WHERE SELF-EVOLVING MEMORY SYSTEMS LOSE THEIR GAINS: A FULL-LOOP AUDIT

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving memory systems use large language model (LLM) agents to evaluate performance, diagnose failures, and revise shared memory components. These loops often improve early and then plateau. Their logic rests on three assumptions: evaluation identifies failures, edits repair them, and accepted repairs persist. We audit this chain in several self-evolution methods for memory systems on two conversational memory benchmarks, using question-level evidence independent of the agent. *We find that evaluation provides unstable feedback for repair.* Reevaluating an unchanged system changes which questions fail. At least half of the accepted single-step gains fall within this variation. *Even edits that target the bottleneck stage do not prevent regressions.* These edits raise scores much more often than other edits. Yet they break previously correct answers just as often, because the components they change are shared across queries. *Across rounds, repairs do not accumulate reliably.* Repairs decline toward the level of answer flips observed in an unchanged system. Regressions, however, persist. *Addressing these weaknesses improves individual revisions but does not ensure sustained progress.* We extend MemPro, one of the audited methods, with verified feedback, repair rounds, and paired acceptance. Development accuracy rises by up to 16.9 percentage points. Yet final held-out gains are at most 3.4 points, within the original loop's rerun variation. *The remaining gains suggest that the bottleneck returns to feedback.* They are below what a development set of the original size can resolve. Sustained self-evolution therefore requires repeated evaluation and question-level tracking of repairs and regressions. Selecting revisions also requires enough evidence to distinguish gains at the intended scale from evaluation variation.

1 INTRODUCTION

As large language model (LLM) agents take on long-running conversations and tasks that span many sessions, much of what they need to know lies in earlier interactions that no longer fit in their context window (Packer et al., 2023; Maharana et al., 2024; Wu et al., 2025). To bridge this gap, agent memory systems extract information from past interactions, maintain it as persistent memory, and retrieve it for later queries (Packer et al., 2023; Xu et al., 2025; Zhou et al., 2026). However, their effectiveness depends on many coupled design choices that different workloads stress differently (Zhou et al., 2026). Self-evolving memory systems therefore hand the tuning of these choices to an LLM agent (Liu et al., 2026d;e), much as prompts and agent programs are evolved from execution feedback (Agrawal et al., 2026; Zhang et al., 2025b). In each round, the agent evaluates the system, diagnoses its failures, and revises components shared by all queries, keeping a revision only when the aggregate score improves. In practice, such loops typically gain substantially in their first few rounds and then level off (Liu et al., 2026d;e). Yet why these early gains are so difficult to sustain remains poorly understood.

To study this problem, we first distill the chain of reasoning on which these loops implicitly rely. Specifically, the loop assumes that its evaluation shows which questions fail and how well the system performs, that its diagnosis finds the cause and the revision acts on it, and that accepted revisions keep their repairs. We argue that gains are lost mainly at these three links. At the first, the evaluation may fluctuate or report failures whose cause is hard to locate. At the second, the revision may miss the cause, and even a well-aimed revision may disturb other questions. Finally, at the third, a recorded repair may be a mere flip that fails again later. Yet such losses are hard to notice from within the

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

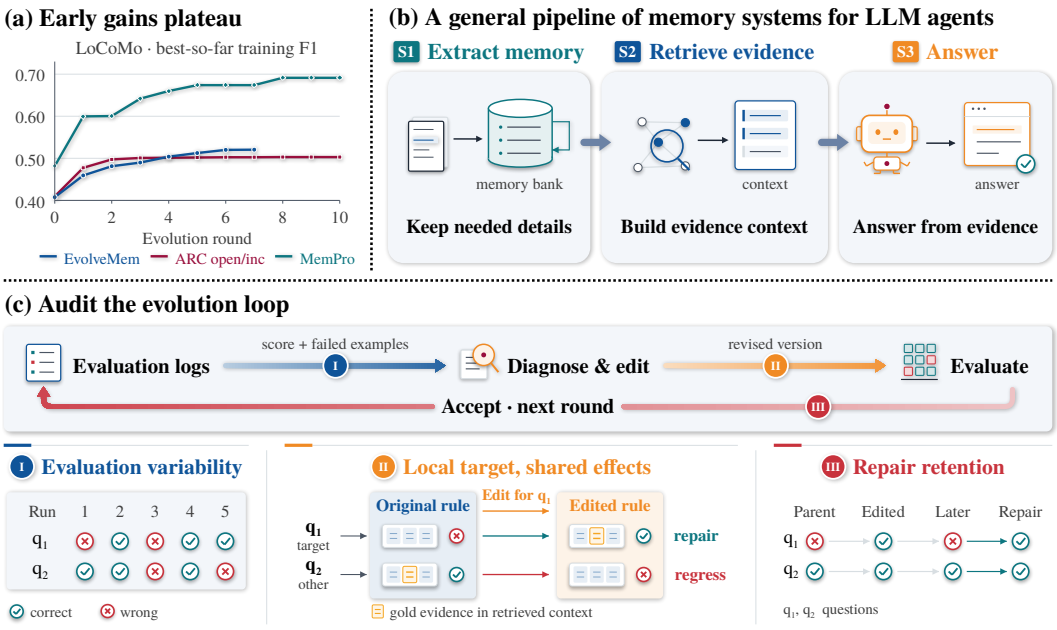

Figure 1: **Memory systems, observed evolution, and our full-loop audit.** (a) Best-so-far in-sample category 1–4 F1 on LoCoMo, shown through round 10. EvolveMem and ARC open/inc show conversation-averaged F1 as the mean over four runs of ten conversations; their search used the native mixed score. MemPro shows question-averaged training F1 from one F1-driven run (seed 2028; 154 questions). (b) The three failure stages group the memory modules of Zhou et al. (2026), with planning, integration, and context assembly in S2 and the answer prompt and answer parsing in S3. (c) The audit checks the three links marked on the loop, namely evaluation variability (I), shared edit effects (II), and repair retention (III). II illustrates how editing a retrieval rule used by all queries can restore q_1's evidence while removing q_2's, causing a regression. Only (a) contains measurements.

loop, because each round typically reduces all outcomes to a single score and a few failed examples. Studies of these loops from the outside likewise compare aggregate scores (Lin et al., 2026; Wang et al., 2026). Selection studies show that accepting every score increase on a noisy development set commits false edits (Shawn, 2026; Cawley & Talbot, 2010), but not where along the loop the gains are lost. Locating the losses therefore requires question-level evidence independent of the agent's own account.

We therefore audit the full evolution loop (Figure 1) of several self-evolution methods for memory systems on two long-conversation memory benchmarks (Maharana et al., 2024; Bei et al., 2026). For the first link, we reevaluate fixed versions, rerun entire loops, and probe failures by replay and with the annotated evidence. For the second, these interventions locate each failure at memory extraction (S1), memory retrieval (S2), or answer generation (S3), and we compare the bottleneck stage, where most attributable failures lie, with the stages targeted by the diagnosis, proposal, and executed edit. We also pair each revision with the version it modified and count the questions it repairs and breaks. For the third, we follow repairs and regressions along the sequence of accepted versions.

We find that the gains are lost at every link, and not where the loops look for them. The feedback is less stable than the loop assumes, since reevaluating an unchanged version alters which questions fail, and half or more of the accepted single-step gains lie within this variation. Hitting the bottleneck stage decides whether an edit gains but not what it breaks, since in EvolveMem and the original AutoResearchClaw loop, 17 of the 28 edits that target the bottleneck stage raise the score, against only 1 of the 42 that do not, yet both kinds break previously correct questions at a similar rate through shared components. Nevertheless, we find that stage information remains useful, as a stage-guided proposer finds a useful upgrade earlier in a bounded action space. Finally, progress is smaller than the loop records, since repairs shrink after the first rounds while regressions persist, and many recorded repairs are flips that an unchanged system also produces.

These findings suggest that a loop should verify its feedback, weigh each revision by what it breaks as well as what it repairs, and keep only repairs that hold. We therefore extend MemPro with

verified feedback, repair rounds for regressions, and paired acceptance based on stable repairs. The interventions work as intended, since the agent turns to stable failures and repair rounds recover many regressions. We find, however, that these local improvements do not accumulate, as held-out gains stay at the level the unmodified loop reaches early while development accuracy keeps rising. We find that these later gains are smaller than the resolution of a development set of the size used by the original protocol, which makes them difficult to select reliably. At the variation we measure, resolving them would take more development questions than the benchmark contains, which suggests that the bottleneck returns to the first link.

We make three contributions.

- **A full-loop audit.** We distill the chain of reasoning behind self-evolving memory loops and audit each of its three links with question-level evidence independent of the agent, namely repeated evaluation, failure stages defined by intervention, and ledgers of repairs and regressions, across several self-evolution methods for memory systems on conversational memory benchmarks.
- **Findings.** We find losses at every link. Half or more of the accepted gains lie within evaluation variation, revisions that target the bottleneck stage still break correct questions through shared components, and repairs shrink across rounds while regressions persist.
- **Improvements and their limit.** Verified feedback, repair rounds, and paired acceptance in MemPro improve local targeting and regression control, but show no held-out advantage beyond the original loop's rerun variation. We propose limited development-set evidence as one reason, since the late-stage gains that remain are smaller than a set of this size can resolve.

2 Auditing the Evolution Loop

We audit this reasoning chain in several self-evolution methods for memory systems. EvolveMem (Liu et al., 2026d) tunes the configuration of a retrieval-based memory pipeline, whereas MemPro (Liu et al., 2026e) lets its agent rewrite its own memory program. AutoResearchClaw (ARC) (Liu et al., 2026c), the autonomous research loop that discovered the Omni-SimpleMem memory system (Liu et al., 2026b) (Appendix E.2 runs it on that system), runs on EvolveMem's initial pipeline in eight configurations, which vary its action space, the version its diagnoser reads, memory rebuilding, and guards. Together with EvolveMem and MemPro, they form the ten configurations of our audit, which GEPA (Agrawal et al., 2026) joins as a prompt-evolution baseline on MemPro's pipeline. We use LoCoMo (Maharana et al., 2024) and caption-only Mem-Gallery (Bei et al., 2026) as benchmarks. On them, EvolveMem and ARC evolve on all questions and select by their native F1-based score, whereas MemPro and GEPA evolve on a random 10% of the questions, select by judge accuracy, and hold out the rest. Each configuration label names the action space, closed for the configuration only and open for the extraction and answer prompts as well, and the version the diagnoser reads, r0 for the initial version and inc for the current incumbent. Appendix B.2 details these configurations and Appendix B.4 the data splits, and Table 6 there summarizes what each loop edits, reads, and selects on. The main runs use Qwen models for the memory pipeline and the judge, and Appendix E.3 repeats the audit with GPT-4o and GPT-4o-mini as diagnoser, answer model, and ARC agent.

The evolution loop. Although these systems differ in what they edit, each round of their loops, starting from an initial version called the root, consists of evaluation, diagnosis, and execution,

$$\begin{aligned} \mathcal{E}(R) &= \big\{ y_q^{(k)}(R),\ \text{trace} \big\}_{q \in Q}, \\ (\text{claim},\ \text{proposal}) &= D\big(\mathcal{E}(R_{\text{seen}})\big), \\ R' &= A\big(R_{\text{applied}},\ \text{proposal}\big), \qquad \text{action} = \text{diff}(R_{\text{applied}}, R'), \end{aligned} \tag{1}$$

where $y_q^{(k)}(R) \in \{0, 1\}$ is the outcome of question q on version R in evaluation k (nonzero F1, or the judge's verdict), and the claim is the failure stage that the diagnosis names. An acceptance decision then selects the version that the next round builds on, which we call the incumbent. The version the diagnoser reads, R_{seen}, can differ from the parent the edit modifies, R_{applied}, and guards, no-ops, and execution deviations can separate the proposal from the action. In practice, however, the loop typically reduces the output of $\mathcal{E}$ to one score and a few failed examples. The diagnosis and the acceptance decision therefore rarely see whether an outcome y holds across evaluations, or which questions an edit has repaired and which it has broken.

Failure stages. Judging whether a claim, a proposal, or an action targets the right stage requires a reference that does not come from the diagnosis itself. We therefore define stages by intervention rather than by the system's own account. Each failure is replayed and then answered from its annotated gold evidence, from the memories extracted from that evidence, and from its retrieved context with these memories injected (Algorithm 1 in Appendix C.1). If the gold evidence turns a failure correct, the interventions assign it to one stage. It belongs to memory extraction (S1) when no memory was extracted from that evidence or its neighboring turns, or when answering from these memories still fails. It belongs to answer generation (S3) when answering from those of them already in its retrieved context turns it correct, and to memory retrieval (S2) when injecting them into the retrieved context turns it correct. The same interventions also yield two outcomes outside these stages. A *replay-correct* failure turns correct on replay, and a *gold-unresolved* failure stays wrong even with the gold evidence. The failures assigned to a stage are *attributable*. The *bottleneck stage* of a version is then the stage with the most attributable failures,

$$s^*(R) = \operatorname*{arg\,max}_{s \in \{\mathrm{S1,S2,S3}\}} \big|\{q : \, y_q(R) = 0, \, \mathrm{stage}(q) = s\}\big|, \tag{2}$$

which we leave unscored when fewer than five failures are attributable. Separating S1 from S2 requires *provenance*, the turns each memory was extracted from, and recording it causes no measurable loss in accuracy (Appendix E.1). We then compare the claim and the proposal with $s^*(R_{\mathrm{seen}})$, and the action with $s^*(R_{\mathrm{applied}})$ (Appendix F.1).

Repairs and regressions. Pointing at the right stage, however, does not show what an edit achieves. We therefore pair each child R' with the parent it actually modifies, $R = R_{\mathrm{applied}}$, and count

$$\mathcal{F} = \{q : \, y_q(R) = 0, \, y_q(R') = 1\}, \qquad \mathcal{B} = \{q : \, y_q(R) = 1, \, y_q(R') = 0\}. \tag{3}$$

On the same N questions, the change in accuracy is then exactly $(|\mathcal{F}| - |\mathcal{B}|)/N$. For the edit audit, we keep rejected revisions as well as accepted ones, whereas for retention we follow only the *evolution trajectory*, the sequence of accepted versions from the root to the final one, and in both cases reevaluations of unchanged versions provide the noise reference (Appendix B.5). With these objects in place, link I asks whether the output of $\mathcal{E}$ is stable and attributable (Section 3), link II asks whether the claim, proposal, and action point to s^* and what net change $|\mathcal{F}| - |\mathcal{B}|$ they produce (Section 4), and link III asks whether $\mathcal{F}$ survives along the accepted trajectory (Section 5). Section 6 then feeds these objects back into the loop.

3 Link I: System Variability Changes the Feedback

The first link of the chain is the evaluation $\mathcal{E}$, on which every later step of the loop builds. From each evaluation, the loop takes two signals, the failed questions that say what to repair and the score that says which version to keep (Liu et al., 2026d;e; Agrawal et al., 2026). Both uses assume that the version, evaluated again, would fail on the same questions with the same score, and that each failure exposes a defect of the system whose cause can be located. We test both assumptions on the systems' own end-to-end outputs.

Reruns change the failure set and the size of the gain. We test the first assumption at the version level, for the feedback and score change of a single step, and at the evolution level, for a loop's root-to-best reading.

At the *version level*, reevaluating a fixed version already changes which questions fail and how far the score moves (Table 1a). We measure the flip rate, the share of questions whose outcome y changes between two evaluations, and 2σ of the score difference between them. With configuration, memory, and prompts fixed, about 3% of outcomes flip in the F1-selected loops and about a tenth in the judge-selected ones. The failure list that the diagnoser reads moves more than the score, as reevaluating MemPro's roots turns a third of their failures correct while the net score moves by under 4 pp. Of the accepted steps, 50–96% change the score by less than 2σ of this variation. Such gains need not be spurious, but a loop that evaluates each revision once cannot tell them from a reevaluation of its parent (Shawn, 2026).

At the *evolution level*, rerunning the whole loop from the same root reproduces the direction of the gain but not its size (Table 1b). We report the gain of each loop's best version over the root and its

Table 1: Variation of the feedback under repetition. (a) The version each loop selected, evaluated three more times. (b) The whole loop, rerun four times from the same root. Scores are the loops' own, mixed F1 for EvolveMem and ARC in points (pts) and judge accuracy for MemPro and GEPA in pp.

		(a) Same version, 3 more evaluations				**(b) Same root, loop run 4 times**				
	Selection	Flip	2σ of	Steps within 2σ			Dev. gain		Held-out gain	
System	metric	rate (%)	score	k/n	(%)	Rounds	Mean	SD	Mean	SD
EvolveMem	F1	2.7	1.4	7/14	50	7	+8.2	1.0	+9.5	0.9
ARC closed/r0	F1	2.7	2.7	21/31	68	10	+10.8	0.8	+10.5	0.7
ARC closed/inc	F1	3.2	1.6	15/27	56	10	+11.3	0.5	+10.9	0.8
ARC open/r0	F1	2.7	1.5	9/17	53	10	+6.2	0.6	+0.9	1.9
ARC open/inc	F1	2.4	1.6	9/16	56	10	+6.6	1.6	+7.2	4.9
MemPro	Judge	8.7	4.7	22/23	96	10	+16.2	0.5	+3.8	2.0
GEPA	Judge	10.8	5.3	15/17	88	29	+5.7	1.4	+0.2	1.0

spread across four runs. Every F1-selected loop gains in every run, yet the spread of the selected versions across runs is several times that of the root, and the two open ARC variants keep a version identical to their root in some runs and gain nothing on the conversations they never saw. MemPro's development gain is stable across runs whereas its held-out gain varies with an SD of 2.0 pp, and GEPA's held-out gain changes sign. These reruns also share one extracted memory, and rebuilding it with unchanged code alone flips 17% of outcomes, five times the flips of answering twice from the same memory (Appendix E.1), so the spreads above understate the variation of a full rerun. A single root-to-best reading is therefore one draw (cf. Bouthillier et al., 2021), and Section 6 compares its arms against a rerun of the unmodified loop.

Only a subset of observed failures can be attributed. The second assumption holds only if a failure exposes a defect of the memory system whose cause can be located. We test it on the failures MemPro's loop actually receives, replaying each one and then answering it from its gold evidence (Section 2, Appendix Table 18). On LoCoMo, about a third of the root failures turn correct on replay alone, a third turn correct once the gold evidence is supplied, and a third stay wrong even with it. On Mem-Gallery, a fifth replay correct, a quarter recover with the gold evidence, and more than half either stay wrong with it or are adversarial questions that have no evidence. Only the recovered group can be traced to memory extraction, retrieval, or answer generation. Whether a gold-unresolved failure counts as wrong depends on the answer model, the prompt, and the judge rather than on the memory (Appendix E.5), yet the loop receives it like any other failure. Even under our coarse partition, then, only about a third of the failures on LoCoMo and a quarter on Mem-Gallery can be attributed to a stage, so the loop cannot tell which failures are worth repairing, and the second assumption fails as well.

Rejudging fixed MemPro answers with a second judge likewise moves a selected version's held-out gain from $+2.74$ to -0.15 pp (Appendix Table 19, cf. Thakur et al., 2025). Taken together, the loop acts on one list that mixes unchanged-system flips and gold-unresolved failures with attributable ones. The rest of the audit separates them, reading repairs and regressions against unchanged-system flips and computing s^* from attributable failures only, and Section 6 gives the same distinctions to the evolving agent.

4 Link II: Local Edits Have Wider Effects

The second link of the chain runs from the diagnosis D to the execution A, which together turn what the evaluation shows into a change of the system. Self-evolution presumes that finding why questions fail, then editing accordingly, raises the score (Shinn et al., 2023; Yuksekgonul et al., 2024; Agrawal et al., 2026). Taking the feedback as given, we compare three objects on EvolveMem, ARC closed/r0, and ARC open/inc, namely the *diagnosis* the loop writes, the *edit* it executes, and our *audit*, whose bottleneck stage on the parent's failures serves as the reference (other systems in Appendix F.1). We measure how often the three agree, what an edit repairs and breaks depending on whether it hits the bottleneck, where its regressions come from, and what a proposer that knows the bottleneck stage gains.

Table 2: Repairs, regressions, and score changes of every executed EvolveMem/ARC edit, paired with the parent it modified and grouped by whether it targets that parent's bottleneck stage. Each edit is classed against its own 2σ reevaluation band, with the classes defined in Appendix Table 21.

			Rate (%)		Class against own 2σ band				Score change		
System	Aligned	n	Repair	Regr.	Gain	In band	Offset	None	Up	Noise	Down
EvolveMem	yes	14	33.7	5.7	8	0	3	3	7	3	4
	no	33	8.4	4.7	0	2	9	22	0	22	11
ARC closed/r0	yes	14	40.2	3.6	11	0	0	3	10	4	0
	no	9	13.0	5.0	0	0	3	6	1	5	3
ARC open/inc	yes	10	40.6	7.1	7	3	0	0	7	2	1
	no	4	19.7	16.2	0	0	2	2	0	1	3

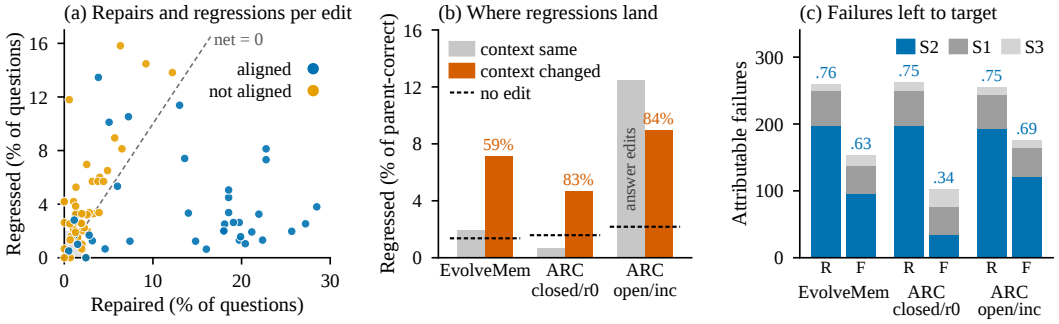

Figure 2: Where the repairs and regressions of the edits in Table 2 fall. (a) Questions repaired and regressed per edit (% of evaluated) against the parent version, with the dashed line at zero net change and the grey band at unchanged-system flips. (b) Regression rate among the parent's correct questions, split by whether the edit changes their retrieved context, with the share changed as labels and the dashed line at an unchanged system evaluated twice. ARC open/inc's unchanged-context regressions come from two answer-prompt edits. (c) Attributable failures of the root (R) and final versions (F), summed over ten conversations, with the S2 share ϕ as labels.

Proposals often miss the bottleneck. The three rarely agree (cf. Ma et al., 2024). In EvolveMem, 56 of the 67 proposals miss the bottleneck stage, and 18 of its 34 scored diagnoses follow a retrieval explanation with an answer-stage proposal. ARC closed/r0, the original ARC loop, reads the initial version by design, yet edits an evolved incumbent in 47 of its 57 rounds, and in 28 of them this stale reading is where the diagnosis first goes wrong (Appendix E.4). Three causes recur (Appendix F.1). The diagnoser cannot see the stage, whose defining provenance is absent from its input, often reads a version other than the one being edited, and proposes changes that drift from its own diagnosis, while rejected recipes return for lack of a record of attempts.

Aligned edits repair more, but regressions remain. Pairing each edit with the parent it modified (Equation 3), we report its repair rate, the share of the parent's failures it fixes, and its regression rate, the share of the parent's correct questions it breaks (cf. negative flips in Yan et al., 2021; Echterhoff et al., 2024), and we count a gain beyond noise when both its repairs and its net change exceed the edit's own binomial 2σ band. Repairs follow direction. Edits on the bottleneck repair 34–41% of their parents' failures, against 8–20% for the others, and every gain beyond noise is aligned (Table 2, Figure 2a). Read against the loops' own noise floor, 17 of the 28 aligned EvolveMem and ARC closed/r0 edits raise the score, against 1 of the 42 misaligned ones. Regressions do not follow direction. Aligned and misaligned edits break a similar share of previously correct questions, about 5% in both loops, so the two populations in Figure 2a differ in repairs alone. Hitting the bottleneck therefore makes a gain likely without lowering its price, and 14 of the 46 misaligned edits repair beyond noise yet still lose the gain to regressions.

Shared components extend edits beyond their targets. The regressions have one source. An edit aims at one failure class, but the retrieval knob, extraction template, answer prompt, or pipeline

code it changes is shared by every question, and most of the parent's correct questions receive a different retrieved context after the edit. These questions are lost at several times the rate of questions whose context is unchanged, which in turn is close to the rate of an unchanged system evaluated twice (Figure 2b), and 427 of the 499 regressions fall on changed contexts (cf. Shi et al., 2023). ARC open/inc's two answer-prompt edits lose 12.5% of correct questions through another shared component while leaving every context intact. Limiting an edit to the diagnosed category does not keep its repairs (Appendix F.1), and the failed edit aligned with the bottleneck in Appendix G breaks correct answers by the same route as a misaligned one. Combining the two effects, and assuming that repairs concentrate in the targeted pool,

$$\mathbb{E}[\Delta\mathrm{Acc}] \approx \underbrace{(1-p)\phi r}_{\text{repairs}} - \underbrace{ph}_{\text{regressions}}, \tag{4}$$

where p is parent accuracy, ϕ the targeted share of its failures, r the repair rate there, and h the regression rate among correct questions. Direction raises ϕr, whereas h comes with the shared component. The repair term, moreover, draws on a pool that drains. At the common root, S2 holds three quarters of the attributable failures, so a retrieval upgrade meets a large pool, yet by each conversation's final version the S2 failures fall by half or more in EvolveMem and ARC closed/r0 and by a third in ARC open/inc, whereas S1 and S3 failures barely change (Figure 2c). The repair term thus shrinks with its pool while the regression term stays, so repairs and regressions must be verified together at the question level (Section 6, cf. Xu & Wu, 2026).

Stage information brings the upgrade forward, but cannot extend the gains beyond it. To test whether knowing the bottleneck stage helps, we fix EvolveMem's round 0, guard, acceptance rule, and 12-recipe menu and compare three proposers, namely the upstream LLM, random menu orders, and a proposer that audits the incumbent each round and takes the next menu item for its bottleneck stage (Appendix Table 22). After seven rounds, the stage-guided proposer ends about 4 pts above both others, and its advantage comes from finding the retrieval upgrade first rather than from the later rounds. All 17 score gains of the aligned edits above likewise occur before the parent enables semantic retrieval. Within this menu, supplying the bottleneck stage thus finds the useful change earlier, and once that pool is spent, further gains depend on repair coverage and regression control, which Section 5 follows along the trajectory.

5 Link III: Repairs Do Not Reliably Persist Across Rounds

The third link of the chain is retention across rounds, since self-evolution also presumes that gains add up, in that each accepted version keeps what earlier versions repaired and adds repairs of its own (Liu et al., 2026d;e; Zhang et al., 2025b). Given that a single evaluation can misreport a version (Section 3) and that every edit breaks some correct questions whether or not it hits the bottleneck (Section 4), we follow the evolution trajectories to see how these effects compound, reading each step against the flips of an unchanged system evaluated twice, which we call no-edit flips. Table 3 reports repair and regression volumes, the questions a revision repairs or breaks against its actual parent as a share of all evaluated questions, so that, unlike the rates of Section 4, the two can be compared directly (counts and F1-scored rows in Appendix Table 23).

Repair volumes shrink while regressions persist. Repair volume falls from early to late revisions in every setting, whereas regression volume falls less or not at all in all but one setting (Table 3). As a result, late revisions regress at least as many questions as they repair in ten of the twelve settings. This is the pattern Equation 4 predicts once the early retrieval upgrades have drained much of the bottleneck-stage pool, since $(1-p)\phi r$ then shrinks whereas ph, which comes with the shared components, does not. Once the pool that direction can reach is drained, an edit to shared components thus tends to break about as many questions as it repairs.

Many recorded repairs are only flips. Late repairs approach the flips of an unchanged system (Table 3). By its late revisions, ARC closed/r0 repairs about as many questions as its unchanged configuration flips between two evaluations, MemPro's late repairs fall below that level, and GEPA's stay near it throughout. A wrong-to-correct flip is therefore weak evidence of a repair. Of the questions an unchanged system turns correct on reevaluation, about half turn wrong again on a third

Table 3: Repair and regression volumes of early and late revisions, against the no-edit flips of an unchanged system. The same six methods on both datasets, each scored by the metric it selects on. Entries are the questions repaired or regressed per revision against its actual parent, in pp of evaluated questions.

		LoCoMo						Mem-Gallery					
		Repair		Regression		No-edit flips		Repair		Regression		No-edit flips	
System	Score	E	L	E	L	Rep.	Reg.	E	L	E	L	Rep.	Reg.
EvolveMem	F1	5.0	2.9	2.4	3.3	1.0	1.0	3.5	2.2	1.3	2.7	1.0	0.7
ARC closed/r0	F1	8.7	1.2	1.8	2.3	1.1	1.3	7.2	1.0	1.3	2.5	0.3	0.4
ARC closed/inc	F1	10.9	2.8	2.7	5.5	0.6	0.8	6.8	1.2	1.6	2.1	0.5	0.0
ARC open/inc	F1	7.6	2.2	3.7	3.0	1.2	1.9	4.4	1.8	1.4	2.4	0.6	0.5
MemPro	Judge	6.7	4.7	6.1	5.6	6.0	6.1	4.7	4.0	4.5	3.9	2.3	4.1
GEPA	Judge	6.6	6.3	6.0	7.0	5.5	6.7	5.3	4.0	4.3	2.9	2.3	4.1

evaluation (Appendix F.2), and questions that a step newly repairs fail at the next update far more often than questions that were already correct (Appendix Table 24). Over a whole trajectory, this turnover dwarfs the residual. Along MemPro's trajectories on LoCoMo, about a third of the held-out questions change state at least once, yet each trajectory ends only 3–6 pp above its root, and one step flips about as many questions as a reevaluation of the root (Appendix Table 27). The residual is real, as MemPro gains on every evaluated split (Appendix Table 26), but much of what the loop records as repair is measurement variation.

Neither the regressions that outlast the repairs nor the flips recorded as repairs are visible to the loops, which evaluate each version once, count such flips as progress, and see an edit's regressions only through the net change. Section 6 therefore tests changes that count only repairs and regressions surviving reevaluation and replay, and that accept a revision by weighing them against the incumbent.

6 Local Improvements Do Not Accumulate

Sections 3–5 locate a loss at each link, and we add one change per loss to MemPro's loop. Every new version is evaluated twice and its failures are replayed at temperature zero, which marks a question *fragile* when the evaluations disagree, *ghost* when both fail but the replay succeeds, and stable otherwise. *Verified feedback* gives the agent a ledger of these labels, a *repair round* shows a separate session only the questions a revision repaired and regressed and asks it to narrow the change, and *paired acceptance* (Algorithm 2) lets the harness accept a revision only if its stable repairs outnumber its stable regressions against the incumbent (cf. Blum & Hardt, 2015; Shawn, 2026). We compare the original loop (E0) with arms that receive the ledger (E2) or the ledger and the failure stages (E3), each with and without repair rounds and paired acceptance, all from the same root and development split for ten rounds (Appendix F.4).

Each change removes the loss it targets. With the ledger, stable failures make up 96% of the agent's targets against 69% under E0, repair rounds cut stable regressions from 2.7 to 1.0 per revision without losing repairs (Appendix Table 28), and paired acceptance discards the false repairs, which without it make up 52–71% of the questions a child newly answers correctly (Table 4). These local improvements do not accumulate. Development accuracy rises 7.5–16.9 pp in every arm, yet the round-10 held-out gains of $+1.52$ to $+3.44$ pp differ less across arms than two runs of the original loop do. We find that the later gains are smaller than the resolution of the development set, which makes them difficult to select reliably. After round 5, an accepted step adds one or two net stable repairs, or about 1 pp, whereas a rerun of the unchanged root moves its score on the 154 development questions by 5.8 pp (2σ), so a loop that evaluates each version once would need more development questions than LoCoMo contains to resolve such steps (Appendices F.4 and E.6, cf. Card et al., 2020; Miller, 2024). This suggests that the plateau traces back to Link I. Since selection is itself feedback, sustaining gains would then require selection evidence at the scale of the intended gain, such as larger development sets and acceptance data withheld from proposal feedback (Dwork et al., 2015).

Table 4: Revisions and incumbents of the five arms on MemPro after ten rounds (seed 2028). E2 gives the agent the ledger, E3 adds the failure stages, and +repair adds repair rounds and paired acceptance. Targeting columns give the share of the agent's targets that are stable failures and the share of a child's newly correct questions that were flips in its parent. Regression columns count stable regressions per revision before and after its repair round. Dev. is the mean of two evaluations on the 154 development questions (root 75.97) and ΔHO the paired gain over the root on 1,386 held-out questions in pp.

	Targeting (%)		Stable regr. per revision		Incumbent, round 10	
Arm	Stable targets	False repairs	Before repair	After repair	Dev.	ΔHO
E0: MemPro loop	69	52	—	—	85.72	$+1.95$
E0 rerun	—	—	—	—	88.96	$+5.56^*$
E2: + ledger	86	66	—	—	83.44	$+1.52$
E3: + failure stages	96	71	—	—	89.29	$+2.61^*$
E2 + repair	—	—	2.7	1.0	92.84	$+3.44^*$
E3 + repair	—	—	3.0	2.7	89.29	$+3.05^*$

*Paired $z \geq 2$ against the root.

7 Related Work

Memory systems for LLM agents keep agent state outside the context window (Packer et al., 2023; Chhikara et al., 2025; Xu et al., 2025; Rasmussen et al., 2025), and recent analyses decompose benchmark scores by module to locate the bottleneck of a fixed system (Zhou et al., 2026; Yuan et al., 2026; Guo et al., 2026). **Self-improving agents** turn feedback from their own runs into lasting changes (Gao et al., 2026), and memory systems now evolve their skills, architecture, retrieval configuration, or whole program (Zhang et al., 2026b; 2025a; Liu et al., 2026d;e). Studies of this loop question its gains (Lin et al., 2026; Wang et al., 2026) and show that accepting every score increase on a noisy development set commits false edits (Shawn, 2026), yet, like the systems, they compare only aggregate scores. Since language models localize errors poorly from traces (Tyen et al., 2024; Zhang et al., 2025d), recent work on **failure attribution** relies on interventions such as counterfactual replay, fault injection, and edit-and-rerun (Zhang et al., 2026a; Liu et al., 2026a; Ma et al., 2025), and memory-specific tools attribute failures to memory operations or stages (Deng et al., 2026; Liu et al., 2026f;g). These methods, however, score attribution on fixed traces. We instead label failures by intervention on every version of a changing system, use these labels as the reference inside a running loop, and track which questions each version repairs and breaks, and whether the repairs survive. Appendix A discusses each line in more detail.

8 Conclusion

Self-evolving memory systems typically gain early and then level off, and we traced where along their loop the gains are lost. At the first link, reevaluating an unchanged version changes which questions fail, and many failures either vanish on replay or persist with the annotated evidence. At the second, even edits aimed at the bottleneck stage break correct questions through shared components. At the third, repairs shrink across rounds while regressions persist, and much of the recorded turnover is measurement variation. Verifying feedback and repairing regressions improves each revision, yet held-out gains stay where the unmodified loop leaves them, a plateau consistent with a development set too small to resolve the steps that remain. Self-evolving systems should therefore evaluate revisions more than once, weigh repairs against regressions at the question level, and size their selection evidence to the gains they seek. Our study is limited to a few self-evolution methods for memory systems on conversational memory benchmarks, its MemPro and GEPA held-out questions share conversations with the development set, and most intervention arms run once on one split (Appendix H). Whether still larger or withheld selection sets let later gains accumulate is the natural next test.

REPRODUCIBILITY STATEMENT

Section 2 and Appendix B specify the audited loops, the models in each role, the benchmarks and splits, and the measurement conventions. Appendix C gives the failure-stage probes and the acceptance rule of the intervention arms as Algorithms 1 and 2, Appendix D the prompts of the audit and of the arms, and Appendix F the per-method tables behind every pooled number of the main text. We will release the harness code, the audit scripts, and the question-level records of every evaluated version.

AI USE STATEMENT

Generative AI tools were used to assist with language revision and manuscript polishing, and to support the development and checking of mathematical arguments and proofs. In particular, AI assistance was used to improve the presentation of the paper and to help formulate, refine, and verify parts of the theoretical analysis. All AI-assisted mathematical arguments and manuscript revisions were reviewed and verified by the authors, who take responsibility for the final content

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

Yuanchen Bei, Tianxin Wei, Xuying Ning, Yanjun Zhao, Zhining Liu, Xiao Lin, Yada Zhu, Hendrik Hamann, Jingrui He, and Hanghang Tong. Mem-Gallery: Benchmarking multimodal long-term conversational memory for MLLM agents. *arXiv preprint arXiv:2601.03515*, 2026.

Avrim Blum and Moritz Hardt. The ladder: A reliable leaderboard for machine learning competitions. In *International Conference on Machine Learning*, pp. 1006–1014, 2015.

Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Naz Sepah, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Dmitriy Serdyuk, Tal Arbel, Chris Pal, Gaël Varoquaux, and Pascal Vincent. Accounting for variance in machine learning benchmarks. In *Proceedings of Machine Learning and Systems*, volume 3, pp. 747–769, 2021.

Dallas Card, Peter Henderson, Urvashi Khandelwal, Robin Jia, Kyle Mahowald, and Dan Jurafsky. With little power comes great responsibility. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 9263–9274, 2020.

Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(70):2079–2107, 2010.

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.

Xinle Deng, Ruobin Zhong, Hujin Peng, Xiaoben Lu, Yanzhe Wu, Guang Li, Buqiang Xu, Yunzhi Yao, Jizhan Fang, Haoliang Cao, Junjie Guo, Yuan Yuan, Ziqing Ma, Yuanqiang Yu, Rui Hu, Baohua Dong, Hangcheng Zhu, and Ningyu Zhang. MemTrace: Tracing and attributing errors in large language model memory systems. *arXiv preprint arXiv:2605.28732*, 2026.

Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. Generalization in adaptive data analysis and holdout reuse. *arXiv preprint arXiv:1506.02629*, 2015.

Jessica Echterhoff, Fartash Faghri, Raviteja Vemulapalli, Ting-Yao Hu, Chun-Liang Li, Oncel Tuzel, and Hadi Pouransari. MUSCLE: A model update strategy for compatible LLM evolution. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7320–7332, 2024.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026.

Minghao Guo, Qingyue Jiao, Zeru Shi, Yihao Quan, Boxuan Zhang, Danrui Li, Liwei Che, Wujiang Xu, Shilong Liu, Zirui Liu, Mubbasir Kapadia, Vladimir Pavlovic, Jiang Liu, Mengdi Wang, Yiyu Shi, Dimitris N. Metaxas, and Ruixiang Tang. MemEye: A visual-centric evaluation framework for multimodal agent memory. *arXiv preprint arXiv:2605.15128*, 2026.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory OS of AI agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25961–25970, 2025.

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.

Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, et al. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving LLM agents. *arXiv preprint arXiv:2605.30621*, 2026.

Jiale Liu, Huajun Xi, Shaokun Zhang, Yifan Zeng, Tianwei Yue, Chi Wang, Jian Kang, Qingyun Wu, and Huazheng Wang. Who&When Pro: Can LLMs really attribute failures in AI agents? *arXiv preprint arXiv:2607.09996*, 2026a.

Jiaqi Liu, Zipeng Ling, Shi Qiu, Yanqing Liu, Siwei Han, Peng Xia, Haoqin Tu, Zeyu Zheng, Cihang Xie, Charles Fleming, Mingyu Ding, and Huaxiu Yao. Omni-SimpleMem: Autoresearch-guided discovery of lifelong multimodal agent memory. *arXiv preprint arXiv:2604.01007*, 2026b.

Jiaqi Liu, Shi Qiu, Mairui Li, Bingzhou Li, Haonian Ji, Siwei Han, et al. AutoResearchClaw: Self-reinforcing autonomous research with human-AI collaboration. *arXiv preprint arXiv:2605.20025*, 2026c.

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*, 2026d.

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*, 2026e.

Xule Liu, Yijun Liu, Chao Li, and Shao Kun. D^2ACCI: A dual-loop diagnostic protocol for evidence-preserving agent memory. *arXiv preprint arXiv:2608.17756*, 2026f.

Xule Liu, Hanlin Teng, Chao Li, Yanan Ni, Shuo Lu, Audrey Wang, Yijun Liu, Yunfei Wang, Xiaofeng Li, Xian Yi, Yuanfa Li, Kang Zhao, Jian Liang, Yuxuan Chen, Jinyuan Chen, Heng Qu, Kun Shao, and Jian Luan. Mi-Memory: A lifecycle memory framework for personal AI. *arXiv preprint arXiv:2607.18975*, 2026g.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. *arXiv preprint arXiv:2512.06749*, 2025.

Ruotian Ma, Xiaolei Wang, Xin Zhou, Jian Li, Nan Du, Tao Gui, Qi Zhang, and Xuanjing Huang. Are large language models good prompt optimizers? *arXiv preprint arXiv:2402.02101*, 2024.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024. doi: 10.18653/v1/2024.acl-long.747.

Daniel Melcer, Qi Chen, Wen-Hao Chiang, Shweta Garg, Pranav Garg, and Christian Bock. Textual gradients are a flawed metaphor for automatic prompt optimization. *arXiv preprint arXiv:2512.13598*, 2025.

Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. *arXiv preprint arXiv:2411.00640*, 2024.

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.

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.

Zayx Shawn. PACE: Anytime-valid acceptance tests for self-evolving agents. *arXiv preprint arXiv:2606.08106*, 2026.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pp. 31210–31227, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. Judging the judges: Evaluating alignment and vulnerabilities in LLMs-as-judges. In *Proceedings of the Fourth Workshop on Generation, Evaluation and Metrics (GEM^2)*, pp. 404–430, 2025.

Gladys Tyen, Hassan Mansoor, Victor Cărbune, Peter Chen, and Tony Mak. LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*, 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. *Transactions on Machine Learning Research*, 2024.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents. *arXiv preprint arXiv:2607.12227*, 2026.

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

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, 2025.

Xiaonan Xu and Wenjing Wu. What aggregate scores miss: Measuring item-level regressions in commercial LLM API migrations. *arXiv preprint arXiv:2608.17719*, 2026.

Yang Xu, Jiefu Zhang, Haixiang Sun, Zihan Zhou, Tianyu Cao, and Vaneet Aggarwal. Towards reliable LLM evaluation: Correcting the winner's curse in adaptive benchmarking. *arXiv preprint arXiv:2605.05973*, 2026.

Sijie Yan, Yuanjun Xiong, Kaustav Kundu, Shuo Yang, Siqi Deng, Meng Wang, Wei Xia, and Stefano Soatto. Positive-congruent training: Towards regression-free model updates. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14299–14308, 2021.

Boqin Yuan, Yue Su, and Kun Yao. Diagnosing retrieval vs. utilization bottlenecks in LLM agent memory. *arXiv preprint arXiv:2603.02473*, 2026.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

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.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *International Conference on Learning Representations*, 2026a.

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*, 2026b.

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.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025c.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *International Conference on Machine Learning*, 2025d.

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.

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.

APPENDIX CONTENTS

A Extended Related Work **15**

B Notation and Experimental Setup **15**

B.1 Notation . . . 15

B.2 Audited loops and configurations . . . 15

B.3 Models and roles . . . 17

B.4 Datasets and splits . . . 18

B.5 Measurement conventions . . . 18

C Audit Protocol **20**

C.1 Failure-stage probes . . . 20

C.2 Label reliability . . . 20

C.3 Stage maps for claims, proposals, and edits . . . 21

C.4 Verified feedback and paired acceptance . . . 22

D Prompts **22**

D.1 Provenance contract . . . 23

D.2 Gold-evidence probe . . . 23

D.3 Stage annotators . . . 23

D.4 Feedback of the intervention arms . . . 27

D.5 Repair session . . . 30

E Additional Experiments **32**

E.1 Effect of provenance recording . . . 32

E.2 The ARC loop on the released Omni-SimpleMem system . . . 33

E.3 Model substitutions . . . 34

E.4 Diagnosis audit funnel . . . 37

E.5 Gold evidence on observed MemPro failures and judge sensitivity . . . 38

E.6 A larger development split . . . 39

F Full Tables by Method **39**

F.1 Stage agreement and edit alignment . . . 39

F.2 Repair, regression, and retention . . . 40

F.3 Held-out gains and turnover . . . 42

F.4 Intervention arms . . . 42

G Case Studies: Four Failed Edits **44**

H Limitations **45**

A Extended Related Work

Memory systems for LLM agents. Memory systems keep agent state outside the context window through tiered context (Packer et al., 2023), extraction and update pipelines (Zhong et al., 2024; Chhikara et al., 2025; Kang et al., 2025), linked notes (Xu et al., 2025), or knowledge graphs (Rasmussen et al., 2025). Benchmarks such as LoCoMo (Maharana et al., 2024), LongMemEval (Wu et al., 2025), and Mem-Gallery (Bei et al., 2026) score final answers, and recent analyses decompose these scores by module (Zhou et al., 2026; Yuan et al., 2026; Guo et al., 2026). These analyses locate the bottleneck of a fixed system, whereas we label failures by intervention on every version of a changing system and track how an edit to a shared component affects the questions it did not target (Sections 2 and 4).

Self-improving agents. Self-improving agents turn feedback from their own runs into lasting changes (Gao et al., 2026), from reflections and skills (Shinn et al., 2023; Wang et al., 2024) to prompts (Agrawal et al., 2026), workflows (Zhang et al., 2025c), harness code (Lee et al., 2026), and their own source (Zhang et al., 2025b), and memory systems now evolve their skills (Zhang et al., 2026b), architecture (Zhang et al., 2025a), retrieval configuration (Liu et al., 2026d), or whole program (Liu et al., 2026e), with one multimodal memory system discovered end to end by such a loop (Liu et al., 2026b). Studies of the loop itself find that evolver capability barely changes harness gains (Lin et al., 2026), that evolved harnesses do not consistently beat test-time scaling at matched budgets (Wang et al., 2026), and that accepting every score increase on a noisy development set commits false edits (Shawn, 2026), a known selection effect (Cawley & Talbot, 2010; Xu et al., 2026). Like the systems, these studies compare aggregate scores, whereas we follow which questions each version repairs and breaks, and whether the repairs survive (Sections 5 and 6).

Failure attribution. Language models localize errors poorly from traces (Tyen et al., 2024), and the best method on Who&When finds the decisive step of a multi-agent failure in 14.2% of cases (Zhang et al., 2025d). Recent work therefore grounds attribution in interventions such as counterfactual replay, fault injection, and edit-and-rerun (Zhang et al., 2026a; Liu et al., 2026a; Ma et al., 2025), and memory-specific tools trace operation graphs (Deng et al., 2026) or gate interventions on stage-level traces (Liu et al., 2026f;g). In prompt optimization, reflecting on pseudo errors or wrong reference answers usually yields comparable prompts (Ma et al., 2024; Melcer et al., 2025). These works score attribution on fixed traces, whereas we use replay and gold-evidence labels as the reference inside a running loop and score the diagnosis, proposal, and executed edit against the version each read or modified (Section 4).

B Notation and Experimental Setup

This section collects the notation of Section 2, the audited loops and their configurations, the model in each role, the datasets and splits, and the measurement conventions that every table of the paper follows.

B.1 Notation

Table 5 lists the symbols used in the main text and in the algorithms of Appendix C.

B.2 Audited loops and configurations

Table 6 summarizes the audited loops, and the paragraphs below define the ARC configurations and state where our runs depart from the released systems.

Configurations. The ten primary configurations consist of EvolveMem, eight ARC variants, and MemPro. Variants change the editable action space, the diagnostic reference version, memory rebuilding, or execution guards. Closed configurations may edit the configuration only, whereas open configurations may also edit the extraction and answer prompts. With r0 the diagnoser reads the initial version, as in the original ARC loop, and with inc it reads the incumbent that the loop currently keeps. The eight variants are closed/r0, the original ARC loop, and closed/inc, together with

Table 5: Notation.

Symbol	Meaning
R, R'	A version of the memory system, and the revision that an edit produces from it
R_{seen}, R_{applied}	The version whose evaluation the diagnoser reads, and the parent that the edit actually modifies
incumbent, parent, revision, proposal	The version the loop currently keeps, the version an edit modifies, the version an edit produces, and the change the diagnosis proposes
Q, q	The development questions of a run, and one question
$y_q^{(k)}(R) \in \{0, 1\}$	Outcome of q on R in evaluation k, namely nonzero F1 or the judge's verdict
$\mathcal{E}(R)$, D, A	The evaluation output of R, the diagnoser, and the editing agent (Eq. 1)
claim, proposal, action	The stage the diagnosis names, the change it proposes, and the executed diff
$\text{stage}(q) \in \{\text{S1}, \text{S2}, \text{S3}\}$	The probe-attributed stage of a failure, namely memory extraction, memory retrieval, or answer generation (Algorithm 1)
$s^*(R)$	The bottleneck stage of R, which holds the most attributable failures (Eq. 2)
$\mathcal{F}$, $\mathcal{B}$	The questions an edit repairs and breaks, respectively (Eq. 3)
2σ band	Twice the standard deviation of the score over reevaluations of an unchanged version
p, ϕ, r, h	Parent accuracy, share of failures at the bottleneck stage, repair rate there, and regression rate among correct questions (Eq. 4)
f, b, z	Stable repairs, stable regressions, and $(f - b)/\sqrt{f + b}$ in the acceptance rule (Algorithm 2)
closed, open	ARC action spaces, editing the configuration only or also the extraction and answer prompts
r0, inc	ARC diagnostic reference, the initial version or the incumbent
E, L	The first and the second half of a run's revisions in production order
E0, E2, E3	Intervention arms, namely the original MemPro loop, the loop with the ledger, and the loop with the ledger and failure stages
pp	Percentage points

Table 6: The audited loops. What each loop edits, which version its diagnoser reads, how it selects the next incumbent, and what it evolves on. All loops run on LoCoMo and on caption-only Mem-Gallery.

System	Edited object	Diagnoser reads	Selection	Evolves on	Rounds
EvolveMem	Retrieval configuration of a fixed pipeline	Last evaluated revision	Mixed F1 score	All questions of a conversation	$\leq$7
ARC closed/r0	Configuration only	Initial version	Mixed F1 score	All questions of a conversation	$\leq$10
ARC closed/inc	Configuration only	Incumbent	Mixed F1 score	All questions of a conversation	$\leq$10
ARC open/r0	Configuration, extraction and answer prompts	Initial version	Mixed F1 score	All questions of a conversation	$\leq$10
ARC open/inc	Configuration, extraction and answer prompts	Incumbent	Mixed F1 score	All questions of a conversation	$\leq$10
MemPro	Its own memory program, code and prompts	Parent it edits	Judge accuracy	Random 10% of the questions	10–15
GEPA	Five prompts of MemPro's initial pipeline	Incumbent's traces	Judge accuracy	The same 10%	15 passes

Open ARC configurations rebuild the memory bank whenever the extraction template changes and run with the execution guard on. EvolveMem's released loop is elitist, so its diagnoser reads the revision evaluated last, accepted or not, while the proposal is applied to the incumbent. MemPro and GEPA hold out the remaining 90% of the questions.

open/r0 and open/inc in three versions each, one that appends re-extracted memories to the existing bank and two that rebuild the bank whenever the extraction template changes, with the execution guard on or off. Unless noted otherwise, open/r0 and open/inc denote the guarded rebuild versions. Separate experiments add alternative models and judges, Mem-Gallery, and GEPA. ARC uses its released optimization loop on EvolveMem's weak initial memory pipeline, not a third independently designed memory architecture. Its reported memory application is Omni-SimpleMem (Liu et al., 2026b), whose naive starting point and iteration records are not released, so we run the loop on a pipeline whose every version we control and record, and Appendix E.2 additionally runs it on the released Omni-SimpleMem itself. MemPro evolves its own construction–retrieval implementation. Our analysis concerns these instantiated loops and their available edit spaces.

Correspondence with the original systems. EvolveMem and MemPro run their released code, and settings not listed here follow the original papers. EvolveMem starts from the paper's initial configuration, namely BM25 only with $k = 5$ and context budget 8, and runs round 0 plus at most seven evolution rounds, as in its reported trajectory (Liu et al., 2026d). It keeps the acceptance rule of the released code, an elitist loop that differs from the paper's Algorithm 1, and Table 23 reruns Algorithm 1 from the same round 0. Our harness replaces GPT-4o with Qwen3-max, fixes the model of each role, disables the LoCoMo-specific answer flags that the diagnoser can otherwise switch on, and records provenance during extraction (Appendix E.1). MemPro runs the released evaluation, with task temperature 0.3, at most three research iterations, and the released judge prompt, on the paper's Qwen3-30B-A3B-Instruct-2507 task model, and like the paper it evolves on a random 10% of LoCoMo and selects by judge accuracy (Liu et al., 2026e). The release contains no split, so we draw it with our own seeds. MemPro departs from the paper in its evolving agent (kimi-k3), its judge (Qwen3-max instead of GPT-4o-mini), and its decoding, since the paper decodes this model at temperature 0.7 and top-p 0.9 whereas the released code uses 0.3. At 0.7 and 0.9 the root scores 80.00 on all 1,540 questions, against 78.25 and 78.64 in two evaluations at 0.3. The released root already contains evolved components and scores above the paper's initial framework on held-out questions (78.50 against 72.35), yet its held-out gain after ten rounds, $+3.8$ pp, is close to the paper's $+4.7$. GEPA follows MemPro's baseline setting, with the same development questions and optimizer model and fifteen passes over them.

B.3 Models and roles

Table 7 lists the model in each role of the loops and of the audit. The primary EvolveMem/ARC setting uses Qwen3-max in every role, and Appendix E.3 substitutes GPT-4o or GPT-4o-mini for the diagnoser, the answer model, and the ARC agent.

Table 7: Models by role. Roles of the loops are listed first and roles of the audit second.

Role	Loops	Model
Extraction and answer model	EvolveMem, ARC, Omni-SimpleMem	qwen3-max (GPT-4o-mini answers in Appendix E.3)
Diagnoser or agent	EvolveMem, ARC, Omni-SimpleMem	qwen3-max (GPT-4o and GPT-4o-mini in Appendix E.3)
Evolving agent	MemPro; GEPA reflection	kimi-k3
Task model	MemPro, GEPA	Qwen3-30B-A3B-Instruct-2507
Selection judge	MemPro, GEPA	qwen3-max
Probe solver	EvolveMem, ARC stage probes	qwen3-max (GPT-4o-mini in Appendix C.2)
Gold-evidence probe answer models	All	qwen3-max, qwen3-30b-a3b, gpt-4o-2024-11-20, gpt-4o-mini
Stage annotator of claims and diffs	EvolveMem claims; MemPro and GEPA diffs	qwen3.8-max, two samples
Second judge for fixed-answer rejudging	MemPro held-out answers	GPT-4o-mini

Scoring. EvolveMem/ARC LoCoMo evolution scores combine category 1–4 F1 with the category-5 refusal score and are averaged equally across conversations. F1-based query flips and the main retention tables use category 1–4 zero/nonzero F1. A positive F1 indicator is not a claim of fully correct semantic answering. MemPro's original loop and GEPA select by judge accuracy, and fixed-answer rejudging applies a second judge to the same answers. Gallery analyses distinguish equal-scenario and question-weighted aggregates.

B.4 Datasets and splits

Table 8 gives the size of the two benchmarks and of the development and held-out splits, and Table 9 the scope of the records that the audits draw on.

Table 8: Datasets and splits. Development questions are drawn at random within each dataset, and held-out questions are the rest. LoCoMo development and held-out questions are disjoint but share conversation histories, and Mem-Gallery splits share scenarios in the same way.

Dataset	Units	Questions	Development	Held-out
LoCoMo	10 conversations	1,540 (categories 1–4)	154 (seeds 2027, 2028, 2029)	1,386
LoCoMo, larger split	10 conversations	1,540	308 (stratified)	1,232
Mem-Gallery, caption-only	20 scenarios	1,711	171 (seeds 2026–2029)	1,540

EvolveMem, ARC, and Omni-SimpleMem evolve on all questions of one conversation or scenario and are audited in sample, with conversation-level transfer in Table 1b. MemPro and GEPA evolve on the development split and are scored on the held-out split.

Table 9: Recorded scope of the LoCoMo audits and their extensions. Counts refer to their stated units, rather than a common pooled sample.

Setting	Recorded scope
EvolveMem	10 conversations, up to 8 rounds; 77 evaluations and 3,981 zero-score records
ARC variants	10 conversations, up to 10 rounds, eight configurations; 361 rounds
MemPro	Three split-specific trees; 45 diagnoses and 1,234 failed query–version records
Same-input pairs	4,152 EvolveMem; 6,410 ARC; 462 MemPro
GEPA extension	Four LoCoMo splits and Mem-Gallery seed 2026

GEPA updates the five prompts of the initial MemPro pipeline on the same development questions as MemPro, using kimi-k3 reflection and Qwen3-30B-A3B execution.

B.5 Measurement conventions

The conventions below fix which revisions enter each analysis, what a repetition holds fixed, and why reevaluation of an unchanged version, rather than temperature-zero decoding, serves as the noise reference.

Revision populations and ancestry. An edit is always paired with the version it actually modifies. All evaluated revisions, including rejected revisions, enter the edit analysis, whereas only accepted links enter the retention analysis. Early/late comparisons in Tables 3, 23, and 25 use the first versus the second half of each run's revisions in production order, per conversation or scenario for EvolveMem and ARC and per seed for MemPro and GEPA, with an odd extra revision counted as late. Continuous scores, binary outcomes, diff status, and acceptance decisions are stored separately. Configuration and memory hashes distinguish unchanged-system steps from actual updates. Under this definition, two ARC closed/r0 transitions that change only category overrides count as actual updates.

Repetition boundaries. Fixed-version reevaluation freezes configuration, memory, prompt, and query inputs while regenerating outcomes. Table 1a applies it uniformly, with three further evaluations of each loop's selected version. Reevaluating MemPro's three roots once at the original temperature (462 question pairs) turns 39 of their 117 failures correct and 22 of their 345 correct answers wrong, a net change of $+3.7$ pp and a flip rate of 13.2%. Evolution repetitions freeze the starting system, initial memory, problem set, and loop settings, but generate new proposal and execution samples. Each loop has four repetitions. For EvolveMem and the four ARC variants, the per-conversation standard deviation of endpoint scores is 1.3–3.3 pts, compared with 0.4–0.5 pts at the initial version, and the signs of conversation-level gains do not change across repetitions. MemPro's and GEPA's held-out gains are measured against the reevaluated root (78.50%), with development-score ties going to the earlier version. These endpoints have one held-out evaluation each, so their spread includes evaluation noise.

Decoding temperature. The flips of an unchanged version are not an artefact of the sampling temperature. MemPro's released evaluation decodes its task model at temperature 0.3 while its judge already runs at temperature 0, and setting all six task-model calls to temperature 0 does not make a version reproducible on the model service we use (Table 10). The variation originates in the service, since five calls with an identical 9,000-character prompt at temperature 0 return five distinct outputs from Qwen3-30B-A3B, with or without a fixed seed, diverging within the first 50 characters, and Qwen3-max returned five distinct outputs on one day and two on another. We therefore keep the released temperature and use reevaluation of the unchanged version as the noise reference throughout.

Table 10: Two evaluations of the same MemPro root at temperature 0 and at the released temperature 0.3. Questions are those of the 308-question split, and the last column gives the share of questions whose answer text is identical in both evaluations.

Evaluation pair	Questions	Outcomes flipped	Identical answers
Temperature 0, twice	78	12 (15.4%)	42%
Temperature 0.3, pairs	78	7.7–15.4%	44%
Temperature 0.3, pairs	308	7.8–11.0%	—

Notes to the main-text tables. Table 1a evaluates each loop's selected version three more times, namely the best round per conversation for EvolveMem and ARC and the selected version of the seed-2028 tree for MemPro and GEPA, on the loop's own questions with the loop's own code. Flips and 2σ come from the three pairs, and 2σ is empirical over conversations for EvolveMem and ARC and follows from per-question score differences for MemPro and GEPA. Accepted steps are accepted revisions for EvolveMem and ARC, the steps on the selected versions' lineages of the three LoCoMo splits for MemPro (23 steps, read against the band of the seed-2028 version), and the steps on the selected version's lineage for GEPA. In Table 1b, the best version is the one each loop selects by its own development score, with ties going to the earlier version, and MemPro and GEPA measure held-out gains on 1,386 questions disjoint from development. Table 2 reads score changes against the ± 0.011 noise floor (2σ) of the same-configuration reevaluations recorded by the loops, and excludes edits without a change or with fewer than five attributable parent failures. In Table 3, the halves are taken per conversation or scenario for EvolveMem and ARC, per split for MemPro, whose four splits run fifteen rounds each with rounds 1–7 early and 8–15 late, and per split for GEPA, with four LoCoMo splits and one Mem-Gallery split and fully evaluated revisions only, and the no-edit flips of MemPro and GEPA come from their shared root evaluated repeatedly. The rounds entry of GEPA in Table 1b is its mean number of reflective iterations per run, and the held-out gains of EvolveMem and ARC there are those of the configuration selected on one conversation on the roots of the other nine conversations (1,787 questions). In Table 3, the no-edit flips of ARC closed/inc on LoCoMo and of MemPro and GEPA on Mem-Gallery rest on fewer than five evaluation pairs, and the open action space of ARC open/inc on Mem-Gallery adds extraction templates and re-extraction but not answer prompts. In Table 4, the E0 rerun is an independent run of the original loop, the same run as in Table 1b, and its development score comes from one evaluation. The E0 arm of Table 4 reports the incumbent that the loop itself keeps at round 10 (v0009), whereas the MemPro row of Table 1b applies the tie rule above to the same run and selects the earlier v0007.

C AUDIT PROTOCOL

This section specifies the failure-stage probes of Section 2 as an algorithm, reports how reliable their labels are, defines how claims, proposals, and edits are mapped to stages, and gives the acceptance rule of the intervention arms of Section 6. The prompts that the probes and the arms use are in Appendix D.

C.1 FAILURE-STAGE PROBES

Algorithm 1 attributes an observed failure to a stage by a sequence of interventions on the answer context, each of which changes one stage at a time and keeps the system's own answer model, prompt, and context format. A replay-correct outcome is retained as a separate feedback category, and a failure that the gold evidence itself does not resolve is ineligible for stage attribution. Among eligible failures, the probes distinguish information missing from memory extraction (S1), information stored in memory but not successfully prepared for the answer (S2), and failure of the final answer interface despite the required evidence being present (S3). Two interventions keep provenance from deciding a stage alone. A failure goes to S3 only when answering from the gold-linked memories already in its answer context turns it correct, and to the injection probe otherwise. Since an extractor may cite a turn next to the annotated one, the memories that cite a turn within two positions of a gold turn in the same session are used when the directly linked memories are missing or fail. A probe answer counts as correct at a token F1 of at least 0.5, and the two interventions change 4.9% of the EvolveMem and ARC labels. Injection failures not resolved by the protocol remain unresolved.

Algorithm 1 Stage attribution of one failed question

Require: version R, question q with $y_q(R) = 0$, gold turns G_q, memory bank $M(R)$ with provenance, answer context $C_q(R)$

1: replay q on R; **if** correct **then return** REPLAY-CORRECT
2: answer q from G_q with the systems own answer prompt; **if** wrong **then return** INELIGIBLE
3: $M_g \leftarrow$ memories of $M(R)$ that cite a turn of G_q, or, when none does, memories that cite a turn within two positions of one in the same session
4: **if** $M_g \cap C_q(R) \neq \emptyset$ and answering from $M_g \cap C_q(R)$ is correct **then return** S3
5: **if** $M_g = \emptyset$ **then return** S1 ▷ no memory holds the gold fact
6: replace the tail of $C_q(R)$ with M_g (at most half of the retrieval depth) and answer; **if** correct **then return** S2
7: **return** UNRESOLVED

EvolveMem/ARC use this strict protocol for the main stage comparisons. For MemPro, a probe that reruns integration and the research loop can label an unsuccessful injection as S3 even when final-answer evidence has not been isolated. The MemPro/GEPA sequential audit therefore assigns planning, retrieval, integration, sufficiency checks, and continued retrieval to S2, with S3 starting from the final evidence-summary answer interface, and applies the same order of checks, namely gold evidence first, then all persistent memories that share provenance with the gold turns, then the original integrated memory. It uses three consistent answers as the primary probe criterion, with two-of-three as a sensitivity analysis. This audit has 2,809 completed question-level records.

C.2 LABEL RELIABILITY

Table 11 reports three checks of the probe labels, namely repeated probing of the same failures, a second probe solver, and a manual audit, and Table 12 repeats the stage readings of the main text under the judge verdict.

Repeated labels and a second solver. Repeated labels agree at 0.90 on EvolveMem and ARC records. Under the looser MemPro boundary, which lets a probe rerun integration and the research loop, repeated labels agree on only half of the individual labels, which motivates the sequential boundary of Appendix C.1. With GPT-4o-mini as probe solver, record-level agreement is lower than round-level agreement, since the two solvers attribute different records but name the same bottleneck stage in nearly every round.

Table 11: Agreement of the probe labels under repeated probing, a second probe solver, and a manual audit. Bottleneck-stage rows compare the stage that holds most attributable failures per round.

Check	Records	Agreement
Repeated probe labels, EvolveMem	293	0.90
Repeated probe labels, ARC	175	0.90
Repeated probe labels, MemPro under the looser boundary	20	10/20 individual, 12/20 four-way
GPT-4o-mini probe solver against Qwen3-max, LoCoMo	739 sampled	0.704
records attributable under both solvers	145	0.883
bottleneck stage per round	16 rounds	14/16
GPT-4o-mini probe solver against Qwen3-max, Mem-Gallery	319	0.871
records attributable under both solvers	68	1.000
bottleneck stage per round	5 rounds	5/5
Manual audit against the probe, EvolveMem and ARC	92	0.86

Manual audit. Repeated probes show that the labels are stable, not that they name the stage a reader would name. We therefore audit 92 EvolveMem and ARC failures by hand, drawn at random within each probe label. The annotator sees the question, the gold answer and its turns, the conversation, the stored memory without provenance, and the prompts and outputs of the answer model, but no probe result, and decides whether the memory holds the needed fact and whether the answer context holds it. The probe stage matches the manual stage on 86% of the failures that both assign to a stage, with each probe label weighted by its share of all failures.

Verdict rule. The F1 threshold decides both which loop answers count as failures and which probe answers count as correct. Table 12 repeats the stage readings with the released LoCoMo judge (Liu et al., 2026e) in both places, with and without the two interventions. The judge passes answers that match the gold answer in meaning, so fewer failures stop at the gold-evidence probe and the S1 share falls. Every variant keeps the readings of Section 4, as aligned edits repair more of their parents' failures than misaligned ones, nearly all gains beyond noise are aligned, S2 is the root bottleneck in all ten conversations, and most EvolveMem proposals miss the bottleneck stage. The main text uses the F1 verdict with both interventions, since F1 is the metric the loops optimize.

Table 12: Stage readings of EvolveMem, ARC closed/r0, and ARC open/inc under the F1 and the judge verdict, with and without the two added interventions. Shares give S1, S2, and S3 among attributable failures, pooled over distinct question–version instances, and root ϕ gives the S2 share at EvolveMem's round 0. Repair gives the mean repair rate of aligned and of misaligned edits as a range over the three systems, gains count the aligned gains beyond noise among all such gains, and missed counts the EvolveMem proposals, out of 67, that miss the bottleneck stage. The judge rows use the judge for failures, probes, repairs, and regressions.

Verdict	Interventions	Shares	Root ϕ	Repair (%)	Gains	Missed
F1	original	.32/.55/.14	0.68	35–41 / 9–20	26/26	53
F1	added (main text)	.26/.62/.12	0.76	34–41 / 8–20	26/26	56
Judge	original	.25/.60/.15	0.74	33–43 / 8–18	27/29	51
Judge	added	.18/.68/.14	0.82	28–42 / 0–23	26/29	46

C.3 Stage maps for claims, proposals, and edits

The stage checks of Section 2 compare a claim, a proposal, and an executed action with the bottleneck stage. This subsection defines how each of the three is read as a stage and checks the readings against an independent annotator.

Stage maps. ARC states a stage in its own diagnosis output, whereas EvolveMem's stated stage is read from its free-text root causes by an LLM judge (qwen3.8-max, two samples), which excludes claims naming all three stages. Proposed and executed EvolveMem and ARC changes pass through an operator table. Extraction templates and re-extraction map to S1, answer prompts, answer-format

flags, and verification map to S3, and retrieval and context settings map to S2. A few fields that act on several stages, such as per-category overrides, map to a stage set, and a change to memory alone counts as S1. MemPro and GEPA diffs are annotated by qwen3.8-max against the S2/S3 boundary of Appendix C.1.

Map checks. The operator table is checked against an independent reading. qwen3.8-max labels each field from its documentation line alone, agrees with the table on every field it labels, and abstains on four. Replacing the table by this judge map leaves the stage of all 47 EvolveMem and 24 ARC executed actions unchanged, so the action rows of Tables 2 and 21 do not depend on the table. The judge reading of EvolveMem proposals agrees with the table on 62 of 67 rounds, the five differences being S2 read as S2+S3, and the agreement in Table 20 is unchanged. The judge reading of ARC diagnosis text matches the loop's own stated stage in all 57 rounds. For MemPro and GEPA, the annotator sees only the parent–child diff, not scores, gold evidence, acceptance, or the diagnosis text. Its two samples agree on 112 of 119 edits and match a rule baseline on the same 112, and the remaining seven are kept as unresolved rather than assigned a stage.

C.4 Verified feedback and paired acceptance

Algorithm 2 gives the rule by which the repair arms of Section 6 classify development questions and accept a version. The rule is an operational way to focus the gate on repeat-supported differences. It does not make repeated agreement a guarantee of ground-truth correctness or permanent retention, and it protects its measured transition criterion rather than forcing the raw two-evaluation mean to be monotone.

Algorithm 2 Verified feedback and paired acceptance of a revision

Require: incumbent R, revision or repair child R', development questions Q

for all $V \in \{R, R'\}$ **do**
  $A_V \leftarrow$ evaluation of V on Q; $B_V \leftarrow$ a second, complete evaluation of V on Q
  $T_V \leftarrow$ temperature-0 replay of the questions wrong in A_V
  for all $q \in Q$ **do**
    $\text{class}_V(q) \leftarrow$ FRAGILE if $A_V(q) \neq B_V(q)$; STABLE-CORRECT if both correct; GHOST if both wrong and $T_V(q)$ correct; STABLE-WRONG otherwise
  end for
end for
$f \leftarrow |\{q : \text{class}_R(q) \in \{\text{STABLE-WRONG}, \text{GHOST}\},\ \text{class}_{R'}(q) = \text{STABLE-CORRECT}\}|$
$b \leftarrow |\{q : \text{class}_R(q) = \text{STABLE-CORRECT},\ \text{class}_{R'}(q) \in \{\text{STABLE-WRONG}, \text{GHOST}\}\}|$ ▷ fragile questions are excluded
$z \leftarrow (f - b)/\sqrt{f + b}$
R' passes the gate **iff** $f - b \geq 1$ and $z \geq 1$; among versions that pass, keep the largest $f - b$

Repair rounds. The E2+repair and E3+repair arms add a repair session and harness-controlled acceptance to the corresponding ledgers. A repair child is registered from its revision, and the agent sees repaired and regressed development queries and changed files, and is asked to preserve beneficial behavior while narrowing or reverting the regressing change. Per-query or gold-based special casing is prohibited, with diffs retained for checks. The repair-specific part of the prompt adds no stage labels, and the E3+repair arm keeps the stage note of E3. Revision and repair child are compared with the actual incumbent by Algorithm 2, whose threshold is operational rather than a significance test.

D Prompts

This section prints, verbatim, the prompts that the audit and the intervention arms add to the released systems. Placeholders in braces are filled at run time. The prompts of the released loops themselves are unchanged and are not repeated, and MemPro's answer judge is the released prompt of Liu et al. (2026e).

D.1 PROVENANCE CONTRACT

The EvolveMem harness prefixes every dialogue turn of an extraction window with a turn identifier and appends the following contract to the extraction request. A reply without valid identifiers is rejected and regenerated. The ARC harness on the released Omni-SimpleMem appends a contract of the same form with its own wording.

Provenance contract appended to every EvolveMem extraction request

```
[PROVENANCE OUTPUT CONTRACT -- REQUIRED]
 Every memory object MUST include "source_dgids": ["exact DGID", ...].
Select all dialogue turns that support that memory from the [DGID=...]
labels in the current dialogue window. Return a nonempty list of unique
 exact IDs. Do not cite previous-window memory context, invent IDs, or
copy DGID labels into lossless_restatement, keywords, persons, entities
 or topic. Session identity is assigned by code; do not generate a
session ID. Example object shape: {"lossless_restatement":"...", "
keywords":[], "timestamp":null, "location":null, "persons":[], "
entities":[], "topic":"...", "source_dgids":["lc::TENANT::D1"]}. A
response missing source_dgids is invalid and must be regenerated.
```

Provenance contract of the ARC harness on Omni-SimpleMem

```
[PROVENANCE OUTPUT CONTRACT - REQUIRED]
Every memory entry MUST include "source_dgids": ["exact DGID", ...].
Select all dialogue lines that support that entry from the [DGID=...]
labels in the current window dialogues. Return a nonempty list of
unique exact IDs. Do not cite the previous-window entries, invent IDs,
or copy DGID labels into lossless_restatement, keywords, persons,
entities or topic. Example entry shape: {"lossless_restatement":"...",
"keywords":[], "timestamp":null, "location":null, "persons":[], "
entities":[], "topic":"...", "source_dgids":["lc::conv-26::D1:3"]}. A
response missing source_dgids is invalid and must be regenerated.
```

D.2 GOLD-EVIDENCE PROBE

The dataset-level gold-evidence probe of Appendix E.5 sends the following system and user messages once per question and answer model at temperature 0, with the annotated gold turns as the only evidence. The per-failure probe of Algorithm 1 instead answers from the gold turns with the audited system's own answer prompt.

Gold-evidence probe, system message

```
You answer questions about a person's past conversations. Use only the
evidence provided. Answer as briefly as possible: a short phrase, a
date, a name, a number, or yes/no. Only if the evidence contains
nothing relevant to the question, reply exactly: Not mentioned.
```

Gold-evidence probe, user message

```
Evidence:
{evidence}

Question: {question}
Answer:
```

D.3 STAGE ANNOTATORS

The stage annotator of Appendix C.3 reads each EvolveMem diagnosis with the system message and the user message below, two samples at temperature 0, and a stage set is kept only when both samples parse and agree. The ARC diagnosis form and the configuration-field form replace the subject of the user message and follow the same stage definitions. MemPro and GEPA revisions are annotated from

their parent-to-child diff alone with the edit prompt, and MemPro's diagnosis texts with the claim prompt.

Stage annotator, system message

```
You are a careful annotator. Answer with a single JSON object and
nothing else.
```

Stage annotator, user message for EvolveMem root causes

```
The memory system answers questions about a long multi-session
conversation (LoCoMo) in three stages. A failure is attributed to the
stage where the answer was lost.

S1 memory_formation (extraction / writing): an LLM reads the
conversation session by session and writes a bank of memories, each
with content, keywords, persons, entities, timestamp and topic. This
stage is fixed during the optimisation. A failure belongs to S1 when
the fact needed for the answer never made it into the bank, or was
written in a form that does not answer the question (paraphrased away,
wrong date, merged with other facts).

S2 retrieval (selection / ranking): for each question the system
searches the bank with a BM25 keyword view, a dense (BGE) semantic view
 and structured indices over persons / entities / topics, fuses the
views (keyword_only, weighted_sum, rrf, with per-view top-k and weights
), optionally decomposes the question into sub-queries or plans intents
, optionally reranks for diversity (MMR), and keeps at most max_context
 memories. A failure belongs to S2 when the right memory exists in the
bank but is not in the retrieved list, is ranked below the context cut-
off, or is crowded out by irrelevant memories.

S3 answering (generation): the answer LLM receives the question and the
 retrieved memories formatted as context (up to 25 items), under an
answer prompt / style (concise, explanatory, verifying), optionally
followed by a verification pass; the output is scored with token F1
against the gold answer. A failure belongs to S3 when the needed memory
 is in the context but the model still answers wrongly, abstains ('Not
mentioned'), gives a vague or incomplete answer, or answers in a form
that does not match the gold answer.

Below is the diagnosis an optimisation LLM returned, as JSON, for one
evaluation round of the memory system: three free-text cause
descriptions (their bucket labels are withheld because the LLM was
required to fill all three buckets every round) and its
priority_actions lines. Its parameter suggestions and recipe names are
withheld because they describe actions, not causes.

cause_1: {cause_1}
cause_2: {cause_2}
cause_3: {cause_3}
priority_actions: {priority_actions}

Task: record the stage or stages that the author explicitly blames for
the failures. This is a transcription task, not a diagnosis: do not
decide which stage is actually at fault, and do not complete the author
's reasoning for them. Read all four lines; each line can name a cause.

What counts as naming a stage as the cause: a phrase that says the
stage's own output was insufficient, irrelevant, shallow, misaligned,
missing or wrong. Examples that count: 'insufficient or irrelevant
retrieval', 'retrieval insufficiency', 'retrieval appears insufficient
', 'relevant memories are present but not surfaced', 'the facts exist
in the bank but are not retrieved', 'extraction dropped the date', '
```

```
memories are missing from the bank', 'the answer model ignored the
evidence in the context'. Examples that do not count: a stage mentioned
 only as the setting of a symptom ('facts are missed despite 5
retrieved memories', 'low F1 after retrieval'), or a stage mentioned
only in a list of options.

Rules, applied in order:
1. Symptom descriptions never count, whatever stage they hint at: zero-
score counts, abstentions, 'not mentioned' answers, vague, verbose,
incomplete or wrong answers, 'missing key facts'. They describe the
outcome and are compatible with every stage.
2. A count or a category name is not evidence.
3. Statements about intended improvements, recipes, flags to enable, or
 what a change is meant to achieve are not evidence.
4. The evidence field must be a verbatim quote of the phrase that names
 the cause. If you cannot quote such a phrase, the answer is the empty
list.
5. When rules 1-4 leave any doubt, the empty list is the correct answer
. An empty list is never a mistake here; an inferred stage is.

Return JSON only: {"stages": [<zero or more of "S1", "S2", "S3">], "
evidence": "<verbatim quote or empty>", "reason": "<one sentence naming
 the rule applied>"}
```

Stage annotator, ARC diagnosis form

```
Below are the first three lines of the '## Diagnosis' section an
optimisation agent wrote before changing the configuration of the
memory system (its 'change' line is withheld because it describes
actions, not causes).

failure_pattern: {failure_pattern}
suspected_stage: {suspected_stage}
target: {target}

Task: record the stage or stages that the author explicitly blames for
the failures. This is a transcription task, not a diagnosis: do not
decide which stage is actually at fault, and do not complete the author
's reasoning for them. Read all three lines; each line can name a cause
.

What counts as naming a stage as the cause: a phrase that says the
stage's own output was insufficient, irrelevant, shallow, misaligned,
missing or wrong. Examples that count: 'insufficient or irrelevant
retrieval', 'retrieval insufficiency', 'retrieval appears insufficient
', 'relevant memories are present but not surfaced', 'the facts exist
in the bank but are not retrieved', 'extraction dropped the date', '
memories are missing from the bank', 'the answer model ignored the
evidence in the context'. Examples that do not count: a stage mentioned
 only as the setting of a symptom ('facts are missed despite 5
retrieved memories', 'low F1 after retrieval'), or a stage mentioned
only in a list of options.

Rules, applied in order:
1. Symptom descriptions never count, whatever stage they hint at: low
or zero F1, abstentions, 'Not mentioned' answers, vague, incomplete,
wrong or overly long answers, 'facts were missed'. They describe the
outcome and are compatible with every stage.
2. The suspected_stage line alone: if it repeats three or more of the
options, that line carries no information (the other two lines are
still read); if it names exactly one stage, that stage counts; if it
names two stages joined by '|', '/', 'or' and no other line commits to
```

```
one of them, the answer is the set of those two; if it says 'mixed, but
 primarily / dominant X', X alone counts.
3. Statements about intended improvements or what a change is meant to
achieve are not evidence.
4. The evidence field must be a verbatim quote of the phrase that names
 the cause. If you cannot quote such a phrase, the answer is the empty
list.
5. When rules 1-4 leave any doubt, the empty list is the correct answer
. An empty list is never a mistake here; an inferred stage is.

Return JSON only: {"stages": [<zero or more of "S1", "S2", "S3">], "
evidence": "<verbatim quote or empty>", "reason": "<one sentence naming
 the rule applied>"}
```

Stage annotator, configuration-field form

```
Below is one configuration field of the memory system that an
optimisation agent changed, with the documentation line the agent was
shown and the agent's own explanation of its changes.

field: {field}
documentation: {documentation}
proposed_change: {proposed_change}
agent_explanation: {agent_explanation}

Task: record the stage or stages this field acts on directly. This is a
 transcription of the documentation, not a judgement about whether the
change helps; do not guess from the field name alone when the
documentation says nothing.

What counts as acting on a stage: the documentation says the field
changes what that stage produces. S1: how memories are written into the
 bank. S2: which memories are retrieved or how they are ranked (
retrieval views, per-view top-k, fusion mode and weights, query
decomposition or intent planning, coverage checks that trigger more
retrieval, reranking, entity or graph expansion, time decay and its
anchor date). S3: how the answer is generated from the retrieved
context (answer prompt or style, verification passes, the answer model)
. A field that sets how many retrieved memories are handed to the
answer prompt acts on both S2 and S3.

Rules, applied in order:
1. The agent's explanation may confirm the documentation but cannot add
 a stage the documentation does not support.
2. A field the protocol forces back to its default still maps to the
stage it would act on; a field that only groups other fields (a
container of per-category overrides, a recipe name) maps to no stage.
3. The evidence field must be a verbatim quote of the documentation
phrase, or of the explanation only when the documentation line is
missing and the explanation states the mechanism. If you cannot quote
such a phrase, the answer is the empty list.
4. When rules 1-3 leave any doubt, the empty list is the correct answer
. An empty list is never a mistake here; a guessed stage is.

Return JSON only: {"stages": [<zero or more of "S1", "S2", "S3">], "
evidence": "<verbatim quote or empty>", "reason": "<one sentence naming
 the rule applied>"}
```

MemPro and GEPA edit annotation

```
S1 is persistent memory formation: creation or rewriting of stored
abstracts and raw memory pages. S2 is the complete research loop up to
the
```

```
final integrated summary: planning, retrieval, integration, date/
identity
processing, information checks, stopping, and follow-up search. S3 is
the final
answer module after that summary: answer prompt, router, validator, and
 answer
postprocessing. Integration and InfoCheck are S2. Classify the
operation that
the supplied edit actually changes, not the likely reason it was
proposed.

Read the parent-to-child edit below. Determine which stage(s) the edit
directly
changes. Do not use evaluation results, gold evidence, diagnosis notes,
 or
selected/rejected status; they are intentionally absent. A prompt or
function
that spans more than one stage may be MIXED. If the diff does not
establish a
stage, use UNKNOWN. Quote exact short lines from the supplied edit as
evidence.
Return JSON: {"stages":["S1|S2|S3|MIXED|UNKNOWN"],"operation":"
memory_formation|planning|retrieval|integration|info_check|
answer_generation|router_validation|mixed|unknown","evidence":["exact
quote"],"reason":"brief"}.


EDIT (the only evidence available):
{edit}
```

MemPro diagnosis-claim annotation

```
Read one historical diagnosis source. Return only explicitly asserted
observed
faults that identify a stage operation. Do not turn a repair or
intention into a
diagnosis, and do not infer a stage from an outcome, score, or file
name. Keep
tentative causes out. For every claim quote an exact short substring
from the
source. Return JSON with claims, each having stage (S1/S2/S3),
assertion,
statement_type (asserted_fault or other), quote, and reason.
Schema: {"claims":[{"stage":"S1|S2|S3","assertion":"...","
statement_type":"asserted_fault|other","quote":"...","reason":"..."}]}.


SOURCE:
{source}
```

D.4 Feedback of the intervention arms

Every MemPro iteration prompt consists of the released task prompt followed by the harness notes. E2 appends the evidence-ledger notes, and E3 appends one further note. Between iterations the harness rewrites the ledger of the current version in the agent's workspace, whose E2 view and E3 additions are printed after the notes.

Task prompt of the released loop, all arms

```
Read AGENTS.md, registry/versions.json, and the local scripts first.
Continue MemPro framework evolution for locomo: select a base version,
create one coherent candidate direction, use targeted debug before any
full benchmark, and avoid sample-specific shortcuts.
```

Harness notes appended in every arm

```
REPRODUCTION NOTES (outer evolution iteration {iteration} of {total};
these are harness facts, not instructions about how to analyse):
- The evolution split is data/locomo/locomo10_train.json (154 questions
 over 10 conversations). Always pass
  `--data data/locomo/locomo10_train.json` to scripts/run_eval.py (
after the `--` separator). No other dataset
  file exists here; the held-out split is not available.
- This session must produce exactly one new child version (scripts/
register_version.py, edit only files under
  versions/<new_id>/), benchmark it on the full evolution split, and
record its metrics in registry/versions.json
  yourself from runs/<new_id>/train/batch_statistics_0_9.json: metrics.
train_score = overall_llm_as_judge_avg * 100
  (2 decimals), metrics.train_by_type = {category_type:
llm_as_judge_avg * 100}, metrics.train_num_questions = 154,
  status = "evaluated". Record only full 154-question runs, never a
partial or debug run. Do not write any
  metrics before runs/<new_id>/train/batch_statistics_0_9.json exists;
the driver verifies the recorded train_score
  against the per-question results in runs/<new_id>/train/conv-*/
qa_results.json (all 10 conversations, 154
  questions) and will send you back until they match. Writing
batch_statistics or registry metrics by hand is
  detected and only wastes the session.
- A full train-split run takes 30-60 minutes. Launch it detached so it
survives tool timeouts, e.g.
  `setsid nohup python scripts/run_eval.py --version <id> --split train
 -- --data data/locomo/locomo10_train.json
  > runs/<id>/train_eval.log 2>&1 < /dev/null &`, then poll with `sleep
 120; tail -3 runs/<id>/train_eval.log;
  ls runs/<id>/train/` until batch_statistics_0_9.json appears. Never
kill an evaluation process and never start a
  second evaluation into the same runs/<id>/train directory. If you
edit the version after a debug run, pass
  --force-runtime to run_eval.py so the runtime is rebuilt from your
files.
- Paths are relative to the workspace MemPro/locomo (your cwd). The
evaluation output is
  runs/<id>/train/batch_statistics_0_9.json; check it with `ls runs/<id
>/train/` from the workspace root, not from
  scripts/. Before launching any evaluation run `pgrep -af eval/
locomo_test.py`; if one is running for <id>, wait.
  A second launch into the same runs/<id>/train is refused by run_eval.
py (exit code 75).
- Do not create, copy or edit anything under versions/v0000/,
best_versions/, eval/, data/, scripts/, or
  MemPro/_shared/; do not copy files from best_versions/ into a version
 directory. Do not change model names,
  API settings, temperature or worker counts.
- File editing: `apply_patch` needs the exact patch format (`*** Begin
Patch`, `*** Update File: <path>`, `@@`
  hunks with a context line, lines prefixed by space/+/-; no line
numbers, no JSON wrapper). If a patch is
  rejected, rewrite the whole file with `cat > <path> <<'EOF' ... EOF`
and re-read it to verify.
- End the session with one JSON line:
  {"new_version": "<id>", "parent": "<id>", "train_score": <float or
null>, "summary": "<one line>"}
```

Evidence-ledger notes appended in E2 and E3

```
EVIDENCE LEDGER (harness-measured facts about the registered versions;
read evidence/README.md and evidence/INDEX.md first,
then evidence/<version>_ledger.md of the version you select as base):
- Every evaluated version was re-evaluated once more under the same
configuration (same memory), and every judge-WRONG
  question of it was replayed at temperature 0. From these, each of the
 154 questions is classed for that version as
  stable-wrong / ghost / fragile / stable-right (definitions in
evidence/README.md). The ledger also lists which questions a
  child fixed and which it broke relative to its parent counting only
stable transitions, and a table of everything tried
  so far with both evaluation scores.
- Only stable-wrong questions are reproducible failures. Ghost and
fragile questions flip without any change to the
  framework; fixing one of them proves nothing and it flips back on the
 next evaluation. Choose the failure pattern to
  work on from the stable-wrong set, and read the debug you run against
 it.
- A single train score moves by several points between two evaluations
of the same version; compare a child with its
  parent by both evaluations, not by one number.
```

Additional evidence-ledger note in E3

```
- For each stable-wrong question the ledger also gives the failure
stage found by evidence probes (ineligible / S1 memory
  / S2 retrieval / S3 integration-or-answer), the gold evidence page
ids, whether they are in the memory bank, and which
  pages were retrieved; definitions in evidence/README.md. Use the
stage counts to decide which part of the pipeline to
  change.
```

Ledger of one version, E2 view

```
# Evidence ledger for {version} (harness-measured, generated {timestamp
})

Two independent full evaluations of {version} on the 154-question
evolution split, same configuration, same memory: A (the recorded run)
= {score_A}, B (re-run) = {score_B}. Mean = {mean}. Every judge-WRONG
question of run A was replayed once more at temperature 0 with the same
 version.

## Question classes

| class | n | meaning |
| --- | ---: | --- |
| stable-wrong | {n} | wrong in A, wrong in B, wrong on the temperature
-0 replay. The only reproducible failures. |
| ghost | {n} | wrong in A and B but CORRECT on the temperature-0
replay of this same version: the failure is sampling noise, not a
defect. |
| fragile | {n} | A and B disagree. Boundary questions; they flip under
 any change and flip back. A fix of one of these is not evidence. |
| stable-right | {n} | correct in A and B. |

### stable-wrong questions (work on these)

| question | type | recorded answer | gold |
| --- | --- | --- | --- |
| {conv} q{k}: {question} | {type} | {recorded_answer} | {gold} |
```

```
### ghost and fragile questions (do NOT treat as failures; they flip
without any change)

| question | type | class | A | B |
| --- | --- | --- | --- | --- |
| {conv} q{k}: {question} | {type} | {ghost_or_fragile} | {C_or_W} | {
C_or_W} |

## Change from parent {parent} to {version} (stable transitions only)

- fixed (stable-wrong/ghost in {parent} -> stable-right in {version}):
{n} {question_ids}
- broken (stable-right in {parent} -> stable-wrong/ghost in {version}):
 {n} {question_ids}
- flips that involve a fragile question on either side (not counted): {
n}

## Everything tried so far in this workspace

| version | parent | train A | train B | mean | note |
| --- | --- | ---: | ---: | ---: | --- |
| {version_id} | {parent_id} | {train_A} | {train_B} | {mean} | {
registry_note} |

A single train score moves by up to ~6 points between two evaluations
of the same version (2σ of the recorded re-evaluations); a child is
only better than its parent if both of its evaluations are above both
of the parent's.
```

Ledger of one version, stable-wrong section of the E3 view

```
### stable-wrong questions (work on these)

| question | type | stage (probe result) | gold pages | in bank |
retrieved | recorded answer | gold |
| --- | --- | --- | --- | --- | --- | --- | --- |
| {conv} q{k}: {question} | {type} | {stage_text} ({sublabel}) | {
gold_pages} | {gold_pages_in_bank} | {retrieved_pages} | {
recorded_answer} | {gold} |

Stage counts over stable-wrong: {label} {count}, {label} {count}.
```

Ledger of one version, texts of the stage column in the E3 view

```
ineligible: the answer model + judge fail even when the gold evidence
turns are pasted in directly; not fixable through memory/retrieval/
integration
S1 memory: the gold pages are in the bank but their abstracts do not
carry the needed fact; injecting the raw gold pages fixes it
S2 retrieval: the gold pages are in the bank and their abstracts
suffice, but retrieval did not return them; injecting them fixes it
S3 integration/answer: the gold pages were retrieved, the answer is
still wrong
S3 integration/answer: even with the gold pages injected into the
retrieved set the answer stays wrong
probe failed (no label)
```

D.5 REPAIR SESSION

In the repair arms, a revision with at least one stable repair and one stable regression against the incumbent receives one repair session. Its prompt is the head below, followed by the harness and evidence notes of the arm and the closing line. Regular iterations of the repair arms end with the acceptance note.

Repair session prompt, head

```
REPAIR TURN (harness-initiated; outer evolution iteration {iteration}
of {total}, repair session {k}).

Your candidate {candidate} (built on {base}; note: "{note}") was
evaluated twice by the harness under the same configuration and its
wrong answers were replayed at temperature 0. Counting only STABLE
transitions relative to {base} (definitions in evidence/README.md):
it FIXED {fixed_n} question(s) that {base} got reproducibly wrong, and
it BROKE {broken_n} question(s) that {base} answered
correctly in both of its evaluations. Two scores of {candidate}: {
cand_A} / {cand_B}; of {base}: {base_A} / {base_B}.

BROKEN by {candidate} (correct in {base} in both evaluations, wrong in
{candidate} in both evaluations and on the replay):
| question | type | gold | {base} answer (judged CORRECT) | {candidate}
 answer (judged WRONG) |
| --- | --- | --- | --- | --- |
{broken_rows}

FIXED by {candidate} (keep these):
| question | type | gold | {candidate} answer (judged CORRECT) |
| --- | --- | --- | --- |
{fixed_rows}

Files that differ between versions/{base}/ and versions/{candidate}/: {
changed_files}.

YOUR TASK IN THIS SESSION: register exactly one child of {candidate} (`
python scripts/register_version.py --base-version {candidate} --new-
version <next id>
--note "repair of {candidate}: ..."`) that KEEPS the fixes and REMOVES
the regressions. Do not start a new direction. Procedure:
1. Read the diff (`diff -u versions/{base}/<file> versions/{candidate
}/<file>` for each changed file) and the traces of the broken questions
   under runs/{candidate}/train/<conv>/ (research_trace_q<k>.json,
qa_results.json) next to the same questions under runs/{base}/train/<
conv>/,
   to find which part of the change turned each broken question wrong (
typically: a new instruction that over-applies, a changed
   retrieval plan that drops the evidence, a reformatted answer the
judge no longer accepts).
2. Narrow that part: make it conditional on the situation it was meant
for, restrict it to the question types it helped, or revert
   it and keep the rest. Keep everything else byte-identical to {
candidate}.
3. Debug on the broken and the fixed questions before the full run: `
python scripts/run_eval.py --version <id> --split train -- --data
   data/locomo/locomo10_train.json --conv-id <conv> --question-index <k
>` runs one question (indices as in the tables above; memory for
   that conversation is built on first use). Check that the broken ones
 are correct again and the fixed ones stay correct.
4. Then run the FULL train evaluation, record the metrics and finish
with the JSON line, exactly as in a normal iteration.

HARD RULES: no special-casing by sample id, question index, question
wording, gold answer or answer lexicon (the harness scans the
diff of the new version and rejects such versions); no edits outside
versions/<new id>/; do not touch the memory construction unless
the regressions come from it (then memory must be rebuilt: do not reuse
 {candidate}'s memory).
```

Repair session prompt, closing line

```
HARNESS ACCEPTANCE (repair turn): the base version of the child you
register MUST be {candidate} (`--base-version {candidate}`); the driver
 sends you back otherwise.
```

Acceptance note appended to regular iterations of the repair arms

```
HARNESS ACCEPTANCE: the harness keeps one incumbent version and decides
 acceptance from paired stable transitions (evidence/INCUMBENT.md). The
 current incumbent is {incumbent}. The base version of the child you
register in this session MUST be {incumbent} (`--base-version {
incumbent}`); versions not on the incumbent chain were rejected and
must not be used as base (rejected so far: {rejected_list}). The driver
 sends you back if the registered child has another parent.
```

E Additional Experiments

This section reports six experiments that test the audit instrument and extend the findings of the main text to another system, other models, and a larger split. Each subsection states its setup, the quantities it measures, its results, and how to read them.

E.1 Effect of provenance recording

Separating S1 from S2 requires evidence provenance, the dialogue turns from which each memory was extracted. The EvolveMem harness, on which ARC also runs, records it during extraction. Every turn in the extraction window carries an identifier, and a 726-character contract appended to the extraction prompt asks the extractor to cite the supporting identifiers of each memory, which the harness checks against the window. The identifiers are kept as metadata, outside the indexed text and the answer prompt (2 of EvolveMem's 5,240 round-0 memories copy one into their text), but the longer prompt could still change what is extracted. MemPro's loop instead runs without provenance, which is added only when frozen versions are replayed, so recording cannot affect that loop.

An audit instrument should not produce the changes it measures, so we apply the test of Section 3 to provenance recording, namely whether it changes outcomes more than an unchanged system does. With Qwen3-max for extraction and answering and the weak initial configuration, we answer the first 20 LoCoMo questions of nine conversations from memory extracted without and with provenance, under an identical QA configuration. The two memories come from separate extractions, so Table 13 sets their difference against two references on the same questions and configuration, namely the same memory answered twice and memory extracted twice by the same provenance-recording code.

Table 13: Effect of provenance recording on answers. EvolveMem static QA on the same 180 LoCoMo questions, token F1$\times$100; each row pairs two evaluations.

Pair	ΔF1 (pp)	95% interval (pp)	F1 changed (up/down)	Answer text changed
Same memory, answered twice	$+0.43$	$[-1.5, +2.0]$	12 (7/5)	30
Memory extracted twice, same code	-0.94	$[-5.5, +2.9]$	51 (30/21)	88
Without $\rightarrow$ with provenance	$+0.48$	$[-3.9, +4.4]$	69 (30/39)	103

The first two rows record provenance in both evaluations: EvolveMem's round 0 evaluated again, and two builds by the same code. Intervals: paired bootstrap over the nine conversations, 20,000 draws. Questions: 71 multi-hop, 76 temporal, 29 open-domain, 4 single-hop. One conversation is excluded because its provenance build is incomplete.

Recording provenance moves mean F1 by $+0.48$ pp, less than the -0.94 pp between two extractions with unchanged code, and all three intervals include zero. It changes the F1 of 69 questions, against 51 for re-extraction and 12 for re-answering. Across all 1,388 category 1–4 questions of the nine conversations, re-extraction alone changes 35% of question scores and flips 17% of zero/nonzero outcomes, against 3.5% when the same memory is answered twice. Answer-level turnover thus comes

mostly from rebuilding memory, which every extraction run incurs, rather than from the contract. Score levels agree, as four provenance-recording builds score 35.5–37.7 on these questions and the build without provenance 37.2. We do not read the $+0.48$ pp as a gain, since excluding one baseline answer truncated at the output limit gives -0.07 pp. The provenance build holds 4,866 memories against 4,850. Each carries non-empty, unique identifiers from its own session, and no identifier appears in searchable memory fields or in any of the 180 answer prompts.

The cost falls on extraction. On the nine conversations, extraction uses 34.5% more input and 20.0% more output tokens (22.4% higher estimated cost), mainly for the per-turn identifiers and the repeated contract, whereas answer-generation input tokens fall by 3.7%.

The check has limits. Each arm is one run over nine conversations, so the interval rules out shifts larger than about 4 pp but does not establish equivalence at the scale of a single evolution step (Table 1). It also concerns answers, not whether the contract changes which facts are extracted. Every EvolveMem and ARC version, including rebuilt ones, is extracted with the same recording, so the parent–child comparisons in the main text compare versions recorded the same way.

E.2 The ARC loop on the released Omni-SimpleMem system

The ARC configurations of Appendix B.2 evolve EvolveMem's initial pipeline, because Omni-SimpleMem (Liu et al., 2026b), the memory system that ARC reports discovering, is released only in its final form. This leaves open whether the loop behaves differently on the system it produced. We therefore run the same ARC harness on our reproduction of the released Omni-SimpleMem, in the five LoCoMo configurations closed/r0, closed/inc, open/r0, open/inc, and open/inc with the GPT-4o agent of Appendix E.3. On Mem-Gallery, Omni-SimpleMem stores each dialogue round verbatim without an extraction template, so only closed/r0, closed/inc, and closed/inc with the GPT-4o agent apply. Extraction carries a provenance contract of the same form as EvolveMem's (Appendix D.1), and the retrieval and answer path is reimplemented step by step from the released code so that its configuration fields take effect, since the released runner overrides the retrieval depth internally. Each configuration starts from two versions. The *published* start is the released configuration. The *weak* start switches off the system's own reported discoveries, whose intermediate versions are not released, namely the rule-based answer prompt with structured output, hybrid retrieval, deeper retrieval, and context expansion on LoCoMo, and the dialogue text in context, hybrid retrieval, and the image catalogue on Mem-Gallery. The weak start is thus our construction, comparable to EvolveMem's weak initial configuration, rather than the paper's original baseline. Everything else follows the ARC runs of the main text, including the ten conversations, the twenty scenarios, the stopping rule after two rounds without improvement, and the audit chain. Table 14 reports the in-sample gain of each configuration's best version over round 0, averaged over conversations or scenarios.

From the published start, the loop finds a converged system. Its best versions gain $+1.4$ to $+2.0$ pts on LoCoMo and $+1.0$ to $+1.7$ F1 points on Mem-Gallery, and the runs stop after two to ten rounds. These gains are of the size of the measurement, since reevaluating the published round 0 on the same memory flips 3.6% of its 1,540 LoCoMo outcomes (1.9% up and 1.7% down), and 43 of the 59 accepted steps in the four Qwen-agent LoCoMo configurations lie within the 2σ band of the reevaluations recorded by the loop, as do 19 of the 22 in closed/r0. On Mem-Gallery, split-half replay keeps $+0.2$ to $+1.0$ of scenario-mean gains of $+1.0$ to $+1.7$ F1 points. The loop that produced this system, run again on it, thus mostly accepts steps that a reevaluation of the parent could also produce, the pattern of Table 1.

From the weak start, the loop gains $+32.7$ to $+36.9$ pts on LoCoMo, of which $+29.5$ to $+35.6$ pts are in place after round 2. In nine of the ten closed/r0 conversations the first accepted edit replaces the free-form answer call with the released rule-based prompt, and in six of them it also enables hybrid retrieval, so the agent regains the released system's first discoveries in one step. The later rounds accept adjustments of retrieval and context settings, and 51 of the 134 accepted steps across the five weak-start configurations lie within 2σ. On Mem-Gallery, the weak start gains $+23.2$ to $+24.6$ scenario-mean F1 points, of which split-half replay keeps $+22.2$ to $+23.9$. This one step is a general repair rather than a fit to the development conversation. Applying the configuration that closed/r0 selects on one conversation after its first round to the banks of the other nine conversations raises their mixed score from 24.1 to 58.5, a gain of $+34.4$ pts that is positive in all nine (from $+28.6$ to $+36.8$), and applying the configuration selected on one Mem-Gallery scenario to the other nineteen

Table 14: The ARC loop on the released Omni-SimpleMem from the published and from the weak start. LoCoMo: mixed score×100, mean over ten conversations, with category 1–4 F1 gains in parentheses. Mem-Gallery: official F1×100 over all 1,711 questions, with the scenario-mean gain and its split-half replay. Round 2: gain after round 2. Accepted: accepted steps, and among them those within the 2σ band of the same-configuration reevaluations recorded by the loop.

Start	Configuration	Round 0	Gain	Round 2	Rounds	Accepted (in 2σ)
LoCoMo						
Published	closed/r0	59.1	+1.8 (+1.2)	+1.1	2–9	22 (19)
	closed/inc	59.4	+1.4 (+0.9)	+1.1	2–7	11 (7)
	open/r0	59.2	+1.7 (+0.4)	+1.0	2–10	15 (10)
	open/inc	59.3	+1.7 (+0.6)	+1.6	2–6	11 (7)
	open/inc, GPT-4o	59.0	+2.0 (+2.3)	+1.2	2–8	16 (—)
Weak	closed/r0	23.8	+36.9 (+21.9)	+35.1	3–10	31 (16)
	closed/inc	23.7	+36.6 (+21.2)	+35.6	3–8	21 (4)
	open/r0	23.9	+33.3 (+16.3)	+32.6	3–6	21 (6)
	open/inc	23.9	+32.7 (+16.4)	+29.5	3–10	32 (14)
	open/inc, GPT-4o	23.7	+36.4 (+21.9)	+30.3	3–10	29 (11)
		Round 0 → best	Scen. mean	Split-half	Rounds	Accepted
Mem-Gallery						
Published	closed/r0	69.0 → 70.4	+1.4	+0.7	67	26
	closed/inc	69.0 → 70.5	+1.7	+1.0	74	29
	closed/inc, GPT-4o	68.8 → 69.8	+1.0	+0.2	76	31
Weak	closed/r0	41.7 → 68.1	+24.6	+23.9	96	47
	closed/inc	41.8 → 68.1	+24.5	+23.6	93	46
	closed/inc, GPT-4o	41.7 → 66.9	+23.2	+22.2	95	48

LoCoMo rounds: rounds run per conversation before the stopping rule. Mem-Gallery rounds and accepted steps are summed over the twenty scenarios, and all runs stop after 2–6 rounds. Split-half: a retrospective replay of the accepted steps on the other half of each scenario's questions. The GPT-4o agent run on LoCoMo records no same-configuration reevaluation, so its steps are not classified.

raises their F1 from 42.5 to 64.8, positive in all nineteen (from +10.3 to +44.7). This mirrors the retrieval upgrade that carries the EvolveMem and ARC gains in Section 4.

The diagnoses follow the pattern of Table 20. The stated stage matches the bottleneck stage of the version read in 0.57 of the scorable published closed/r0 rounds and in 0.82 of the weak closed/r0 rounds, yet in neither case exceeds a constant reading of retrieval (0.63 and 1.00), and it falls to 0.00–0.34 once the reference is the incumbent or the action space is open. In the weak-start open configurations the agent names the answer stage in 39–48 rounds and rewrites the answer prompt. Repeated labels agree at 0.90–0.94 on the weak-start LoCoMo runs and at 0.96–0.98 on Mem-Gallery.

E.3 Model substitutions

The audits of the main text use Qwen models in every role of the loop, so the low stage agreement of Table 20 could reflect a weak diagnoser rather than a property of the loop, and a stronger diagnoser could raise agreement and, with it, the gain. To test whether the readings of Sections 3–5 depend on the model family, we replace the EvolveMem diagnoser with GPT-4o or GPT-4o-mini, the EvolveMem answer model with GPT-4o-mini under a GPT-4o diagnoser, and the ARC agent with GPT-4o, on both datasets. Each run keeps the root memory, the loop, the acceptance rule, and the Qwen3-max probe labels of its Qwen counterpart, and the answer-model runs re-answer round 0 on the same memory. We then repeat the three measurements of the audit on these runs, namely the same-input flips and the accepted steps within 2σ of Section 3, the alignment ledger of Table 2, and the early and late repair and regression volumes of Table 3. Table 15 first reports, per run, how often the stage of a proposal (EvolveMem) or of a stated diagnosis (ARC) matches the bottleneck stage of the version read, together with the in-sample gain of the best version over round 0.

Agreement and gain. We find that agreement with the audited stage does not order the gains, whereas the round in which a loop reaches the retrieval upgrade of Section 4 does. On LoCoMo, GPT-4o agrees more often than Qwen3-max (0.30 against 0.16) and gains +11.9 rather than +7.9

Table 15: Stage agreement and in-sample gain under model substitution. Agreement compares the stage of the proposal (EvolveMem) or of the stated diagnosis (ARC) with the bottleneck stage of the version read, on rounds with at least five attributable failures. Gain is the in-sample gain of the best version over round 0 in pts of the loop's score, the mixed score on LoCoMo and F1 on Mem-Gallery, averaged over conversations or scenarios. The LoCoMo ARC runs rebuild memory when the extraction template changes, unlike the open/inc run of Table 20. The Mem-Gallery GPT-4o ARC row covers the 17 of 20 scenarios whose runs pass verification, and the Mem-Gallery GPT-4o-mini answer-model run has no probe labels.

System	Diagnoser or agent	Answer model	Agreement	Gain (pts)
LoCoMo				
EvolveMem	Qwen3-max	Qwen3-max	0.16 (11/67)	+7.9
	GPT-4o	Qwen3-max	0.30 (19/63)	+11.9
	GPT-4o-mini	Qwen3-max	0.69 (44/64)	+2.9
	GPT-4o	GPT-4o-mini	0.58 (40/69)	+7.9
ARC open/inc	Qwen3-max	Qwen3-max	0.50 (20/40)	+5.5
	GPT-4o	Qwen3-max	0.68 (41/60)	+11.0
Mem-Gallery				
EvolveMem	Qwen3-max	Qwen3-max	0.82 (59/72)	+11.0
	GPT-4o	Qwen3-max	0.86 (69/80)	+8.0
	GPT-4o	GPT-4o-mini	–	+5.5
ARC open/inc	Qwen3-max	Qwen3-max	0.22 (8/37)	+5.7
	GPT-4o	Qwen3-max	1.00 (23/23)	+13.8

pts, but GPT-4o-mini agrees most often (0.69) and gains only $+2.9$ pts. GPT-4o proposes semantic retrieval in round 0 of all ten conversations, as the stage oracle A3 does, and ends close to A3's $+12.3$ pts (Table 22). GPT-4o-mini instead changes a retrieval fusion setting without enabling semantic retrieval, so only three of its conversations reach the upgrade, in rounds 4 to 7, against seven for Qwen3-max. With GPT-4o-mini as the answer model, the answer stage becomes the bottleneck in 32 of 69 rounds, against none of 67 with Qwen3-max, and GPT-4o's proposals match it in 40 of the 69, yet the loop gains the same $+7.9$ pts as the Qwen loop. The GPT-4o ARC agent states retrieval in 62 of its 66 diagnoses and agrees with the bottleneck more often than the Qwen agent (0.68 against 0.50). Since its edits stay within the retrieval settings, all 67 claims in these diagnoses are executed, whereas the guard stops the edits behind 26 of the Qwen agent's 43 diagnoses, and the GPT-4o agent doubles the in-sample gain ($+11.0$ against $+5.5$ pts). On Mem-Gallery, nearly every attributable round has its bottleneck at retrieval. As the EvolveMem diagnoser, GPT-4o agrees with it slightly more often than Qwen3-max (0.86 against 0.82) but gains less ($+8.0$ against $+11.0$ pts), and as the ARC agent it states retrieval in all 23 attributable rounds and gains $+13.8$ pts against $+6.6$ for the Qwen agent on the same 17 scenarios. The substituted model thus changes which stage the loop keeps editing and how early it finds the one upgrade that pays, which is the result of Table 22 seen from another side.

Alignment ledger. Table 16 applies the ledger of Table 2 to the four LoCoMo runs, and the per-edit picture is the same under every substitution. Edits on the bottleneck stage repair 26–31% of their parents' failures against 13% for the others, with the GPT-4o-mini diagnoser lower on both sides (11% against 9%), and 48 of the 52 gains beyond noise in the table are aligned. Regressions again barely depend on alignment, at 4.5% against 5.7% of previously correct questions with the GPT-4o diagnoser, 5.0% against 5.6% with the GPT-4o-mini diagnoser, and 7.7% against 8.4% with the GPT-4o-mini answer model, whereas the GPT-4o ARC agent regresses more on its aligned edits (7.3% against 4.4%), which are its larger retrieval changes. The gains also sit where Section 4 places them, on parents that have not yet enabled semantic retrieval, for 10 of the 11 gains with the GPT-4o diagnoser, 10 of 11 with the GPT-4o-mini answer model, and all 9 with the GPT-4o-mini diagnoser. The GPT-4o ARC agent is the one run that keeps gaining after the upgrade, with 12 gains before it and 9 after, 7 of them aligned, since it goes on enlarging retrieval settings where the Qwen agent turns to the extraction and answer prompts that the guard then stops.

Feedback variability. The flips of Section 3 carry over as well. Pooling each loop's own same-configuration reevaluations with a reevaluation of its round 0, the three GPT EvolveMem runs flip 3.1–3.6% of category 1–4 outcomes, against 2.8% for the Qwen loop on the same kind of pairs, and

Table 16: Alignment ledger under model substitution. The ledger of Table 2 on the LoCoMo runs of Table 15, with the Qwen runs repeated for reference. The noise band of the GPT-4o ARC agent, whose run records a single same-configuration reevaluation, is that of the Qwen run with the same harness and evaluation models.

			Rate (%)		Class against own 2σ band				Score change		
Run	Aligned	n	Repair	Regr.	Gain	In band	Offset	None	Up	Noise	Down
EvolveMem, Qwen3-max	yes	14	33.7	5.7	8	0	3	3 (1)	7	3	4
	no	33	8.4	4.7	0	2	9	22 (6)	0	22	11
GPT-4o diagnoser	yes	27	27.3	4.5	10	1	3	13 (4)	11	7	9
	no	20	12.9	5.7	1	1	5	13 (3)	1	11	8
GPT-4o-mini diagn.	yes	44	11.3	5.0	9	9	3	23 (10)	8	24	12
	no	16	8.9	5.6	0	4	3	9 (7)	0	11	5
GPT-4o-mini answers	yes	20	30.5	7.7	10	2	6	2 (1)	11	4	5
	no	32	12.7	8.4	1	1	11	19 (7)	4	11	17
ARC, Qwen3-max agent	yes	10	40.6	7.1	7	3	0	0 (0)	7	2	1
	no	4	19.7	16.2	0	0	2	2 (2)	0	1	3
ARC, GPT-4o agent	yes	41	25.8	7.3	19	7	7	8 (1)	17	19	5
	no	19	12.6	4.4	2	2	8	7 (3)	3	8	8

Columns as in Table 2. Edits without a change or with fewer than five attributable parent failures are excluded.

10 of 20, 15 of 25, and 8 of 15 of their accepted steps change the score by less than 2σ of this variation. The GPT-4o ARC agent records no such pair, and under the 2σ of its Qwen counterpart 20 of its 38 accepted steps fall within it, against 9 of 16 for that counterpart. On a same-input reevaluation of round 0, GPT-4o-mini answers flip 48 of 1,540 category 1–4 outcomes (3.1%), against 53 (3.4%) for Qwen3-max. The label checks of Appendix C.2 further cover a GPT probe solver.

Table 17: Repair and regression volumes under model substitution. The measurement of Table 3 on the runs of Table 15, all scored by F1$>$0, with the Qwen runs repeated for reference. E / L: early / late revisions. Unchanged: the same configuration and memory evaluated twice. [§]Fewer than five such pairs. The Mem-Gallery ARC rows use the 17 scenarios of the GPT-4o run.

	Repair		Regression		Unchanged	
Run	E	L	E	L	Rep.	Reg.
LoCoMo						
EvolveMem, Qwen3-max	5.0	2.9	2.4	3.3	1.0	1.0
GPT-4o diagnoser	8.8	1.8	3.4	3.2	1.5	1.6
GPT-4o-mini diagnoser	4.2	3.2	3.0	3.5	1.4	1.2
GPT-4o-mini answer model	8.8	2.6	5.5	4.2	1.5	1.7
ARC open/inc, Qwen3-max agent	7.6	2.2	3.7	3.0	1.2	1.9
GPT-4o agent	9.6	2.9	3.4	5.3	1.2[§]	2.5[§]
Mem-Gallery						
EvolveMem, Qwen3-max	3.5	2.2	1.3	2.7	1.0	0.7
GPT-4o diagnoser	2.3	2.8	1.5	2.1	1.7	1.1
GPT-4o-mini answer model	1.8	2.6	1.6	2.7	0.5	1.3
ARC open/inc, Qwen3-max agent	5.2	1.9	1.4	2.7	0.6	0.4
GPT-4o agent	7.7	2.4	2.5	3.1	–	–

Retention. Table 17 repeats the measurement of Table 3. On LoCoMo, repair volume falls from early to late revisions in every substituted run, from 8.8 to 1.8 pp with the GPT-4o diagnoser and from 9.6 to 2.9 pp with the GPT-4o ARC agent, while regression volume stays or rises, so late revisions regress at least as many questions as they repair in all four runs, and late repairs approach the 1.2–1.7 pp that the unchanged systems flip. On Mem-Gallery the GPT-4o ARC agent shows the same profile, whereas the two GPT EvolveMem runs have no early peak to lose. Their repairs stay between 1.8 and 2.8 pp throughout, close to the unchanged-system flips, which matches their smaller gains, and their early-to-late differences of 0.5 and 0.8 pp lie within the spread across revisions (permutation

$p = 0.34$ and 0.06, against 0.04 for the 1.3-pp fall of the Qwen run). Their regressions still rise, from 1.5 to 2.1 pp and from 1.6 to 2.7 pp, and only in the GPT-4o diagnoser run do late repairs stay above regressions. Repairs that a step newly makes fail again at the next update in 5–19% of cases across the seven runs, against 7–12% for the Qwen runs of Section 5.

Across these substitutions, the model thus changes which stage the loop keeps editing, how early it reaches the retrieval upgrade, and with it the in-sample gain, whereas the three readings of the audit hold on each run. The feedback flips on the scale of single-step gains, repairs concentrate on aligned edits while regressions do not, and repairs decay toward the flips of an unchanged system as the pool drains. The conclusions of Sections 3–5 therefore do not rest on the Qwen family.

E.4 Diagnosis audit funnel

The stage checks of Section 2 score a diagnosis, its proposal, and its executed edit separately. This experiment orders them into a sequence and records, for every round, the first check that fails, so that the rounds surviving each check can be counted. We run it on the two closed-action ARC configurations of Appendix B, closed/r0, which is the released loop and reads the summary of the initial version in every round, and closed/inc, which additionally receives the summary of the incumbent that the loop currently keeps. Both use the same ten LoCoMo conversations, the same editable action space, and the stated-stage judgment (A), so the reference version is the only difference between them.

Checks. Rounds whose version read has fewer than five attributable failures are unscorable and excluded before the first check. L1 compares the stated stage with the bottleneck stage of the version the diagnoser read, R_{seen}. L2 asks whether that judgment still holds for the parent being edited, R_{applied}, and can fail only when the two versions differ. L3 requires the proposal to be executed and non-empty, so blocked and no-op edits fail here. L4 requires the executed edit to cover the parent's bottleneck stage. L5 requires a raw-score change above the 0.011 noise floor.

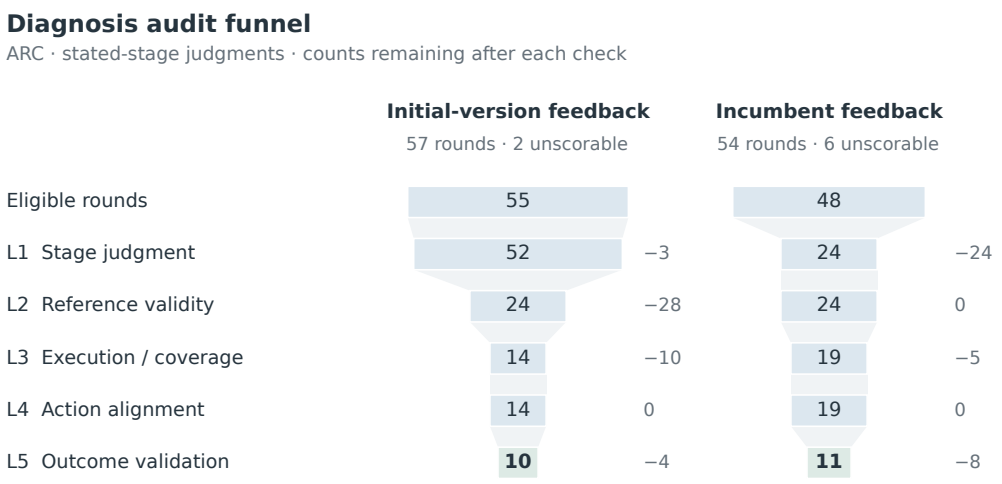

Figure 3: Diagnosis audit funnels of ARC closed/r0 and closed/inc under the stated-stage judgment (A). Widths and internal counts show rounds remaining after each check on a shared count scale, and numbers at right show rounds first failing that check. Unscorable rounds are excluded before L1.

Results. Figure 3 shows the two funnels. In closed/r0, 55 of 57 rounds are scorable and 52 pass L1, because the agent states retrieval in 54 rounds and retrieval is the bottleneck of the initial version in all ten conversations. The loop nevertheless edits an evolved incumbent in 47 rounds, and in 28 of them the parent's bottleneck has moved away from retrieval, so these rounds first fail at L2. Of the remaining 24 rounds, 10 are blocked or empty edits (L3), none miss the parent's bottleneck stage (L4), and 4 change the score within noise (L5), which leaves 10 rounds with a gain. In closed/inc,

R_{seen} equals R_{applied} in all 54 rounds, so L2 cannot fail. Instead, 24 of the 48 scorable rounds first fail at L1. The agent states retrieval in 44 rounds and answer generation in 10, and never states extraction, whereas the bottleneck of the version it reads is extraction in 12 and answer generation in 10 of the 24 failing rounds. Five further rounds fail at L3, none at L4, and 8 at L5, which leaves 11 rounds with a gain. We find that updating the reference moves the failures from the stale-reference check to the stage check while barely raising the number of rounds that reach a gain. Supplying the incumbent's summary therefore removes one failure mode, but the summary does not carry the provenance that would let the agent recognize an extraction or answer-stage bottleneck.

Reading the counts. These are ordered counts of first failures, not a decomposition of causal effects, since an earlier failure can mask a later one. Applying the same sequence to EvolveMem under the proposal-stage judgment (B) places 56 of 67 rounds at L1 and none at L2, although R_{seen} differs from R_{applied} in 39 rounds. Those 56 rounds fail on the stage of the proposal, and the diagnosis text of the same loop is scored separately in Table 20.

E.5 Gold evidence on observed MemPro failures and judge sensitivity

This subsection applies the gold-evidence probe of Algorithm 1 to the failures that MemPro's loop actually observes (Table 18) and tests how far the held-out gains of its selected versions depend on the judge (Table 19).

Table 18: Probes on the failures actually observed by MemPro. Replay uses temperature 0 rather than the original 0.3, and retains each version's integration and answer prompts. Gold evidence is supplied only after replay fails, and a gold-correct failure is assigned to a stage. The "All versions" rows count failed question–version records rather than questions. Mem-Gallery failures without evidence are adversarial questions.

Set	Failures	Replay correct	Gold correct	Gold wrong	No evidence
LoCoMo					
Root, seed 2027	35	3	18	14	0
Root, seed 2028	42	21	7	14	0
Root, seed 2029	40	12	15	12	1
Root total	117	36	40	40	1
All 48 versions (records)	1,234	368	475	381	10
Mem-Gallery					
Root, seed 2026	27	6	9	8	4
Root, seed 2027	43	11	10	15	7
Root, seed 2028	41	4	15	18	4
Root, seed 2029	25	7	1	11	6
Root total	136	28	35	52	21
All 64 versions (records)	1,453	318	612	338	185

For the actual-failure probe in Table 18, the total root proportions are 30.8% replay-correct, 34.2% gold-recovered, 34.2% gold-unresolved, and 0.9% without evidence (rounding explains the sum). Of the 40 gold-unresolved root failures, 27 are answered correctly from the same evidence by at least one of four answer models (Qwen3-max, Qwen3-30B-A3B, GPT-4o, GPT-4o-mini), each answering once from the annotated gold turns under a neutral prompt, and 13 stay wrong under all four, six of them open-domain. On Mem-Gallery, the root failures are 20.6% replay-correct, 25.7% gold-recovered, 38.2% gold-unresolved, and 15.4% adversarial questions without evidence. Of the 52 gold-unresolved ones, 36 are answered correctly by at least one of the four answer models, and 16 stay wrong under all four.

Fixed-answer rejudging changes the measurement without changing the system's answer. For 11,405 unique zero-F1 query–prediction pairs from the loops, Qwen and GPT judges also accept portions of the nominally failed answers, with 95.5% agreement between judges. These protocol effects motivate reporting both the source of the feedback and the metric used to select versions.

Table 19: Judge sensitivity on fixed MemPro held-out answers. Each column compares the selected version to the same reevaluated root answers using that column's judge.

Seed	Version	Qwen gain (pp)	GPT gain (pp)
2027	v0011	$+5.55$	$+1.87$
2028	v0010	$+5.27$	$+4.04$
2029	v0011	$+2.74$	-0.15

E.6 A LARGER DEVELOPMENT SPLIT

The intervention arms of Section 6 evolve on 154 development questions. Doubling the development set resolves only the largest steps. We draw a second split of 308 development questions, stratified by conversation, category, and gold-turn probe outcome, and keep the other 1,232 questions as held-out. On this split, one run of the original loop (E0) produces 16 versions, and we evaluate each of them three times on the development questions. Averaging three evaluations narrows the 2σ band of the unchanged root to 2.54 pp, yet only 2 of the 15 revisions exceed it against their parents, at $+2.71$ and $+2.60$ pp, and both add date anchoring for temporal questions. Along the lineage of the last version, held-out accuracy rises from 77.92% to 83.85%, but only these two steps raise temporal questions significantly, by 15.6 and 9.7 pp, while the other questions change within noise at every step. Across all 15 revisions, 18 questions turn from wrong in all three evaluations of the parent to correct in all three of the child, against 6.0 expected for the unchanged root, and 15 of them are temporal. Moreover, the five that the first revision repairs all stay wrong when the gold evidence is supplied, so its gain does not come from failures of the bottleneck stage. Each revision thus changes too few questions for stage alignment to be tested, and the other 13 revisions, which move the score by at most 1.6 pp, stay inside the band, consistent with the bound above.

F FULL TABLES BY METHOD

This section gives, for each method separately, the quantities behind Tables 2, 3, and 4, together with the held-out gains of every split.

F.1 STAGE AGREEMENT AND EDIT ALIGNMENT

Table 20 reports the stage agreement of claims and proposals on all eligible records, and Table 21 breaks the ledger of Table 2 down by revision class.

Table 20: Stage agreement on eligible records. Statement accuracy (A) and proposal alignment (B) have different eligible populations.

System	Record	n	Agreement
EvolveMem	Proposal (B)	67	0.16
EvolveMem	Statement (A)	34	0.76
ARC closed/inc	Statement (A)	48	0.50
ARC open/inc	Statement (A)	49	0.22

ARC open/inc here is the version that appends re-extracted memories to the existing bank; the rebuild version used elsewhere is scored in Table 15.

Statement–proposal disagreement does not by itself prove a misdirected proposal, since execution constraints, available operators, and the stated reasons must also be checked. EvolveMem additionally has $R_{\text{seen}} \neq R_{\text{applied}}$ in 39/67 rounds. The original ARC loop reads the initial version in all 57 rounds and edits an evolved incumbent in 47 of them, and in 28 the stale reference is where the diagnosis first fails (Appendix E.4). These mismatches are recorded directly, rather than inferred from a final score.

Table 22 gives the round-by-round gains of the three proposers compared in Section 4.

In Table 22, A0 may propose two changes per round, whereas A2/A3 take one menu action. A3 recomputes the incumbent's attributable stages and reaches the useful retrieval upgrade in round 1.

Table 21: Full breakdown of Table 2. Each revision is classified against its own 2σ reevaluation band: gain (repairs and net change beyond the band), in band (repairs beyond it, positive net change within it), offset (repairs beyond it, regressions $\geq$ repairs), none (repairs within it). Positive/neutral/negative counts use the ± 0.011 noise floor on the raw score and are separate from the binary statistics.

System	Aligned?	n	Repair (%)	Regression (%)	gain / in band / offset / none	$+ \,/\, 0 \,/ -$
EvolveMem	Yes	14	33.7	5.7	8 / 0 / 3 / 3	7/3/4
	No	33	8.4	4.7	0 / 2 / 9 / 22	0/22/11
ARC closed/r0	Yes	14	40.2	3.6	11 / 0 / 0 / 3	10/4/0
	No	9	13.0	5.0	0 / 0 / 3 / 6	1/5/3
ARC open/inc	Yes	10	40.6	7.1	7 / 3 / 0 / 0	7/2/1
	No	4	19.7	16.2	0 / 0 / 2 / 2	0/1/3

Table 22: Gain of the incumbent under three proposers that share EvolveMem's root, guard, acceptance rule, and 12-recipe menu. Entries are the in-sample gain (pts) of the incumbent's mixed score over round 0 after each round, averaged over ten conversations.

		Gain after round (pts)						
Proposer	Choice	1	2	3	4	5	6	7
A0	Upstream LLM	$+4.1$	$+5.9$	$+6.1$	$+7.0$	$+7.8$	$+7.8$	$+7.9$
A2	Three random orders	$+2.2$	$+2.7$	$+5.6$	$+6.5$	$+7.3$	$+8.1$	$+8.3$
A3	Bottleneck-stage item	$+10.5$	$+10.8$	$+11.5$	$+11.7$	$+12.0$	$+12.2$	$+12.3$

A0 follows the upstream LLM diagnosis; A2 walks the menu in a random order, pooled over three orders (30 runs); A3 takes the next untried item for the incumbent's bottleneck stage. The three A2 orders end at $+11.6$, $+4.4$, and $+9.0$ pts. Early-stopped runs keep their last incumbent.

The comparison assesses the proposal policy as instantiated, including its action menu and ordering. It is in-sample, and the three random orderings do not support a general equivalence claim for A0 and random search.

Operator scope. Across the audited EvolveMem/ARC edits, 59–84% of the parent's correct questions receive a different retrieved context, and 427 of the 499 regressions occur on them (Figure 2b), and among EvolveMem's misaligned edits that lower the score, the share is 77 of 93. MemPro gates some edits more narrowly. Restricting an edit to the diagnosis's named category does not isolate its repairs. An offline composition control for revisions that do not rebuild memory takes child answers inside the named category and parent answers outside it. It retains 31%/26%/27% of repairs for EvolveMem/ARC closed/ARC open, comparable to 34%/25%/27% for equally sized random category gates, while net gains decline. This is an answer-composition analysis rather than an executed runtime gate, so an edit's affected query set has to be measured rather than inferred from its diagnosis.

MemPro and GEPA stage analysis. Under the sequential protocol, MemPro has 29 aligned, 11 unaligned, and five unscored actions among 45 revisions, and GEPA has 25 aligned and seven unaligned actions among 32 fully evaluated revisions. The bottleneck stage is S2. Forty-two additional GEPA revisions rejected by the small-batch filter do not enter its full-query table. Since these two protocols have different filtering and observability, they are not pooled into Table 2. A split with 308 development questions does not settle MemPro's alignment either, as each of its revisions changes too few questions (Appendix F.4).

F.2 Repair, regression, and retention

Tables 23–25 report, with counts and under both scores, the quantities of Section 5. A separate MemPro run on the seed-2028 split selects by the official LoCoMo category 1–4 F1 for fifteen rounds, with no judge in its feedback or selection. On the 1,386 held-out questions, its round-15 version gains $+5.70$ F1 points but only $+0.29$ pp of judge accuracy over the root, whereas the judge-selected round-15 version gains $+3.17$ and $+5.56$. Reevaluated with its own F1 evaluator, this run flips 6.1% of outcomes with a 2σ of 3.2 pts, and 7 of the 9 steps on its selected version's lineage (78%) fall

within 2σ. GEPA has no F1-selected run. Every run splits its revisions into a first and a second half in production order, per conversation or scenario for EvolveMem and ARC and per seed for MemPro and GEPA (rounds 1–7 and 8–15 for MemPro). Table 23 scores every row of Table 3 by F1>0 and adds two F1-selected arms, the loop of the EvolveMem paper and a MemPro run that selects by F1 for fifteen rounds on the seed-2028 split. Judge-selected EvolveMem and ARC runs are in Table 25.

Table 23: All settings scored by F1>0: repair and regression volumes in pp of evaluated queries. Rows of Table 3, whose MemPro and GEPA runs select by the judge, plus two F1-selected arms. "No edit" is unchanged-system repair/regression volume with the number of comparison pairs; next loss is the share of new repairs and of controls that are wrong after the next update (%, as in Table 24).

Data	System	Early/late n	Repair E/L	Regress E/L	No edit (pairs)	Next loss
LoCoMo	EvolveMem	29/38	5.0/2.9	2.4/3.3	1.0/1.0 (20)	8.3/2.0
	ARC closed/r0	26/31	8.7/1.2	1.8/2.3	1.1/1.3 (31)	9.8/1.2
	ARC closed/inc	23/31	10.9/2.8	2.7/5.5	0.6/0.8 (3)	7.7/2.2
	ARC open/inc	20/23	7.6/2.2	3.7/3.0	1.2/1.9 (31)	0.0[§]/11.2
	MemPro	28/32	5.3/4.5	4.5/4.4	5.2/5.7 (10)	33.3/3.5
	GEPA	20/22	5.1/4.8	4.9/5.3	3.5/4.7 (9)	46.3/3.9
	EvolveMem paper loop[a]	20/26	7.9/4.2	3.5/3.1	2.0/0.0 (1)	23.6/2.9
	MemPro F1-selected	7/8	5.0/4.2	4.3/4.1	5.2/5.7 (10)	14.5/3.9
Gallery	EvolveMem	60/79	3.5/2.2	1.3/2.7	1.0/0.7 (10)	11.7/1.3
	ARC closed/r0	48/54	7.2/1.0	1.3/2.5	0.3/0.4 (23)	7.8/0.9
	ARC closed/inc	53/64	6.8/1.2	1.6/2.1	0.5/0.0 (3)	7.1/1.1
	ARC open/inc	34/47	4.4/1.8	1.4/2.4	0.6/0.5 (55)	3.4[§]/3.5
	MemPro	28/32	4.4/3.3	4.1/3.0	2.3/3.5 (1)	29.0/2.2
	GEPA	5/5	4.2/3.2	4.6/2.7	2.3/3.5 (1)	26.7[§]/0.9

[§]Fewer than 50 new repairs. [a]The non-elitist loop of the EvolveMem paper's Algorithm 1, instead of the released code's default elitist loop, from the same round 0.

Table 24: Share of new repairs (wrong at $t-1$, correct at t) and controls (correct at both) that are wrong at $t+1$ on accepted trajectories (%), by whether step $t+1$ changes configuration or memory. Rows and scores as in Table 3, with MemPro and GEPA scored by the judge.

		After an update			After no edit		
Data	System	n	New	Control	n	New	Control
LoCoMo	EvolveMem	145	8.3	2.0	33	6.1[§]	0.5
	ARC closed/r0	92	9.8	1.2	167	5.4	0.9
	ARC closed/inc	273	7.7	2.2	0	—	—
	ARC open/inc	13	0.0[§]	11.2	97	5.2	0.8
	MemPro	336	28.9	4.6	0	—	—
	GEPA	160	36.9	5.5	0	—	—
Gallery	EvolveMem	163	11.7	1.3	2	0.0[§]	1.5
	ARC closed/r0	179	7.8	0.9	87	1.1	0.1
	ARC closed/inc	211	7.1	1.1	2	0.0[§]	0.0
	ARC open/inc	29	3.4[§]	3.5	20	15.0[§]	0.2
	MemPro	344	33.4	2.5	0	—	—
	GEPA	35	28.6[§]	1.1	0	—	—

n: new repairs followed by that step type; [§]fewer than 50; —: no such step.

Table 24 follows individual repairs one accepted step ahead. The two groups are selected differently, since a new repair is defined by a recent flip, which selects less stable questions, whereas a control has two consecutive correct outcomes. The higher loss of new repairs therefore also appears without any edit (for example, 5.4% versus 0.9% for ARC closed/r0) and is not evidence that later edits target recent repairs, and only the adjacent step is measured. In a separate pure-reevaluation sequence, MemPro loses 12/25 (48%) of queries that just flipped correct, GEPA 14/31 (45%), and ARC open/inc 8/14 (57%), while controls lose 4.8%, 4.4%, and 1.1%, respectively. Those samples show why one flip should be verified before it is treated as a lasting repair. Other pure-reevaluation cohorts contain at most four newly correct queries and are not used for a broad comparison. Counts are adjacent

question-steps, which can share versions and queries, so no independence-based significance claim is made.

Table 25: Judge-selected EvolveMem and ARC runs, scored with their own judge: repair/regression volumes (pp) and next-update failure rates (%). MemPro and GEPA, which select by the judge natively, are in Table 3.

Data	System	Early/late n	Repair E/L	Regress E/L	Next loss: new/control
LoCoMo	EvolveMem	30/39	7.3/2.5	2.2/2.8	11.9/1.2
	ARC closed/r0	24/28	10.4/2.0	2.3/4.6	10.9/1.2
	ARC closed/inc	15/23	16.2/2.3	1.9/4.9	8.0/0.9
	ARC open/inc	18/22	4.9/1.3	2.2/2.5	0/0
Gallery	EvolveMem	59/79	3.8/4.0	2.9/4.5	26.2/3.3

ARC open/inc has only two newly repaired queries followed by an update in the judge-selected cohort. Gallery EvolveMem is an exception to declining repair volume under this selection metric. Selection metric and outcome metric must therefore be specified together, because cross-scoring a judge-selected tree with F1 answers a different question.

F.3 Held-out gains and turnover

Table 26 gives each split's held-out gain over its original root, and Table 27 gives the question-level turnover along the same lineages.

Table 26: Held-out gains of MemPro and GEPA by split. Gain (pp) of the selected version over each split's original root, Qwen judge. MemPro: 15 rounds; GEPA: its own prompt search.

System	Split 1	Split 2	Split 3	Split 4
LoCoMo				
MemPro	—	+3.03	+5.56	+3.39
GEPA	+4.18	−0.72	−0.65	−0.79
Mem-Gallery				
MemPro	+6.30	+2.34	+4.87	+5.84
GEPA	+0.84	—	—	—

—: not evaluated. Split 3 is the seed-2028 split used in Tables 1 and 4; its entry is the round-15 version v0014 against the original root evaluation (78.21%), whereas the E0 rerun of Table 4 is the round-10 incumbent of the same run against the reevaluated root (78.50%), and the E0 arm of that table is a separate ten-round run; Table 1b gives rerun variation. Seeds per split: Appendix B.

Table 27: Held-out turnover along the final MemPro lineages. LoCoMo, 1,386 held-out questions, Qwen judge. Transitions are summed over consecutive evaluated versions on the lineage of each split's final version.

Seed	Steps	W→C	C→W	Changed (%)	Net	Root rerun
2027	9	809	767	41.1	+42	150
2028	6	472	395	28.9	+77	162
2029	8	624	577	32.6	+47	157

W→C / C→W: wrong-to-correct and correct-to-wrong transitions summed over lineage steps; individual steps flip 108–236 questions. Changed: questions whose outcome changes at least once along the lineage. Net: end-to-end change in correct questions, equal to W→C − C→W; it corresponds to the +3.03/+5.56/+3.39 pp of Table 26. Root rerun: questions whose outcome differs when the original root is evaluated again.

F.4 Intervention arms

Tables 28–31 give the details behind Table 4. The initial ledger of the seed-2028 split, classified by Algorithm 2, contains 106 stable-correct, 22 fragile, six ghost, and twenty stable-wrong questions, totaling 154, and its two root evaluations score 72.73% and 79.22%, whose mean is 75.97%.

Feedback arms and targeting. E0 receives the loop's own evaluation output and per-query results, and the audit records are withheld from it. E2 receives repeated evaluation, replay, parent–child differences, and attempted-action records. E3 additionally receives attributable stages and evidence provenance. Root files, development split, and initial evaluation match across arms. E0/E2/E3 each complete ten rounds with the original selection logic. The reported stable-failure shares among agent-targeted questions are 69%/86%/96% across 13/81/28 target records. Among 91/98/106 observed repairs, 52%/66%/71% originate from parent queries classified as ghost or fragile, that is, from uncertain baseline states.

Table 28: All eight within-round repair comparisons with held-out results. Each difference pairs the repair child with its uncorrected revision on common questions. Positive and negative threshold crossings are both retained.

Arm	Round	Paired n	Repair − revision (pp)	z	Reported threshold
E2 + repair	2	1,377	$+2.98$	3.10	Positive
	3	1,359	$+6.25$	6.18	Positive
	4	1,379	$+2.10$	2.31	Positive
	5	1,371	$+1.24$	1.52	Within band
	7	1,367	$+0.73$	0.98	Within band
	8	1,382	-1.95	-2.54	Negative
E3 + repair	3	1,384	$+0.22$	0.26	Within band
	5	1,378	-0.65	-0.76	Within band

Common-query counts for Table 28 range from 1,359 to 1,384. Six pairs are positive, with three positive and one negative $|z| \geq 2$ result. The two E3+repair pairs have $z = 0.26$ and -0.76, both below the threshold. E2+repair generates seven repair children, of which five are accepted, and their mean stable repair/regression counts are 4.3/1.0 versus 4.1/2.7 for the original revisions. E3+repair generates nine repair children, none of which is accepted, with corresponding means of 3.0/2.7 versus 3.1/3.0. Thus, the same repair design has different outcomes across the two arms. Table 29 lists the incumbents that E2+repair selects and their held-out gains.

Table 29: Selected E2+repair incumbents. Held-out gains use each version's common-query pairing with the root. Later improvements on development do not imply a monotone held-out trajectory.

Version	0000	0002	0004	0008	0010	0011	0015	0016	0017
Dev. (%)	75.97	82.14	84.09	87.98	89.29	91.16	91.86	93.18	92.84
Held-out gain	0	+0.22	+3.76	+1.81	+3.05	+1.88	+1.37	+2.34	+3.44

Budget. The arms are matched in rounds rather than in cost, and Table 30 reports what each one consumed. Verified feedback adds one full reevaluation per version and a temperature-0 replay of its 20–40 wrong answers, which the harness runs between rounds, so E2 and E3 evaluate the development set twice per version while the agent's own sessions cost about as much as under E0 (22–27 hours and 120–132 million kimi-k3 tokens per arm). Repair rounds nearly double the rest, since each repair child is a further agent session with its own evaluation and reevaluation, and the two repair arms produce 17 and 19 versions, spend 26 additional agent hours, and use 265–375 million tokens. The E0 rerun of Table 4 therefore matches the arms in rounds only. A rerun of the original loop that evaluated each version twice and kept the better reading would cost about what E2 costs, and we have not run it. Wall-clock is not comparable across arms, because all five ran concurrently on one API quota.

Checkpoints. The round-10 incumbents of Table 4 are E0 v0009, E2 v0009, E3 v0009, E2+repair v0017, and E3+repair v0014, which E3+repair keeps as its incumbent through rounds 8–10.

Development-set resolution. Table 31 sets the resolution of the development set against the size of the accepted steps. On fixed root predictions for 1,540 questions, 20,000 resamples of 154 questions yield a standard deviation of 3.15 pp and a reported 95% interval of [72.08, 84.42]%. Four actual split scores are 79.87/77.27/82.47/75.97%, against a full-pool score of 78.64%. Five same-root

Table 30: Budget of the intervention arms. MemPro, seed 2028, ten rounds. Dev. evaluations count full 154-question evaluations, the agent's own (A) plus the harness reevaluation (B). Replayed: wrong answers replayed at temperature 0 and probed for their stage. Held-out: 1,386-question evaluations. Hours are active session time of the evolving agent, of the repair sessions, and of the harness hooks (B, replay, ledger), and wall-clock from the first to the last round. Agent tokens are kimi-k3 input plus output over all sessions, of which 97–98% are cached input.

		Dev. evals				Hours				Agent
Arm	Versions	A	B	Replayed	Held-out	Agent	Repair	Hooks	Wall	tokens (M)
E0: MemPro loop	10	10	10[a]	238[a]	5	25.5	—	13.7[a]	28.4	132
E2: + ledger	10	10	10	273	4	22.2	—	9.2	32.7	120
E3: + failure stages	10	10	10	206	4	26.5	—	11.6	40.6	130
E2 + repair	10 + 7	17	17	314	16	27.0	25.7	11.0	67.1	375
E3 + repair	10 + 9	19	19	390	7	26.4	26.3	15.2	69.8	265

Versions: agent revisions plus repair children. [a]E0's reevaluations, replays, and hooks were run for the audit only and never shown to the agent. Held-out evaluations of E2 + repair cover every accepted or revised version for Table 28. All arms shared one API quota with other experiments, so wall-clock includes rate-limit waits, and the same 154-question reevaluation took between 10 and 250 minutes. Calls and tokens of the task model and judge were not recorded.

reevaluations on one fixed split score 74.03/78.57/79.22/79.87/81.82%, giving $2\sigma = 5.76$ pp. The root evaluation observed by the loop scores 72.73%, and including it widens the band to about 7.1 pp, so this evaluation is reported separately. The five gold-recovered stable failures use the audited root's stability classification and evidence interface, and they do not bound the effect of changing prompts, code, or answer formatting. If a fraction ρ of N questions flips between two evaluations of one version, a single comparison has a 2σ band of about $2\sqrt{\rho/N}$, the binomial band of Table 1a, so a loop that evaluates each version once resolves a step of size Δ only if $N \geq 4\rho/\Delta^2$. At MemPro's flip rates of 8.7–13.2%, the late steps of 0.65–1.3 pp would need roughly 2,000 to 12,500 development questions, more than the 1,540 in LoCoMo.

Table 31: Development-set resolution for MemPro seed 2028, against the size of accepted steps.

Quantity	pp
Resample the 154 questions, 2σ	6.3
Reevaluate the root, 2σ	5.8
Gold-recoverable failures (5)[a]	3.2
Accepted step, rounds 1–5	1.3–4.5
Accepted step, rounds 6–10	0.65–1.3
One question	0.65

Accepted steps: their net stable repairs in the repair arms. [a]Root stable failures recovered when answered from their annotated gold turns under a neutral prompt. Fuller evidence recovers more, and the value is not a ceiling on code or interface changes.

G Case Studies: Four Failed Edits

This section traces four edits at the question level and shows by which route an edit breaks questions that its parent answered correctly.

Table 32 traces four edits that lower the score, one for each kind of operator, one of which targets its parent's bottleneck stage. For every question that the parent answers correctly and the child answers wrong, we check whether its gold memory still reaches the child's answer context.

Cases A and B, one aligned and one not, lose correct answers in the same way, as the edit reorders or shortens the retrieved context and pushes the gold memory out. In A, the parent answers "Which country was Evan visiting in May 2023?" with "Canada", and the child answers "Not enough information" once the memory of the road trip to Jasper leaves the context. In B, "In what country did Jolene's mother buy her the pendant?" changes from "France" to "Colombia" for the same

Table 32: Four edits that lower the score and the route of their regressions. Context changed: parent-correct questions whose retrieved context changes. Gold in context: parent-correct questions whose gold memory reaches the answer context, in the parent and in the child.

	Edit (stage)	Aligned	Fixed/broken	Context changed	Gold in context	Main route of the regressions
A	EvolveMem, intent planning (S2)	yes	6/21	123/124	33→17	gold memory displaced (15 of 21)
B	ARC closed/inc, time decay and fusion weights, context 20→15 (S2)	no	5/60	158/158	83→36	gold memory displaced (45 of 60)
C	EvolveMem, targeted re-extraction (S1)	no	4/4	41/116	4→3	new memories in context (2 of 4)
D	ARC open/inc, per-category answer prompt (S3)	no	11/15	0/139	45→45	answer changes with gold in context (13 of 15)

reason. Cases C and D keep most gold memories in the context and change the answer instead. In C, a new near-duplicate of the gold memory enters the context and the answer to "How did Joanna feel on October 25, 2022 about seeing her characters come alive on the big screen?" drifts from "Surreal and satisfying" to a paraphrase of that memory, and the same question breaks in both targeted re-extractions from this parent. In D, the rewritten answer prompt leaves every context intact, yet "Which outdoor gear company likely signed up John for an endorsement deal?" changes from "Under Armour" to a refusal, although the supporting memory is still in the context. Whether an edit targets the bottleneck stage thus does not decide how it breaks other questions, since the routes follow the shared component it changes.

H LIMITATIONS

Our evidence supports improvements in specific parts of the loop and suggests conditions under which they would accumulate. Our stage reference is conservative, as it abstains on failures that neither replay nor gold evidence resolves, so its conclusions cover only about a third of observed failures. The three stages also form a coarse partition chosen to apply across systems. This partition, however, already reveals clear trends (Section 4). Stage alignment is descriptive and depends on the parent state, so the effect of an edit has to be read from paired outcomes. Repair retention is measured over adjacent steps rather than arbitrarily long horizons. We audit the three links on development questions, since every decision of the loop rests on them, so a loss at any link on these questions already limits what selection can pass on to held-out questions. Held-out questions therefore serve mainly to measure the gains of selected versions, and tracing which development repairs and regressions carry over to them is left to future work. MemPro's and GEPA's held-out questions share conversation histories with the development set, so their held-out gains do not establish transfer to unseen conversations. Improvements of a repair child over its revision are distinct from improvements over the incumbent or over another arm. The intervention comparison uses one split and one model setting. Finally, still larger development sets and acceptance data withheld from proposal feedback remain to be tested, and a method selected after repeated inspection of held-out results would need a fresh test set.