, 2026.

Yiming Du, Hongru Wang, Zhengyi Zhao, Bin Liang, Baojun Wang, Wanjun Zhong, Zezhong Wang, and Kam-Fai Wong. PerLTQA: A personal long-term memory dataset for memory classification, retrieval, and fusion in question answering. In Kam-Fai Wong, Min Zhang, Ruifeng Xu, Jing Li, Zhongyu Wei, Lin Gui, Bin Liang, and Runcong Zhao (eds.), *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pp. 152–164, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.sighan-1.18/.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Yubin Ge, Salvatore Romeo, Jason Cai, Raphael Shu, Yassine Benajiba, Monica Sunkara, and Yi Zhang. TReMu: Towards neuro-symbolic temporal reasoning for LLM-agents with memory in multi-session dialogues. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18974–18988, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.972. URL https://aclanthology.org/2025.findings-acl.972/.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. arXiv:2405.14831.

Haoyu Han, Li Ma, Yu Wang, Harry Shomer, Yongjia Lei, Zhisheng Qi, Kai Guo, Zhigang Hua, Bo Long, Hui Liu, Charu C. Aggarwal, and Jiliang Tang. RAG vs. GraphRAG: A systematic evaluation and key insights. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 8966–8977, 2026. doi: 10.1145/3770855.3817575.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, Jiaxin Pei, Julian McAuley, Yejin Choi, and Alex Pentland. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

Hao-Lun Hsu, Nikki Lijing Kuang, Boyi Liu, Zhewei Yao, and Yuxiong He. Organize then retrieve: Hierarchical memory navigation for efficient agents. *arXiv preprint arXiv:2606.11680*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025a.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. *arXiv preprint arXiv:2507.05257*, 2025b.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025c.

Wei-Chieh Huang, Weizhi Zhang, Yuchen Wu, Yankai Chen, Eric Hanchen Jiang, Wooseong Yang, Yiwei Yang, Henry Peng Zou, Hanrong Zhang, Ying Nian Wu, Haolun Wu, Kai-Wei Chang, Philip S. Yu, Xue Liu, and Aylin Caliskan. Harness the memory: A holistic evaluation of memory substrates in memory agents. *arXiv preprint arXiv:2608.15008*, 2026.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Alex Iacob, Andrej Jovanović, William F. Shen, Daniel Burkhardt, Meghdad Kurmanji, Nurbek Tastan, Lorenzo Sani, Niccolò Alberto Elia Venanzi, Ambroise Odonnat, Zeyu Cao, Bill Marino, Xinchi Qiu, and Nicholas D. Lane. The red queen gödel machine: Co-Evolving agents and their evaluators. *arXiv preprint arXiv:2606.26294*, 2026.

Bowen Jiang, Zhuoqun Hao, Young-Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle Ungar, Camillo J. Taylor, and Dan Roth. Know me, respond to me: Benchmarking LLMs for dynamic user profiling and personalized responses at scale. In *Conference on Language Modeling (COLM)*, 2025. PersonaMem benchmark; arXiv:2504.14225.

Eric Hanchen Jiang, Zhi Zhang, Yuchen Wu, Levina Li, Dong Liu, Xiao Liang, Rui Sun, Yubei Li, Edward Sun, Haozheng Luo, Zhaolu Kang, Aylin Caliskan, Kai-Wei Chang, and Ying Nian Wu. Memory as a controlled process: Learned adaptive memory management for llm agents. *arXiv preprint arXiv:2607.13591*, 2026.

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

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory OS of AI agent. *arXiv preprint arXiv:2506.06326*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-End optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Wenzhi Li, Dong Nie, Rui Lan, Tongtong Lyu, Peiyao Wang, Lingzi Hong, Weihang Pan, Binbin Lin, Boyuan Pan, and Yao Hu. Dual-layer agentic memory with fast write routing and slow consolidation. *arXiv preprint arXiv:2608.22215*, 2026.

Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Huayi Lai, Hao Wu, Bo Tang, Zhengren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofen Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. MemOS: A memory OS for AI system. *arXiv preprint arXiv:2507.03724*, 2025a.

Zizhen Li, Chuanhao Li, Yibin Wang, Qi Chen, Diping Song, Yukang Feng, Jianwen Sun, Jiaxin Ai, Fanrui Zhang, Mingzhu Sun, and Kaipeng Zhang. InMind: Evaluating LLMs in capturing and applying individual human reasoning styles. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 5038–5076, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.254. URL https://aclanthology.org/2025.emnlp-main.254/.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-Driven automatic evolution of Coding-Agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026a.

Minhua Lin, Zhiwei Zhang, Hanqing Lu, Hui Liu, Xianfeng Tang, Qi He, Xiang Zhang, and Suhang Wang. MemMA: coordinating the memory cycle through Multi-Agent reasoning and In-Situ Self-Evolution. *arXiv preprint arXiv:2603.18718*, 2026b.

Changzhi Liu, Yilun Liu, Sikuan Yan, Volker Tresp, and Yunpu Ma. Mendel gödel machine: Recursive Self-Improving coding agents via comparative evolution. *arXiv preprint arXiv:2608.07645*, 2026a.

Jiaqi Liu, Xinyu Ye, Peng Xia, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. EvolveMem:Self-Evolving memory architecture via AutoResearch for LLM agents. *arXiv preprint arXiv:2605.13941*, 2026b.

Junming Liu, Yifei Sun, Weihua Cheng, Haodong Lei, Yuqi Li, Yirong Chen, and Ding Wang. Hierarchical memory orchestration for personalized persistent agents. *arXiv preprint arXiv:2604.01670*, 2026c.

Qingshan Liu, Guoqing Wang, Wen Wu, Jingqi Huang, Xinqi Tao, Dejia Song, Jie Zhou, and Liang He. MemPro: Agentic memory systems as evolvable programs. *arXiv preprint arXiv:2606.00619*, 2026d.

Mingfei Lu, Mengjia Wu, Feng Liu, Jiawei Xu, Weikai Li, Haoyang Wang, Zhengdong Hu, Ying Ding, Yizhou Sun, Jie Lu, and Yi Zhang. Choosing how to remember: Adaptive memory structures for llm agents. *arXiv preprint arXiv:2602.14038*, 2026.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.747. URL https://aclanthology.org/2024.acl-long.747/.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

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.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`. Model card, `https://huggingface.co/Qwen/Qwen3.5-27B`.

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.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*, 2024. arXiv:2401.18059.

Jürgen Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In Ben Goertzel and Cassio Pennachin (eds.), *Artificial General Intelligence*, pp. 199–226. Springer, 2007. arXiv:cs/0309048.

Zayx Shawn. PACE: Anytime-Valid acceptance tests for Self-Evolving agents. *arXiv preprint arXiv:2606.08106*, 2026.

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

Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. Recursive self-improvement of AI research agents. *arXiv preprint arXiv:2609.26457*, 2026.

Xiaolong Sun, Qichao Wang, Hangyu Li, and Liang Chen. Verifiable memory: Learning unified memory management with local and global verifiers for large language model agents. *arXiv preprint arXiv:2608.03137*, 2026.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*, 2024.

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.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024a.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. Memoryllm: towards self-updatable large language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024b.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-α: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025a.

Yubo Wang, Qiuyu Zhao, Zenghui Sun, Shichao Dong, Jinsong Lan, Xiaoyong Zhu, Haoyang Li, Bo Zheng, and Lei Chen. Cmi-mem: Toward generalizable long-term memory management via cmi-augmented reinforcement learning. *arXiv preprint arXiv:2607.20553*, 2026.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning*, pp. 63897–63911. PMLR, 2025b.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*, 2025. arXiv:2410.10813.

Siwei Wu, Jincheng Ren, Yizhi Li, Haau-Sing Li, Chengran Yang, Yuxuan Zhang, Weicheng Gu, Jian Yang, Riza Batista-Navarro, Chuanyi Zhang, Xianglong Liu, Ming Zhou, Bryan Dai, and Chenghua Lin. ModularRSI: modular and generalizable recursive harness Self-Improvement. *arXiv preprint arXiv:2609.14857*, 2026a.

Yanchen Wu, Tenghui Lin, Yingli Zhou, Fangyuan Zhang, Qintian Guo, Xun Zhou, Sibo Wang, Xilin Liu, Yuchi Ma, and Yixiang Fang. Memory in the LLM era: Modular architectures and strategies in a unified framework. *arXiv preprint arXiv:2604.01707*, 2026b.

Zhaofen Wu, Hanrong Zhang, Fulin Lin, Wujiang Xu, Xinran Xu, Yankai Chen, Henry Peng Zou, Shaowen Chen, Weizhi Zhang, Xue Liu, Philip S. Yu, and Hongwei Wang. GAM: hierarchical graph-based agentic memory for LLM agents. *arXiv preprint arXiv:2604.12285*, 2026c.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 641–649, 2024. BGE embedding models; arXiv:2309.07597.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

Yiming Xiong, Shengran Hu, and Jeff Clune. Learning to continually learn via meta-learning agentic memory designs. *arXiv preprint arXiv:2602.07755*, 2026.

Derong Xu, Yi Wen, Pengyue Jia, Yingyi Zhang, wenlin zhang, Yichao Wang, Huifeng Guo, Ruiming Tang, Xiangyu Zhao, Enhong Chen, and Tong Xu. From single to multi-granularity: Toward long-term memory association and selection of conversational agents. In *The Fourteenth International Conference on Learning Representations*, 2026. arXiv:2505.19549.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Bowen Ye, Yongchao Xu, Zhijian Li, Xiang Yin, Junkai Ma, and Wenzhao Li. CoEvo-Mem: Co-Evolving retrieval policy and memory bank for LLM agents. *arXiv preprint arXiv:2608.01739*, 2026.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025a.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025b.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026b.

Jiadong Zhang and Xiaosong Ma. Memarena: An ego-centric benchmark for on-device agentic personal memory assistants at scale. *arXiv preprint arXiv:2608.02613*, 2026.

Jiayi Zhang, Yongfeng Gu, Jianhao Ruan, Maojia Song, Yiran Peng, Zhiguang Han, Jinyu Xiang, Zhitao Wang, Caiyin Yang, Yixi Ouyang, Bang Liu, Chenglin Wu, and Yuyu Luo. Harnessing agentic evolution. *arXiv preprint arXiv:2605.13821*, 2026c.

Zeyu Zhang, Ziliang Guo, Yihang Sun, Xichong Zhang, Xixuan Hao, Zehao Lin, Yang Zhang, Xiaoyan Zhao, Tong Shen, Bo Tang, Zhi-Qin John Xu, Junchi Yan, Haofen Wang, Xu Chen, Feiyu Xiong, Zhiyu Li, and Tat-Seng Chua. Metis: Memory foundation model. *arXiv preprint arXiv:2607.26760*, 2026d.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. Lifelongagentbench: Evaluating llm agents as lifelong learners. *arXiv preprint arXiv:2505.11942*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Wei Zhou, Xuanhe Zhou, Shaokun Han, Hongming Xu, Guoliang Li, Zhiyu Li, Feiyu Xiong, and Fan Wu. Are we ready for an agent-native memory system? *arXiv preprint arXiv:2606.24775*, 2026.

A LIMITATIONS

Design and models. Fixed $\mathcal{L} = (\text{SKILL}, \text{GRAPH}, \text{SUMMARY}, \text{RAW})$ and the five-method interface restrict search to derived-layer toggles or rewriting one method. Latent memory KV MEMORY is the unevolved Metis-27B fast-weight state, supporting writes only. Its memory-attention reads require open weights (Appendix B.7), so API agents cannot use it. The append-only raw layer grows with the record because eviction changes only its active index, and deleting a source turn invalidates every exact src containing it. Each search generates 64 children, evaluates 53 to 56, and costs 87 to 118 GPU hours per benchmark (Table 10), affordable per deployment but not per user. The paired one-sided McNemar test at $\alpha = 0.05$ replaces the Gödel machine (Schmidhuber, 2007) proof and admits at most one in twenty null edits. The held-out split and three seeds reduce but do not eliminate noise fits. Architectures remain specific to the evolution backbone and judge. A new backbone requires another search, possibly from $\mathcal{A}$ rather than g_0.

Evaluation. Main results evaluate two backbones with prompts capped at 32k tokens. Qwen3.5-27B reads Metis-27B memory (Zhang et al., 2026d), whereas API model GPT-5.4-mini uses only the NL interface. One LLM judge, GPT-4o-mini (Hurst et al., 2024), grades answers. Its biases, including a preference for longer answers (Zheng et al., 2023), affect evolution and reported accuracy. The four benchmarks use English conversational and persona records. PersonaMind draws on two of the seven MEMARENA sources, PersonaMem-V2 and InMind. The main protocol partitions questions within records, so test questions come from records written during search. The record-disjoint protocol of Appendix C.2 holds out whole records and narrows the margin over MemPro from 5.6 to 5.0 points, so a small share of the gain is record-specific. Evidence recall needs gold turns, available by

annotation on two benchmarks and construction on two, but absent in deployment, which needs an annotation-free proxy for evidence recall. Because benchmarks lack external side effects, the SKILL execution check covers only side-effect-free procedures.

B Implementation Details

B.1 Write Path and Self-Modification

Parent selection. A parent is drawn with probability proportional to

$$w(n) \;=\; \sigma\Big(\tfrac{\mathrm{acc}_Q(g_n)-\overline{\mathrm{acc}}}{\hat{\sigma}}\Big)\cdot\frac{1}{1+|\mathrm{ch}(n)|}\cdot(1+d_n), \tag{4}$$

where σ is logistic, $\overline{\mathrm{acc}}$ and $\hat{\sigma}$ the archive accuracy mean and spread, $\mathrm{ch}(n)$ the children of n and d_n its normalized crowding distance on the accuracy-cost Pareto front (zero off it).

Algorithm 3 specifies the write path summarized in the method section, and Algorithm 4 specifies one child expansion in Algorithm 2, from target selection through the acceptance test. When admitting an item, the write path resolves its source turns by Eq. (2), so reads access them without walking the derivation chain. It also logs refused proposals and evicted items in a write log for the diagnosis $\mathcal{D}(g)$. Write-time reads update the access counts used by the initial index for heat. Query-time reads do not, so the store on which a child is evaluated is independent of question order.

Algorithm 3 WritePath

Input: write chunk chunk of appended raw turns, architecture g with layers $\mathcal{L}_g$, write layers $\mathcal{L}_g^{\mathrm{w}}$ (every layer up to the highest one with $\lambda = 1$), store $\mathcal{M}$ over raw layer $\mathcal{V}$, the raw layer's open segment, capacities K_ℓ, and ℓ^+ the next layer above ℓ

Output: updated store $\mathcal{M}$ and active indexes $(\mathcal{I}_\ell)_\ell$

$\mathcal{V} \leftarrow \mathcal{V} \cup \mathrm{chunk},\ \mathrm{src}(v) \leftarrow \{v\}\ \forall v \in \mathrm{chunk}$
$\mathcal{I}_{\text{RAW}} \leftarrow \mathrm{index}_{\text{RAW}}(\mathcal{V})$ ▷ no capacity
$N_{\text{RAW}} \leftarrow \mathrm{chunk},\ N_\ell \leftarrow \varnothing$ for $\ell \neq \text{RAW}$ ▷ new items per layer
for all $v \in \mathrm{chunk}$ **do** ▷ write-time reads
  for all $x^\star$ selected by Alg. 1 on $q = v$ **do**
    $m_{x^\star} \leftarrow m_{x^\star} + 1,\ t_{x^\star} \leftarrow \mathrm{now}$
for $\ell \in \mathcal{L}_g^{\mathrm{w}}$ below its top, finest first **do**
  for all $z \in \mathrm{propose}_\ell(N_\ell, \mathcal{I}_\ell)$ **do** ▷ to ℓ^+, raw also uses its segment
    $\mathrm{in}(z) \leftarrow$ items of ℓ read by z ▷ tracked handles
    if $(\ell = \text{RAW}$ **or** $\mathrm{in}(z) \cap N_\ell \neq \varnothing)$ **and** $\mathrm{admit}_{\ell^+}(z)$ **then**
      add item x of z with $\mathrm{in}(x) \leftarrow \mathrm{in}(z)$ to $\mathcal{M}_{\ell^+}$ and N_{ℓ^+}
      $\mathrm{src}(x) \leftarrow \bigcup_{y \in \mathrm{in}(x)} \mathrm{src}(y)$ ▷ Eq. (2)
    else log z as refused ▷ for $\mathcal{D}(g)$
  $\mathcal{I}_{\ell^+} \leftarrow \mathrm{index}_{\ell^+}(\mathcal{M}_{\ell^+})$
  log items evicted by index_{ℓ^+} ▷ for $\mathcal{D}(g)$
return $\mathcal{M}, (\mathcal{I}_\ell)_\ell$ ▷ $\mathcal{V}$ never shrinks

Algorithm 4 SelfModify

Input: parent node n with architecture g, rejection log $\mathcal{J}_n$ and $E_g = (\mathrm{acc}, \mathrm{tok/q}, \mathbf{v}_g, \mathcal{D}(g))$, child rank j, evolve questions Q, meta agent ρ, validity checks $\mathcal{K}$, level α

Output: child g' with $E_{g'}$ and verdict (ok, why)

$\Theta \leftarrow \mathrm{target}(\mathcal{D}(g))$ ▷ faulty pairs, then toggles
$(\ell, u) \leftarrow \Theta_j$ ▷ j: next unused rank
if $u = \mathrm{toggle}$ **then** $g' \leftarrow g$ with λ_ℓ flipped
else $g' \leftarrow \rho(g, \mathcal{D}(g), \ell, u, \mathcal{J}_n)$ ▷ override u
if not $\mathrm{ontarget}(g, g')$ **then** ▷ only (ℓ, u) differs
  return $(g', \mathrm{false}, \mathrm{target})$
for all $\kappa \in \mathcal{K}$ **do** ▷ four validity checks
  $(\mathrm{ok}, \mathrm{why}) \leftarrow \kappa(g')$ ▷ fixture queries
  if not ok **then return** $(g', \mathrm{false}, \mathrm{why})$
$E_{g'} \leftarrow \textsc{Evaluate}(g', Q)$ ▷ frozen π and judge
$(\Delta_{\mathrm{acc}}, \Delta_{\mathrm{tok}}) \leftarrow E_{g'} - E_g$
$p \leftarrow \mathrm{McNemar}(\mathbf{v}_g, \mathbf{v}_{g'})$ ▷ exact, one-sided
if Accept$(g \to g')$ **by** Eq. (3) **then**
  return $(g', \mathrm{true}, \mathrm{accepted})$
return $(g', \mathrm{false}, \mathrm{test})$ ▷ acceptance test fails

Fault localization assigns a failed trace to the first method that loses a gold turn, namely $\mathrm{propose}_\ell$ when no proposal covers the gold turn, admit_{ℓ^+} when a covering proposal is refused, index_ℓ when its item is evicted, route_ℓ when the layer stops or narrows on a top candidate whose source turns exclude the gold turn, and score_ℓ when a covering item is in scope but not ranked first and the read descends past it. Each of the four validity checks runs the child on 24 fixture queries against a frozen store snapshot. *Append-only* compares $\mathcal{V}$ before and after the child's write path. *Source exactness* requires every item the child creates to name existing inputs in the layer directly below, recomputes

its source turns by Eq. (2), and requires the result to equal the item's stored $\mathrm{src}(x)$, so the check verifies derivation rather than whether the item's content is supported by those turns. *Budget* requires every method to return within B tokens and 2 seconds. *Determinism* runs the fixtures twice and requires identical contexts. The narrowing constraint holds by construction because the runtime itself applies every scope change. The sandbox blocks reads outside a layer's own state and imports outside the standard library. A method may read its layer's active index and call the runtime's query-entity helper through read-only and deterministic interfaces. A violation raises an exception that fails the running check. A child that edits more than its target fails the target check before these checks and is counted with them in Table 10. Targets are the faulty (layer, method) pairs and toggles of layers that spent tokens in failed traces without reaching a gold turn, ranked together by fault count. When a parent is drawn several times in one round, its children continue down the same ranking.

B.2 Initial Layer Programs

The initial architecture g_0 enables every layer and implements the methods as follows. Every layer embeds with BGE (Xiao et al., 2024) and scores by cosine, making c comparable across layers. Under the routing channel, STOP spends the top candidate's source turns in time order, so a summary of a broad segment can fill much of the budget.

Initial layer programs, g_0

[raw] **# propose.** (to SUMMARY) partition the record at topic shifts, carrying the open segment across write chunks, and summarize each segment when it closes, one proposal per segment. **# index.** dense embeddings. **# score.** cosine, k=48 turns under B. **# route.** STOP.
[summary] **# propose.** (to GRAPH) extract (h, r, t, τ) assertions from admitted summaries, resolving relative dates from session timestamps. **# index.** dense embeddings, heat-based eviction to K_{SUMMARY}. **# score.** cosine, c the top cosine. **# route.** $c > 0.30 \rightarrow$ STOP, else DESCEND.
[graph] **# propose.** (to SKILL) distil a procedure from its input assertions, one demonstration per session, recurring in at least two sessions. **# admit.** resolve entities and supersede contradicted assertions. **# index.** entity graph over dense embeddings, heat-based eviction to K_{GRAPH}, superseded assertions leave the active index. **# score.** cosine with entity match. **# route.** $c > 0.55 \rightarrow$ STOP, $c > 0.30 \rightarrow$ NARROW, else DESCEND.
[skill] **# propose.** none. **# admit.** replay the body once per demonstrating session. **# index.** dense embeddings of the trigger, heat-based eviction to K_{SKILL}, superseded versions leave the active index. **# score.** cosine. **# route.** $c > 0.55 \rightarrow$ STOP, $c > 0.30 \rightarrow$ NARROW, else DESCEND.
[all] **# propose.** reads inputs one layer below through tracked handles. The runtime records these inputs and computes their source turns using Eq. (2). **# admit.** constant one unless stated.

Eviction ranks a layer's items by heat, keeps the top K_ℓ in its active index and evicts the rest. The score is a multiplicative form of the MemoryOS heat score (Kang et al., 2025), whose visit count, recency and interaction length correspond to the access count, decay and number of source turns below.

$$\mathrm{heat}(x) = \big(1 + \ln(1 + m_x)\big)\, e^{-\gamma\, \Delta t_x}\, \big(1 + \ln|\mathrm{src}(x)|\big). \tag{5}$$

Here, m_x is the access count from write-time reads, which the write path issues while it writes a chunk, and Δt_x is the conversation time since the item's last read or, if the item has never been read, since its creation. The additive offset ensures that a new item starts with nonzero heat. A broad item that is read often stays active longest.

B.3 Derived Layers and Source Turns

Proposals and derivation. A derived item is linked to the layer below through the items that its proposer reads. Lower-layer items reach a proposer only through access-tracked handles. For each write chunk, the proposer can access the items newly admitted to the layer below for that chunk together with the active index of that layer. The runtime sets $\mathrm{in}(x)$ to exactly the items whose handles the proposer read while producing proposal x, checks that every such item belongs to the layer directly below and then computes $\mathrm{src}(x)$ by Eq. (2). A proposer therefore cannot name unread inputs or declare source turns directly. Every proposal above the raw layer must read at least one item newly admitted for the current chunk, preventing repeated proposals from an unchanged group of indexed items, while the raw layer summarizes each closed segment once using the segment as its identifier. At the raw layer, the open segment is explicit program state that is carried across write chunks and passed to the proposer together with the current chunk. The task agent's backbone serves as the memory writer that produces the textual payload of each proposal. The resulting source turns record

the exact derivation of an item rather than a minimal evidence set. In particular, a summary and the assertions extracted from it share all turns in the corresponding segment. The query-adaptive read subsequently refines retrieval within this derivation through NARROW and ranking at the raw layer.

Summary layer. The SUMMARY layer creates one summary for each segment closed by a topic shift. To create the summary, its proposer reads every raw turn in the closed segment, so those turns constitute both its inputs and its source turns. An open segment remains explicit state of the raw layer until a later topic shift closes it, even when the segment spans multiple write chunks. Each summary also stores the time `t` of its segment's last turn and the set `ents` of entities it mentions, both available to reads of the SUMMARY layer. Summaries are not superseded.

Graph layer. The GRAPH layer builds a temporal entity graph from admitted summaries, following the valid-time intervals of Zep (Rasmussen et al., 2025) and the conflict-aware updates of Mem0's graph variant (Chhikara et al., 2025). For each admitted summary, the memory writer extracts typed entity mentions for people, places, objects and events, together with assertions $x = (h, r, t, \tau)$, where h and t are the stored entity mentions, r is their relation and τ is the valid time. Relative expressions such as "last Saturday" are resolved to absolute dates from the session timestamps of the source turns. Each event is linked to a node carrying its date, which allows temporal queries to order events. The method $\mathrm{admit}_{\text{GRAPH}}$ performs entity resolution and conflict detection. For entity resolution, names are normalized and embedded with BGE together with their type. A candidate is mapped to an existing canonical entity identifier only when cosine similarity is at least 0.85 and the types agree. Otherwise, a new canonical identifier is created. Each assertion retains the entity mentions with which it was extracted, while canonical entity identifiers and their mention mappings reside in a separate alias index. Consequently, an entity merge changes only the alias index and never modifies an assertion or its source turns. To detect conflicts, the memory writer retrieves active assertions whose canonical entities and relation match the new assertion and determines whether it contradicts them. Assertions are immutable, so supersession does not edit a contradicted assertion. Instead, it adds a closure record that links the old assertion to the new assertion that closed it. The closure record is persistent graph index metadata rather than a derived item, so neither a call to $\mathrm{index}_{\text{GRAPH}}$ nor eviction can remove it. It stores the identifiers of the old and new assertions, is sourced by the new assertion's source turns, and determines the effective valid interval of the old assertion together with its own τ. The superseded assertion leaves the active index but remains unchanged in the store with its own source turns. In g_0, the score combines cosine similarity between the query and the assertion text "$h\ r\ t\ (\tau)$" with 0.1 for each query entity whose canonical entity matches that of h or t. The capacity is $K_{\text{GRAPH}} = 4{,}000$ active items per record.

Skill layer. A skill contains a signature, a trigger, a body and an outcome field. The signature names the procedure and its arguments. The trigger is one sentence specifying when the procedure applies, and $\mathrm{score}_{\text{SKILL}}$ embeds this sentence. The body is Python restricted to the standard library. Each signature argument is bound to the head, relation or tail field of a named step assertion. The skill stores this binding so that every replay binds arguments deterministically. The outcome field identifies which assertion in each demonstration records the expected value. The method $\mathrm{propose}_{\text{GRAPH}}$ reads active assertions with the same head whose relations record the steps or outcome of a procedure and that recur in at least two sessions. It groups these input assertions into one demonstration per session. The memory writer then writes a skill from the demonstrations, so the skill's inputs are exactly the input assertions read for the proposal and its source turns are their union under Eq. (2). The method $\mathrm{admit}_{\text{SKILL}}$ replays the body once per demonstration in the sandbox of Appendix B.1. For each replay, it binds the arguments from that demonstration's assertions and takes the expected value from its outcome assertion. It admits the skill only if every replay returns without error within 2 seconds and its output matches the expected value after lowercasing and whitespace normalization. A newer skill with the same signature supersedes the older skill. The older version leaves the active index but remains in the store. At read time, a skill is scored by its trigger. If routing returns STOP, the routing channel adds its source turns within the current scope F to the context. These turns come from the sessions that demonstrate the procedure. The content-channel variant instead places the signature, trigger and body in the context as text. The body is not executed at read time because no benchmark provides an environment with side effects.

Eviction. Eviction removes an item only from its layer's active index and retains it in the store. Heat counts come from query-adaptive reads performed during writes with each new raw turn as the query. Its named inputs and source turns remain unchanged. Consequently, an item derived from an evicted item can still resolve its complete source turns through Eq. (2).

B.4 LAYER INTERFACE SCHEMA

Listing 1 renders the five-method layer interface as a Python-like class, with the two runtime operations of Eq. (1) beneath it and the four memory constraints as comments. Each layer program ψ_ℓ is one such class. A method rewrite by the meta agent ρ returns a subclass that overrides exactly one method body behind the signature shown here, so the write path and the query-adaptive read call the same names in every architecture. The runtime owns `write`, `read` and `assemble` together with the tracked handles, and no layer program can alter them.

```
class Layer:                    # one text layer l in (skill, graph, summary, raw)
    K: int                      # capacity K_l of the active index
    store: set[Item]            # M_l, every admitted item, never deleted
    active: set[Item]           # I_l, the items that reads and proposals may see

    def admit(self, x: Item) -> bool:
        """Reads proposal x; returns whether x enters M_l; changes nothing."""

    def index(self, store: set[Item]) -> set[Item]:
        """Reads M_l; returns the active index I_l of at most K items."""

    def score(self, q: Query, S: list[Item]) -> list[tuple[Item, float]]:
        """Reads q and S; returns ranked (x, c) pairs, the first (x_star, c)."""

    def propose(self, new: Handles, active: Handles) -> list[Item]:   # cross-layer
        """Reads this layer's new and active items by tracked handles; returns items
        for the layer above with in(x) = the handles read and src(x) = union of
        src(y), y in in(x)."""

    def route(self, q: Query, x: Item, c: float) -> Action:   # cross-layer
        """Reads q, x and c; returns Stop, Narrow or Descend; stores nothing."""

# route and propose cross layer boundaries and are the main targets of evolution:
# every evolved architecture rewrites a route method (Table 12).
Action = Literal["Stop", "Narrow", "Descend"]

# Runtime, fixed for every architecture g = (lambda, psi)
def write(M: Store, R_i: list[Turn]) -> Store:
    """Appends R_i to V, then propose, admit and index bottom-up (Alg. 3)."""

def read(M: Store, q: Query) -> Context:
    """Visits active layers coarse to fine, F = V, applying route (Alg. 1)."""

def assemble(items: list[Item], B: int) -> Context:
    """Packs items into C in temporal order under B tokens."""

# Memory constraints, enforced by the validity checks K on every child:
#   (1) append-only       turns are appended to the raw layer V and never changed
#   (2) source exactness  in(x) lies one layer below, src(x) = union of src(in(x))
#   (3) narrowing         Narrow may only restrict the scope, F <- F & src(x_star)
#   (4) budget            every context C fits within B tokens
```

Listing 1: The layer interface as a class. Every text layer implements the five methods, the runtime implements `write`, `read` and `assemble` for every architecture, and the validity checks $\mathcal{K}$ enforce the four constraints in the closing comment.

The schema is read in the order of a write followed by a read. On the write path, `propose` of the layer below produces candidates, `admit` of the current layer accepts or refuses each candidate, and `index` decides which admitted items stay active. On the read path, `score` ranks the active items in scope and `route` decides whether the read stops, narrows or descends. The methods that stay within a layer print in ink and the two that cross layers print in the accent colour. A rewrite replaces one method body and leaves every signature, the runtime and the constraints unchanged, which is what makes each accepted edit attributable to one method of one layer.

B.5 Worked Example of the Layer Hierarchy

Listing 2 traces a two-session persona record through every layer of g_0. The record holds eight raw turns. The user states a coffee preference in the first session and changes it in the second, and files an expense report in both sessions. The write path produces one summary per session, four timed assertions at GRAPH, of which the newer coffee assertion supersedes the older one, and one skill distilled from the two demonstrations of the expense procedure. For every derived item, the listing shows its identifier, its layer, its inputs $\mathrm{in}(x)$ one layer below and its source turns $\mathrm{src}(x)$ resolved to raw turn identifiers by Eq. (2). Two read traces follow, one that stops at GRAPH and one that narrows at GRAPH and descends to the raw turns.

```
# raw layer V: eight turns in two sessions, timestamps give the valid time tau
t1  [S1, Mar 2]   user:  I usually order a flat white in the morning.
t2  [S1, Mar 2]   agent: Noted, a flat white every morning.
t3  [S1, Mar 2]   user:  To file my expenses I export the receipts as one PDF,
                         attach it to a new claim in the portal and submit it.
t4  [S1, Mar 2]   agent: Done, the March claim is submitted and pending approval.
t5  [S2, Mar 16]  user:  I have switched to oat-milk cappuccinos, order that now.
t6  [S2, Mar 16]  agent: Switching your order to an oat-milk cappuccino.
t7  [S2, Mar 16]  user:  Filing this month again: receipts to PDF, new claim, submit
t8  [S2, Mar 16]  agent: Submitted, the claim is pending approval.

# summary layer: one summary per closed segment, in(s) = the turns of its segment
s1  summary  in={t1,t2,t3,t4}  src={t1,t2,t3,t4}
    "Session 1: the user orders a flat white each morning and files expenses by
     exporting receipts to a PDF, attaching it to a new claim and submitting it."
s2  summary  in={t5,t6,t7,t8}  src={t5,t6,t7,t8}
    "Session 2: the user now orders oat-milk cappuccinos and files the month's
     expenses by the same three steps; the claim is pending approval."

# graph layer: timed assertions (h, r, t, tau) extracted from admitted summaries
a1  graph  in={s1}  src={t1,t2,t3,t4}
    (user, orders, flat white, [Mar 2, Mar 16))
a2  graph  in={s2}  src={t5,t6,t7,t8}
    (user, orders, oat-milk cappuccino, [Mar 16, now])
    closure a1 -> a2: a2 supersedes a1, a1 leaves I_graph and stays in M_graph
a3  graph  in={s1}  src={t1,t2,t3,t4}
    (user, files expenses by, PDF; new claim; submit -> pending approval, Mar 2)
a4  graph  in={s2}  src={t5,t6,t7,t8}
    (user, files expenses by, PDF; new claim; submit -> pending approval, Mar 16)

# skill layer: one procedure distilled from the two demonstrations a3 and a4
k1  skill  in={a3,a4}  src={t1,t2,t3,t4,t5,t6,t7,t8}
    signature file_expenses(receipts)
    trigger   "when the user asks how to file expenses"
    body      export receipts to one PDF, attach it to a new claim, submit it
    outcome   "pending approval", replayed once per demonstration before admit

# read 1: "What coffee does the user order now?"            F = V
skill    top k1  c=0.12  route -> Descend     F unchanged
graph    top a2  c=0.71  route -> Stop        a1 is inactive, a2 is the top item
         routing channel: C <- src(a2) & F = {t5,t6,t7,t8} in time order
         answer from t5: an oat-milk cappuccino

# read 2: "Was the user's latest expense claim approved?"   F = V
skill    top k1  c=0.27  route -> Descend     F unchanged
graph    top a4  c=0.41  route -> Narrow      F <- F & src(a4) = {t5,t6,t7,t8}
summary  top s2  c=0.26  route -> Descend     F unchanged
raw      scores in scope t8 > t7 > t6 > t5,   C <- {t5,t6,t7,t8} in time order
         answer from t8: no, the claim is still pending approval
```

Listing 2: A worked example of the layer hierarchy under g_0. Each derived item lists its inputs one layer below and its source turns resolved to raw turns, and the two read traces show one STOP at GRAPH and one NARROW followed by a DESCEND to RAW.

The example makes the source recursion concrete. A summary names the turns of its segment, an assertion names one summary and inherits that segment, and the skill names two assertions and inherits both sessions, so $\mathrm{src}(k_1)$ is the union of the two demonstrations even though the skill text mentions no turn. The first read shows the routing channel at work, because the content channel would place only the text of a_2 in the context, whereas $\mathrm{src}(a_2)$ places the turn in which the user states the change. The second read shows why NARROW is confined to $F \cap \mathrm{src}(x^\star)$, since the scope inherited from a_4 excludes the first session and the raw-layer ranking then selects the turn that answers the question.

B.6 META-AGENT PROMPT, DIAGNOSIS AND AN ACCEPTED CHILD

The meta agent receives one prompt per child. The prompt names one layer and one method, and the diagnosis lists the failed traces localized to that pair.

Meta-agent prompt for one method of one layer

[system] You maintain one layer of a layered agent memory. Every layer implements the same interface, `admit`, `index`, `score`, `propose`, `route`, and you will return a subclass of the current layer that overrides only `{method}`. **# Parent.** The current program of the `{layer}` layer, with the interface it implements. **# Diagnosis.** Questions the parent answered wrongly and a sample it answered correctly, each with the methods executed, the items and turns selected, and the tokens spent, failures first. **# Rejected.** Earlier children of this parent that failed, each with the failed check or test. Do not repeat them. **# Target.** Override exactly `{method}`. State in one sentence the failure pattern you are correcting before you write the code. **# Constraints.** The raw layer is append-only. Every item you create carries the exact source turns of its inputs. A NARROW changes the search scope only by intersection with the top candidate's source turns. No read exceeds the budget. Every read is deterministic. Imports are restricted to the standard library. **# Output.** The subclass in one `python` fence and nothing else.

Diagnosis handed to the meta agent, initial SUMMARY layer on LoCoMo

[failed] Q 417: "Which adoption agency did Caroline choose?" `skill.score`→ ∅, `Descend`, then `graph.score`→ (Caroline, contacts, adoption agencies) at c=.21, `Descend`, then `summary.score`→ S 12 "Caroline researched adoption agencies" from session 9 at c=.34, |src(S_{12})|=31, `Stop`. The read spends 1,204 tokens of session 9. Gold turn v_{2081}, where Caroline states her decision, lies in session 14, outside $\mathrm{src}(S_{12})$, although active summary S_{19} of session 14 also concerns Caroline's adoption.
[failed] Q 522, Q 608, Q 731: the same pattern. A summary stops the read while a later active segment about the same entities holds the gold turn.
[hypothesis] (meta agent) The summary layer stops on the best-matching segment even when a newer segment covers the same entities, so it answers from stale evidence. STOP commits the read to $\mathrm{src}(x)$, and NARROW would intersect the scope with $\mathrm{src}(x)$, so both exclude the newer session. The read should descend and keep the scope.

Accepted child, `route` of the SUMMARY layer (round 1, LoCoMo)

```
class SummaryLayer_child(SummaryLayer_g0):
    def route(self, q, x, c):
        E = entities(q)
        if any(y.t > x.t and E <= y.ents
               for y in self.active):
            return Descend  # x is stale
        return Stop if c > .3 else Descend
```

The child overrides only `route` and reads only its own layer's active index and the runtime's query-entity helper, both of which the sandbox allows. Exact source turns let the diagnosis localize the fault to `route`, since the parent's STOP committed the read to $\mathrm{src}(S_{12})$, which provably excludes the gold turn. On Q 417, the child finds that S_{19} is newer and covers Caroline and adoption. It therefore returns DESCEND and leaves the search scope unchanged, so the raw layer ranks turns across both sessions, v_{2081} enters the context, and the task agent answers correctly. The child passes all four validity checks and is accepted on the 308 LoCoMo evolve questions. It answers 206 correctly against the parent's 196, fixing 12 questions and breaking 2 (one-sided exact McNemar $p = 0.006$), while reducing tokens per question from 2,412 to 2,356 because stale broad segments no longer fill the context. On the test split, it raises accuracy from 63.4 to 65.2 percent and reduces tokens per question from 2.31k to 2.26k.

B.7 LATENT MEMORY

Metis-27B (Zhang et al., 2026d) is an open-weight memory foundation model that inserts fast-weight memory blocks into the full-attention layers of a frozen Qwen3.5-27B. At write time, we run Metis-27B over each record, and its gated delta rule updates a fixed-size key-value state KV MEMORY by test-time training in the forward pass, without gradients. Each write chunk is committed into the state, and the state is reset for each record. At read time, we extract KV MEMORY and inject it into the Qwen3.5-27B task agent through the KV interface, where the same memory attention reads it and fuses its output with the ordinary attention branch, so later queries do not replay the original text. Because Metis-27B keeps the Qwen3.5-27B backbone frozen, the injected state is native to the task agent's representation space and requires no alignment.

KV MEMORY records the turns committed to its state. It is built on the write path, so construction contributes to write cost. For the open-weight task agent, KV MEMORY is read through the KV interface for every query. This access adds no prompt tokens, so tokens per question are unchanged, but its cost appears in latency. GPT-5.4-mini exposes no weights or activations, so it uses only the NL interface and has no latent memory.

B.8 CONSTANTS

The route thresholds of g_0 are 0.30 at SUMMARY (STOP above it) and, at GRAPH and SKILL, 0.55 for STOP and 0.30 for NARROW. The raw-layer read keeps $k = 48$ turns under a budget of $B = 4{,}096$ tokens. A write chunk is 20 appended turns. Capacities are $K_{\text{SUMMARY}} = 2{,}000$, $K_{\text{GRAPH}} = 4{,}000$ and $K_{\text{SKILL}} = 200$ active items per record, and heat decays at $\gamma = 0.1$ per day of conversation time. The loop draws $P = 4$ parents per round with replacement, asks $H = 2$ children of each, runs $T = 8$ rounds, floors the accuracy spread $\hat{\sigma}$ of Eq. (4) at 10^{-3}, and runs the acceptance test at $\alpha = 0.05$ one-sided. The meta agent runs at temperature 0.2 with one call per child.

C EXPERIMENTAL DETAILS

C.1 BENCHMARKS AND SPLITS

Table 4: Sizes of the four benchmarks by records, evolve questions, and test questions.

Benchmark	Records	Evolve	Test
LoCoMo (Maharana et al., 2024)	10	308	1,232
LongMemEval (S split) (Wu et al., 2025)	500	100	400
MemoryAgentBench (MAB) (Hu et al., 2025b)	146	420	1,680
PersonaMind (PersonaMem-V2, InMind) (Jiang et al., 2025; Li et al., 2025b)	300	300	1,200

LoCoMo's headline covers its four answerable question types (Table 4). We exclude its adversarial category because its key contains trap answers rather than recoverable facts. LongMemEval gives each question its own record of about 115k tokens. MemoryAgentBench scores its four competencies equally, and the MAB column reports their mean. PersonaMind draws 300 user records from PersonaMem-V2 (Jiang et al., 2025) and InMind (Li et al., 2025b) and poses five questions per record, 1,500 in all. Each answer must apply a stated preference or a demonstrated procedure. Before answering an evolve or test question, each system receives the question's full record through `write`.

C.2 RECORD-DISJOINT EVALUATION

The main protocol partitions questions within each benchmark, so each test question is new, but its record has been written during search and the answers to its evolve questions have been seen. To measure generalization to unseen records, we repeat the comparison with record-disjoint splits. We hold out whole records, including 2 of the 10 LoCoMo records, 29 of the 146 MAB records and 60 of the 300 PersonaMind records, while LongMemEval already pairs each question with its own record and retains its split. The evolve split contains 20 percent of the questions from the evolve records (249, 336 and 240 questions), and the test split contains every question from the held-out records (296, 420 and 300 questions). No question from a held-out record is seen at any stage. Each record is

written into a fresh store, so no memory is shared across records or users. Only the architecture g, that is, the layer programs and the layer mask, transfers from the evolve records to the held-out records. Each search-based system evolves on the evolve records under the same budget and is evaluated once on the held-out records, with three seeds as in the main protocol.

Table 5: Test accuracy under record-disjoint splits with Qwen3.5-27B, mean of three seeds, with the best per column in **bold** and the second underlined, and the change of the average against Table 1 in parentheses.

Method	LoCoMo	LongMemEval	MAB	PersonaMind	Avg.
AutoMem	59.6	57.3	45.5	52.2	53.7 (−0.4)
MemEvolve	60.0	58.0	46.1	52.9	54.3 (−0.4)
MemPro	60.7	58.4	46.6	53.5	54.8 (−0.4)
`MemCodex` (initial g_0)	63.0	59.7	47.9	55.1	56.4 (−0.4)
`MemCodex`	**65.6**	**62.9**	**51.4**	**59.3**	**59.8** (−1.0)

The margin over MemPro holds on unseen records. Table 5 reports test accuracy. `MemCodex` averages 59.8 against 54.8 for MemPro and 56.4 for g_0, so the margin narrows from 5.6 to 5.0 points and the evolution gain from 4.0 to 3.4 points, while g_0 retains its 1.6-point lead over MemPro. Relative to Table 1, MemPro and g_0 each lose 0.4 points, whereas `MemCodex` loses 1.0. Thus, about 0.6 points of its evolution gain are specific to records seen during search, while the remaining 3.4 points come from layer programs that transfer to new users. LongMemEval, whose split is record-disjoint under both protocols, is unchanged.

C.3 MemArena Protocol

MemArena is distinct from the benchmark of the same name (Zhang & Ma, 2026) and MemoryArena (He et al., 2026), which each contribute one new dataset under their own protocol. MemArena instead wraps existing sources and systems behind its data and access interfaces and metric suite. A memory wrapper exposes two calls for each memory system. `write`$(\mathcal{M}, R_i)$ delivers the next 20 turns of a record, and `read`$(\mathcal{M}, q)$ returns the text context for the NL interface together with any KV MEMORY state for the KV interface. An agent wrapper exposes `answer`$(q, C) \to a$ for each task agent. The harness runs the task agent with the same prompt for every memory system, and the judge grades its answer against the reference using the rubric of Mem0 (Chhikara et al., 2025). The harness reports four metrics. Accuracy acc is the judge's verdict on a against $a^\star$. Evidence recall $\mathrm{Rec} = \mathbb{E}_q[|G(q) \cap C|/|G(q)|]$ is the share of annotated gold turns whose text reaches the task agent's context, on the two benchmarks that annotate evidence. Tokens per question $\mathrm{tok/q}$ count the task agent's prompt tokens. Latency is the wall-clock time from `read(q)` to the last answer token, measured on one NVIDIA A100 80GB with vLLM (Kwon et al., 2023) and batch size one. Write cost, including writes to KV MEMORY, is logged separately and excluded from latency because every system writes once per record.

C.4 Baselines

Every baseline runs from its official code behind the MemArena memory wrapper, with the same backbone as task agent and as writer of any item it generates, and its default retrieval depth. Full context places the most recent 32k tokens of the record in the prompt, matching the 32k prompt cap. Dense RAG embeds the record in 512-token passages with BGE and retrieves the top passages under the `MemCodex` budget B. AutoMem, MemEvolve and MemPro search on the same evolve split with the same backbone as meta agent. Each search stops after 64 generated programs, matching the number of children `MemCodex` generates per search, and reports the program returned by its own selection rule. Mem-α trains its memory policy on the evolve split.

C.5 Ablation Variants

Table 6 reports the corresponding per-benchmark results for the variants in Table 2 and the additional variants below.

Hierarchy depth. Each depth is evolved with the same 64-child budget. Layers are added in the order raw, SUMMARY, GRAPH, and SKILL. *Summary tree* instead replaces the skill, graph, and summary layers with a RAPTOR-style tree of recursive summaries over record segments. It uses collapsed-tree retrieval, retains source turns at every leaf, and scores 56.9 accuracy with 81.5 Rec. and 2.7k Tok/q.

Evidence channel. Full `MemCodex` reads through the routing channel and places $\mathrm{src}(x^\star)$ in C on STOP. *Content channel* replaces these turns with the text of the top candidate $x^\star$ and the next nine candidates from its layer. *Content + routing* places both candidate texts and $\mathrm{src}(x^\star)$ in C.

Evolution. *No evolution* keeps g_0. *Module selection* draws each method from a menu of three predefined implementations, and the search selects an implementation for each layer. *Layer rewrites* asks the meta agent ρ to rewrite one layer program per child. *Program rewrites* asks ρ to rewrite every layer program in one child, matching MemPro. *No acceptance test* archives every child that passes $\mathcal{K}$. *No archive* expands only the latest accepted child. All search variants use the same validity checks $\mathcal{K}$ and 64-child budget.

Read depth. *Always raw* replaces every derived-layer STOP with NARROW and scores 58.3 with 90.1 Rec. and 3.3k Tok/q. *Stop at first hit* stops at the first nonempty score list and scores 56.2 with 74.6 Rec. and 1.6k Tok/q. Adaptive depth avoids the excess context of always descending and the evidence loss caused by stopping prematurely.

Latent memory. *NL interface only* removes KV MEMORY and scores 58.8. *Recall as text* retains the memory model but asks it to recall query-relevant content as prompt text, reaching 60.1 with 2.5k Tok/q. Relative to full `MemCodex` at 60.8 with 2.0k Tok/q, latent injection preserves more accuracy without adding prompt tokens.

Table 6: Per-benchmark results for every ablation variant of `MemCodex`, including LoCoMo evidence recall and prompt tokens per question. LME, MAB and PMind abbreviate LongMemEval, MemoryAgentBench and PersonaMind.

Family	Variant	LoCoMo	LME	MAB	PMind	Rec. (%)	Tok/q (k)
	`MemCodex`	67.1	62.9	52.3	60.9	88.4	2.0
Hierarchy depth	1 layer (raw)	58.9	58.7	45.4	53.4	71.2	1.8
	2 layers (+summary)	62.4	60.3	47.6	55.6	78.9	2.3
	3 layers (+graph)	65.2	61.9	50.6	58.4	84.6	2.1
	Summary tree	62.9	60.6	47.9	56.1	81.5	2.7
Evidence channel	Content channel	60.3	59.8	46.3	55.2	63.0	3.9
	Content + routing	65.8	62.0	51.2	59.4	88.7	4.6
Evolution	No evolution	63.4	59.7	48.3	55.7	82.6	2.4
	Module selection	64.8	60.9	49.6	57.7	84.9	2.3
	Layer rewrites	62.7	60.2	49.6	56.3	80.9	2.9
	Program rewrites	62.1	59.9	48.8	56.0	79.8	3.2
	No acceptance test	64.8	60.5	49.5	57.6	84.7	2.6
	No archive	65.3	61.9	50.8	58.8	86.8	2.1
Read depth	Always raw	64.2	61.5	49.6	57.9	90.1	3.3
	Stop at first hit	61.5	59.8	47.2	56.4	74.6	1.6
Latent memory	NL interface only	65.9	61.6	50.5	57.2	88.4	2.0
	Recall as text	66.8	62.4	51.6	59.6	88.4	2.5

D ADDITIONAL RESULTS

D.1 EVIDENCE RECALL

Among memory systems, evidence recall and accuracy rise together, while unstructured baselines depart from this trend (Table 7). Dense RAG counts a gold turn as delivered once its 512-token passage is retrieved, yet the task agent often misses that turn inside the passage. Full context reaches nearly every LoCoMo gold turn yet scores below `MemCodex`, which delivers 88.4 percent of them. On LongMemEval, full context has the lowest recall because its window truncates each record, while `MemCodex` delivers the most evidence at 93.7 percent.

Table 7: Evidence recall of MemCodex and the baselines on the two benchmarks with annotated gold turns, with the best per column in **bold** and the second underlined.

Method	LoCoMo	LongMemEval
Without memory structure		
Full context	**97.8**	64.5
Dense RAG	74.6	88.4
Hand-designed memory		
MemGPT	62.5	70.3
A-Mem	66.0	79.8
Mem0	68.7	81.5
MemoryOS	72.9	83.0
HMO	74.1	84.6
Adaptive and evolved memory		
Mem-α	75.8	86.1
AutoMem	77.2	87.0
MemEvolve	78.0	88.2
MemPro	79.3	89.1
Ours		
Initial g_0	82.6	90.3
MemCodex	88.4	**93.7**

D.2 LoCoMo by Question Type

Table 8: LoCoMo test accuracy of MemCodex and the baselines by question type, with the best per column in **bold** and the second underlined.

Method	Single-hop (n=673)	Multi-hop (n=226)	Temporal (n=257)	Open-domain (n=76)	All
MemPro	70.5	49.6	52.0	44.8	61.2
MemCodex (initial g_0)	72.1	51.8	55.2	47.9	63.4
MemCodex	**74.6**	**57.4**	**60.1**	**52.8**	**67.1**

Evolution gains most on multi-hop questions ($+5.6$) and temporal and open-domain questions ($+4.9$ each), and least on single-hop questions at $+2.5$ (Table 8). The larger gains occur because 72, 57 and 72 percent of multi-hop, temporal and open-domain reads stop above the raw layer under the evolved architecture, whereas 65 percent of single-hop reads reach it (Figure 5, middle).

D.3 Seed Variance

Table 9: Test accuracy of the search-based systems over three search seeds, reported as the mean and standard deviation.

Method	LoCoMo	LongMemEval	MAB	PersonaMind
AutoMem (Du et al., 2026)	60.1 ± 0.6	57.3 ± 0.5	46.0 ± 0.7	52.9 ± 0.8
MemEvolve (Zhang et al., 2025a)	60.6 ± 0.5	58.0 ± 0.6	46.6 ± 0.6	53.5 ± 0.7
MemPro (Liu et al., 2026d)	61.2 ± 0.5	58.4 ± 0.4	47.1 ± 0.6	54.0 ± 0.7
MemCodex	67.1 ± 0.4	62.9 ± 0.3	52.3 ± 0.5	60.9 ± 0.6

We repeat the search of AutoMem, MemEvolve, MemPro and MemCodex with three seeds, which vary the meta agent's sampling and the parent draws (Table 9). The standard deviation is at most 0.6 points for MemCodex and 0.8 for any baseline. Every benchmark margin of MemCodex over MemPro exceeds four standard deviations of either system. Table 1 and Figure 5 report the means, while Table 12 and Table 10 analyze the median seed.

D.4 Evolution Statistics and Cost

Table 10: Outcomes and cost of the 64 children of each search, median seed.

Benchmark	Accepted	Failed checks	Failed test	GPU hours	Meta-agent tokens (M)
LoCoMo	18	9	37	92	2.9
LongMemEval	14	11	39	118	3.3
MAB	16	8	40	104	3.1
PersonaMind	19	10	35	87	2.8

Between 14 and 19 of each benchmark's 64 children are accepted (Table 10). Most rejections come from the acceptance test because the validity checks $\mathcal{K}$ on fixture queries reject only the 8 to 11 children that violate a constraint. Table 11 shows that the acceptance test preserves 53 to 78 percent of the evolve-split gain on the test split, compared with 10 to 24 percent when every child that passes the validity checks is accepted. A search costs 87 to 118 GPU hours, dominated by evaluating children on the evolve split, and consumes 2.8 to 3.3 million tokens for the meta agent ρ. On LoCoMo, test tokens per question fall in every round, from 2.31k at the initial architecture g_0 to 1.93k at the evolved architecture $g^\star$.

Table 11: Evolve-split and test accuracy of g_0, $g^\star$ and the variant without the acceptance test, averaged over three seeds, where Ret. is the test gain over g_0 as a fraction of the evolve gain.

	g_0		$g^\star$			No acceptance test		
Benchmark	Evolve	Test	Evolve	Test	Ret.	Evolve	Test	Ret.
LoCoMo	63.6	63.4	69.4	67.1	64	71.1	64.8	19
LongMemEval	59.0	59.7	65.0	62.9	53	67.0	60.5	10
MAB	48.1	48.3	54.0	52.3	68	55.5	49.5	16
PersonaMind	56.0	55.7	62.7	60.9	78	64.0	57.6	24

D.5 Evolved Architectures

Table 12: Architectures evolved on each benchmark: active layers of $g^\star$ and methods rewritten along its chain of accepted edits.

Benchmark	Active layers	Methods rewritten
LoCoMo	all four	SUMMARY.`route`, GRAPH.`admit`, RAW.`score`, SUMMARY.`index`, SKILL.`route`, GRAPH.`propose`
LongMemEval	GRAPH, SUMMARY, RAW	RAW.`score`, SUMMARY.`route`
MAB	all four	GRAPH.`route`, SKILL.`admit`, SUMMARY.`propose`
PersonaMind	all four	SUMMARY.`index`, GRAPH.`score`, SKILL.`route`

The four searches accept different edits (Table 12). LongMemEval, whose records are each built around one question, switches SKILL off and rewrites the raw layer's scoring, while MAB and PersonaMind, whose tasks reuse procedures and preferences, keep every layer and rewrite SKILL's `admit` and `route`.

Every search rewrites a route method, and three also rewrite a write-path method. Figure 6 evaluates each $g^\star$ unchanged on every test split. Each native $g^\star$ beats the best transferred architecture by 1.5 to 2.8 points, while transferred architectures retain 0.6 to 2.4 points over g_0, about one third of the native gain. The LongMemEval architecture switches SKILL off and adds only 0.6 on MAB and 0.9 on PersonaMind, where 11 and 16 percent of reads stop at SKILL.

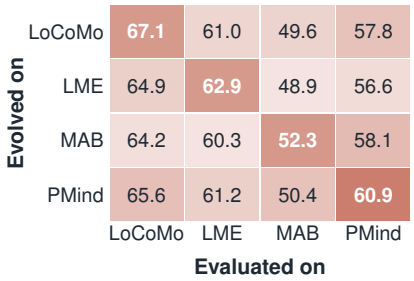

Figure 6: Transfer of evolved architectures across benchmarks, as test accuracy of each $g^\star$ on each test split, darker for larger gains over g_0 and bold on the diagonal.