# VESTRUM: IMPROVING AGENT HARNESSES BY ADAPTING THEIR VERIFICATION, STRUCTURE AND MEMORY

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

An agent harness controls how a language model accesses information, uses tools, preserves memory, and checks its work. Improving this software is costly when each evaluation requires a long interaction with an environment. We introduce Vestrum, a framework that turns failures in execution traces into scoped harness changes without training the task model. Its organizing overhypothesis is that tasks of a shared kind may exhibit recurring failures whose remedies transfer within that kind. Vestrum expresses failures as recognizable classes, proposes changes across verification, retrieval, decomposition, and knowledge synthesis, and screens their scope before evaluating them as a bundle. A persistent lessons file informs subsequent proposals. Across five settings and two baseline harnesses, the frozen harnesses improve held-out performance: UltraHorizon rises from 47.6 to 59.8 over GAM, Terminal-Bench 4 Hard from 63.7% to 70.3% of checks passed over Claude Code on eight held-out tasks at $1.03\times$ test cost, and cell-type annotation agreement from 67.5% to 77.8% on held-out sections of one slide, alongside gains on LoCoMo and AMA-Bench. Across our searches, verification grounded in evidence helped both intermediate steps and final answers, at lower cost at intermediate steps, while critics asked to rebuild finished answers broke more than they repaired. On the three memory benchmarks, Vestrum also scores above the evaluated GEPA configurations in every paired evaluation.

## 1 INTRODUCTION

An agent harness is the software around a language model that controls how it reads its environment, uses tools, preserves information, and decides when its work is done. These decisions shape what the model can accomplish over a long task: which evidence remains available, how work is divided, and whether an intermediate result is checked before later steps depend on it. A memory system can itself be a harness. General Agentic Memory (GAM), for example, pairs a memorizer that records an incoming stream with a researcher that retrieves and reasons over the resulting store (Yan et al., 2025a). We study how to adapt GAM and Claude Code from limited development experience, and what their resulting changes reveal about the design of verification.

A long episode presents both a cost and an opportunity. It may contain hundreds of tool calls, with an early mistake propagating through subsequent decisions (Luo et al., 2025). Its final score summarizes the outcome, but its execution trace records what the agent retrieved, what it preserved, and which assumptions it acted on. Those details can suggest a precise intervention. The difficulty is to turn a diagnosis of one episode into a change that remains useful elsewhere, without repeatedly paying for complete episodes to test every idea.

Vestrum approaches this problem through an *overhypothesis*: a higher-level expectation that guides what can be learned from individual examples (Goodman, 1955; Kemp et al., 2007). Here the expectation is that tasks of a shared *task kind* may exhibit recurring *failure classes*, and that a remedy for such a class may transfer within that kind. Task kinds specify where a remedy could apply; failure classes describe behavior recognizable in traces. A proposed harness change, which we call a *mechanism*, connects the two. A generalization check examines whether its condition and content extend beyond the originating example, while empirical evaluation determines whether the resulting



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.

harness improves. The overhypothesis guides categorization and scope; it is an inductive bias, not a guarantee of transfer.

Four *design dimensions* organize the possible interventions. Verification concerns what is checked and when a result can be used; retrieval concerns what information the agent accesses and in what form; decomposition concerns how work is divided; and knowledge synthesis concerns what the harness distills and records for later use. A localized failure can suggest changes in several dimensions, from a prompt or a tool to a memory operation or a control-flow constraint. Vestrum evaluates scoped mechanisms together, refines the bundle, and retains it only when the admission criteria are met. A persistent lessons file carries experience from successful and unsuccessful proposals into later epochs.

This design builds on trace-driven harness revision (Lee et al., 2026; Lin et al., 2026; Chen et al., 2026a), reflective lessons (Shinn et al., 2023; Zhao et al., 2024), failure attribution and taxonomy-guided search (Cemri et al., 2025; Zhang et al., 2025c; Cemri et al., 2026), and reflective prompt evolution (Agrawal et al., 2026). Its contribution is an operational framework connecting localized failures, task-kind scope, four design dimensions, search memory, and empirical admission. Appendices A and F place these choices in the wider literature.

Vestrum also exposes verification as an adaptable part of the harness. The learned checks act at *points of commitment*: writing a memory, merging evidence, accepting a rule, or editing code. These are moments when later computation begins to depend on an intermediate result.

We evaluate frozen harnesses in five settings spanning coding, agentic memory, and cell-type annotation. Memory results improve over GAM across the reported test repetitions, with the task models used during search differing from the model used at test. Coding and annotation provide applications starting from Claude Code. We also compare with GEPA on the memory benchmarks. In the evaluated configurations, Vestrum obtains higher scores with fewer estimated task-agent evaluations on LoCoMo and AMA-Bench and comparable counts on UltraHorizon. This advantage concerns task executions: learning-token use is higher on LoCoMo and UltraHorizon, and the configured search spaces differ.

**Contributions.**

- **Overhypothesis-guided harness adaptation.** Vestrum connects task kinds and failure classes to scoped mechanisms across four design dimensions, screens their generality, and evaluates bundles while retaining search lessons.
- **Evidence-grounded verification.** Across four settings, our searches showed a consistent pattern: verification helped when it was grounded in evidence. Checks at intermediate steps repaired 17 of the 18 AMA-Bench notes they checked (with GPT-4o-mini), gave the best LoCoMo development pipeline, and entered the UltraHorizon bundle that raised validation from 26.7 to 33.3, while acting on only 12–13% of LoCoMo questions. A selective, evidence-grounded final-answer check was also net positive (6 repaired, 4 broken). By contrast, critics asked to re-verify and rebuild answers already given overturned more correct answers than they fixed (9 against 5; 6 against 2). The pattern points to grounding checks in evidence and to placing them where intermediate results are committed, where they cost least.
- **Empirical gains across five settings and two baseline harnesses.** The learned harnesses improve held-out performance over their starting harnesses across coding, agentic memory, and cell-type annotation. On the memory benchmarks they also score above the evaluated GEPA configurations with fewer or comparable estimated task-agent evaluations, at higher learning-token use on two benchmarks.

## 2 Overhypothesis-Guided Harness Adaptation

Vestrum maintains a current harness and a persistent record of search lessons. Each epoch uses development traces to diagnose failures, propose scoped mechanisms, and evaluate a bundle of changes. Four design dimensions structure the proposals; the generalization check screens their formulation before execution. The harness changes only when a candidate meets the admission

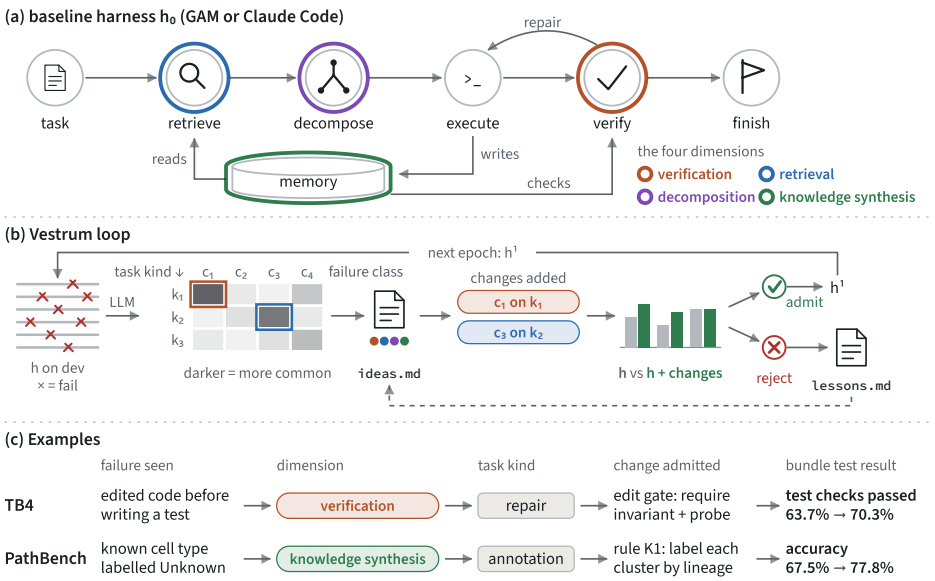



Figure 1: **(a)** The four design dimensions within an agent harness. **(b)** Development failures are organized by task kind and failure class; darker cells indicate recurrence across more tasks. Scoped mechanisms are composed, screened, and evaluated, with lessons carried into subsequent search. **(c)** Two example mechanisms and the results of their containing harnesses. The reported gains belong to the evaluated bundles, not to the illustrated mechanisms alone.

criteria (Figure 1; Algorithm 1). Vestrum uses a general adaptation loop across all five settings, with benchmark interfaces for executing, scoring, and categorizing tasks.

### 2.1 Task kinds, failure classes, and search state

Starting from a baseline harness $h_0$, GAM or Claude Code, the search maintains the current harness $h$ and lessons $L$. Mechanisms may change prompts, tools, memory operations, or control flow. Within a search, the task model, grader, and benchmark checks remain fixed, and no model is trained. Development tasks supply traces and outcomes; validation, where used, is disjoint from development and test. Test outcomes never return to the loop.

Task kinds are assigned before proposals using task descriptions or benchmark categories. On Terminal-Bench a task kind combines task mode, such as repair or build, with the check style described in the instruction. On the memory benchmarks it is an environment, question category, or probe axis. Failure classes describe the behavior a mechanism should address. Neither categorization is redefined to justify a proposal. Together they operationalize the overhypothesis: the task kind defines a possible scope of transfer, and the failure class states what may recur within it.

### 2.2 From traces to recognizable failures

A failed check, whether a benchmark test or a judged probe, identifies a development outcome that needs explaining. A language model reads the trace, localizes the apparent failure, and expresses it as a predicate recognizable in other traces, building on structured failure analysis (Cemri et al., 2025). For example, a repair agent may change a shipped file before establishing the invariant the code must preserve. Here *repair* is the task kind and *changing code before establishing the invariant* is the failure class. The predicate describes behavior without prescribing its remedy: several mechanisms may address the same class, and one mechanism may address its occurrences across tasks.

### 2.3 PROPOSALS ACROSS FOUR DESIGN DIMENSIONS

For each failure class within a task kind, the proposer reads the lessons and considers verification, retrieval, decomposition, and knowledge synthesis. These dimensions identify aspects of the harness that can change; a mechanism specifies the actual intervention. In the repair example, verification suggests an edit gate requiring an invariant and a probe script before the first change to a shipped file. Pilot experiments motivated the dimensions: open-ended proposals rarely produced useful early changes. They guide the search without requiring the final harness to contain a mechanism in every dimension.

Each proposal states its target failure class, applicable task kinds, recognition predicate, and predicted check improvements. A language model scores proposals, prioritizing classes observed on more tasks, and promising mechanisms are composed into a bundle $S$. We write $h + S$ for the current harness with that bundle applied, each mechanism restricted to its stated scope. Selection, composition, and generalization screening precede execution; Appendix D describes the combined composer used on Terminal-Bench.

### 2.4 GENERALIZATION SCREENING AND SCOPE

The overhypothesis provides an inductive bias for learning from limited experience (Goodman, 1955; Kemp et al., 2007): recurring failures within a task kind may admit shared remedies. The generalization check asks whether a proposal expresses such a remedy. A language-model reviewer examines whether its condition and content apply beyond the originating example, and a lexical leakage lint rejects task-specific names, symbols, paths, and constants. Permissible supervision is specified in Appendix E.1.

Passing this check makes a mechanism eligible for evaluation; it does not establish transfer. Task-kind scope separately determines where it executes: the repair gate applies to repair tasks, not to every request. Thus categorization supplies the proposed scope, screening examines what the mechanism encodes, and evaluation tests the resulting harness. Appendix E develops this connection.

### 2.5 BUNDLE EVALUATION AND ADMISSION

The loop evaluates $h + S$ against the current harness on development tasks, reusing existing runs of the current harness where available. Trace analysis identifies mechanisms that did not act or appeared neutral or harmful, guiding a pruned bundle $S'$. If pruning changes the bundle, $h+S'$ is also evaluated on development. These diagnoses guide refinement; admission rests on the performance of the evaluated bundle.

Where validation is used, development results determine which candidates advance to it. The pruned bundle is validated only if it outperforms the full bundle on the development metric. A candidate replaces $h$ only after satisfying the admission criteria, including validation where used; otherwise $h$ remains. The criteria are set per setting (Table 8). On Terminal-Bench, for example, a bundle must pass checks that Claude Code fails in every *draw* (one run of a task), lose no check on a task of another kind, and cost under $1.5\times$ the baseline per task; Appendix D gives the reported epoch.

A Terminal-Bench search illustrates how scope affects admission (Table 3, third row). A mechanism that wrote reference notes and project instructions on every task was refused after a task of another kind lost a check. Restricting the same mechanism to repair tasks, where its failure class had been observed, allowed it to be admitted. This development case shows how evaluation can revise a mechanism's scope while preserving its useful application.

### 2.6 LEARNING ACROSS EPOCHS

Algorithm 1 summarizes one epoch. Its lessons file records successes, failures, and repeated refusals for subsequent proposals, separately from task memory. Epochs repeat with the retained harness and updated lessons; the final harness is frozen before test evaluation. Appendix D gives the Terminal-Bench prompts.

Here $R$ contains execution traces and scores. DIAGNOSE returns failure classes and recognition predicates; PROPOSE searches the four design dimensions with task-kind scope, and SCREEN ap-

**Algorithm 1** Vestrum: one epoch of harness adaptation

**Require:** harness $h$; development tasks $D$; task kinds $K$; lessons $L$
**Require:** optional validation tasks $V$
$R_0 \leftarrow \textsc{EvaluateOrReuse}(h, D)$
$F \leftarrow \textsc{Diagnose}(R_0, K)$
$\mathcal{D} \leftarrow \{\text{verification, retrieval, decomposition, knowledge synthesis}\}$
$P \leftarrow \textsc{Propose}(F, K, L, \mathcal{D})$
$S \leftarrow \textsc{RankAndCompose}(P); \quad S \leftarrow \textsc{Screen}(S)$
$R \leftarrow \textsc{Evaluate}(h + S, D)$
$\mathcal{C} \leftarrow \{(h + S, R)\}$
$S' \leftarrow \textsc{Prune}(S, R_0, R)$
**if** $S' \neq S$ **then**
    $R' \leftarrow \textsc{Evaluate}(h + S', D)$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(h + S', R')\}$
**end if**
$h' \leftarrow \textsc{AdmitBest}(h, R_0, \mathcal{C}, V)$
$L' \leftarrow \textsc{UpdateLessons}(L, F, \mathcal{C}, h')$
**return** $(h', L')$

plies the generalization check. ADMITBEST applies the admission criteria and optional validation, considering a pruned bundle for validation only if it improves on the full bundle during development; it returns $h$ if no candidate qualifies.

## 3 HELD-OUT PERFORMANCE AND EVALUATION RESOURCES

We evaluate the harnesses produced by Vestrum in five settings. GAM provides the starting point for UltraHorizon, LoCoMo, and AMA-Bench, with A-Mem as an additional comparator (Yan et al., 2025a; Xu et al., 2025); Claude Code provides it for Terminal-Bench 4 Hard and PathBench. All five settings use the shared loop in Section 2; Table 7 summarizes the splits and evaluation units, and Table 8 records each setting's search protocol. UltraHorizon decides admission on fresh validation seeds, each used once; LoCoMo and Terminal-Bench use validation to check development gains; AMA-Bench and PathBench rely on paired development comparisons. One selected harness per setting was frozen before test evaluation. No test outcome returned to search; development, validation where used, and test tasks are disjoint.

Claude Opus 5 optimized the memory harnesses using Claude Haiku 4.5 for task execution on UltraHorizon and LoCoMo, and GPT-4o-mini on AMA-Bench. All memory systems were then tested with Gemini 3.7 Flash, so the learned harnesses are evaluated across a change of task model. Terminal-Bench uses Claude Sonnet 5 and PathBench uses Claude Opus 5 for task execution during both adaptation and testing.

A blinded judge grades memory answers against a fixed rubric by majority vote over three passes (Appendix I). Repetitions measure variation in execution and judging of frozen harnesses, rather than independent searches.

### 3.1 AGENTIC MEMORY: ULTRAHORIZON, LOCOMO, AND AMA-BENCH

The three memory benchmarks exercise different uses of retained information. UltraHorizon requires discovering environment rules over long interactions (Luo et al., 2025); LoCoMo asks questions about extended conversations (Maharana et al., 2024); and AMA-Bench asks questions about tool-using trajectories (Zhao et al., 2026). On LoCoMo and AMA-Bench, *judge accuracy* is the share of answers graded fully correct, each assessed against the reference answer's key facts. Appendix H specifies the question categories; LoCoMo excludes the adversarial, unanswerable category.

**Performance across settings.** The learned harness improves over GAM on all three benchmarks (Table 1). UltraHorizon rises from 47.6 to 59.8, a 12.2-point gain with a positive difference at every seed. LoCoMo rises from 71.5% to 74.9% across three replicates, with net gains of $+24$, $+39$, and $+34$ probes; lexical F1 also improves in each. AMA-Bench improves in both generation runs, by

Table 1: Memory benchmark performance with Gemini 3.7 Flash (±: standard deviation). Upper block: test repetitions of each frozen harness; the AMA-Bench row is the second answer-generation run, and the difference row gives both runs. Lower block: a separate comparison in which GAM, GEPA, and Vestrum were rejudged together, so GAM and Vestrum scores differ slightly from the upper block; GEPA differences are computed within it.

| | UltraHorizon score / 100 | LoCoMo judge acc. | AMA-Bench judge acc. |
|---|---|---|---|
| A-Mem | 45.7 | $49.3 \pm 0.6\%$ | 56.2% |
| GAM | 47.6 | $71.5 \pm 0.6\%$ | 64.2% |
| **Vestrum** | **59.8** | $\mathbf{74.9 \pm 0.8\%}$ | **74.2%** |
| Vestrum − GAM | +12.2 (3/3 seeds) | +3.4 (3/3 replicates) | +12.1, +10.0 (2 runs) |
| *GEPA comparison* | | | |
| GAM | $47.6 \pm 0.7$ | $72.5 \pm 0.7\%$ | $65.7 \pm 0.9\%$ |
| GEPA on GAM | $47.9 \pm 2.0$ | $74.0 \pm 0.8\%$ | $70.8 \pm 1.5\%$ |
| **Vestrum** | $\mathbf{59.8 \pm 4.7}$ | $\mathbf{76.3 \pm 0.7\%}$ | $\mathbf{76.4 \pm 0.9\%}$ |
| Vestrum − GEPA | +11.9 (3/3 seeds) | +2.3 (3/3 seeds) | +5.6 (3/3 judging runs) |

+12.1 and +10.0 points. Its second run, used for the three-way comparison, rises from 64.2% to 74.2%, with the largest margin on state abstraction. Appendices J.1–J.3 give the complete paired records.

**Comparison with reflective prompt optimization.** GEPA reflects on traces to evolve prompts (Agrawal et al., 2026). We ran it on the same development data, using Gemini 3.7 Flash as the task model and Claude Opus 5 for reflection. In this configuration, memory construction remains fixed; Vestrum can additionally change memory operations, tools, and control flow. The comparison therefore evaluates these configurations, rather than isolating the effect of the search algorithm.

Vestrum scores above GEPA in every reported paired evaluation, with mean margins of 2.3 points on LoCoMo, 5.6 on AMA-Bench, and 11.9 on UltraHorizon. GEPA improves AMA-Bench by 5.1 points over GAM; Vestrum's further gain accompanies mechanisms such as deterministic trajectory views, outside the configured prompt search. On UltraHorizon, GEPA's search over 60 episodes retained GAM's original prompt, so its test results are new executions of that program (lower block of Table 1).

**Task evaluations and learning tokens.** Under the accounting in Appendix L, Vestrum uses about a quarter of GEPA's estimated task-agent evaluations on LoCoMo and AMA-Bench and the same number on UltraHorizon. These counts describe task executions and include a $1.5\times$ allowance per Vestrum epoch for pruning and retesting. They are not a measure of equal learning compute: the searches use different task models, and analysis also consumes tokens. Reported learning-token use is approximately 30M versus 22.8M on LoCoMo, 9.5M versus 15.7M on AMA-Bench, and 74M versus 21.9M on UltraHorizon, for Vestrum and GEPA respectively. The result is a favorable task-evaluation tradeoff for the evaluated configurations, with higher token use on two benchmarks. Appendix L gives the counts, stopping conditions, and uncertainty.

### 3.2 TERMINAL-BENCH 4 HARD

Terminal-Bench grades software tasks using tests hidden from the agent (Merrill et al., 2026). Because neither system passes every check on any held-out task, we report the share of checks passed. On eight held-out tasks with two draws per system, Vestrum increases checks passed from 63.7% to 70.3%, with test cost at $1.03\times$ Claude Code under the same per-task spend cap. Four task means improve, four are unchanged, and none declines (Table 2).

The learned harness classifies each request and supplies four scoped mechanisms before the first turn (Table 5). Table 3 ablates the loop by removing its components. Removing the lessons file lowers development gains from 16 to 10 checks and validation gains from 11 to 6, at $1.6\times$ cost. Removing the ceiling probe, which runs each development task once with its solved procedure to

Table 2: Terminal-Bench 4 Hard: test checks passed on the eight test tasks, mean over two draws per system.

| task | checks | Claude Code | Vestrum | Δ |
|---|---|---|---|---|
| hof-topology-interpenetration | 38 | 21.5 | 25.0 | +3.5 |
| bun-sourcemap-leak | 36 | 24.0 | 26.5 | +2.5 |
| cargo-flight-dispatch | 27 | 17.0 | 19.0 | +2.0 |
| sglang-qwen-burst | 13 | 5.0 | 7.0 | +2.0 |
| vllm-deepseek-streaming | 5 | 1.0 | 1.0 | 0 |
| vpp-loss-divergence | 5 | 4.0 | 4.0 | 0 |
| glycan-ms2-elucidation | 12 | 11.0 | 11.0 | 0 |
| photonic-waveguide-routing | 14 | 12.0 | 12.0 | 0 |
| total | 150 | 95.5 | 105.5 | +10.0 |

Table 3: Terminal-Bench 4 Hard: ablation of the loop. The second row removes only the lessons file from the full loop; below the line, components are removed cumulatively from the full loop, each row also lacking those removed above it. A gain is a development check Claude Code fails in every draw and the candidate passes in every draw; a loss is the reverse. Cost is task spend relative to Claude Code; validation counts checks gained on tasks disjoint from test.

| loop | mechanisms found | gain | loss | cost | validation |
|---|---|---|---|---|---|
| full loop | repair procedures; edit gate; scale decision | +16 | 1 | 1.32× | +11 |
| − lessons file | all other loop components retained | +10 | 0 | 1.6× | +6 |
| − ceiling probe | scoped reference notes; fixture rule | +3 | 0 | 0.86× | +5 |
| − failure classes, generalization check | adversarial checks; non-author reviewer | +2 | 0 | 1.7× | 0 |
| − localization | constraint checks; pre-planning probe | +9 pts[†] | – | 2.3× | 0 |
| − trace access (final outputs only) | audit checklist; two-phase audit | +1 | 0 | 2.1× | 0 |

[†] Percentage points of mean check share from single draws; individual checks cannot be counted.

find failures that procedural help can remove, lowers validation gains to 5; removing failure classes, generalization screening, and task-kind scope as well leaves no validation gain (Appendix C).

### 3.3 PathBench: reconstructing expert cell-type annotations

PathBench evaluates whether an agent can reconstruct expert cell-type annotations from spatial transcriptomics measurements. We use CosMx data from the AnchorR study (Morris et al., 2026), comprising six tissue sections from one head-and-neck squamous cell carcinoma (HNSC) slide. A bioinformatician's Seurat-based analysis pipeline supplies the reference cell labels, an expert-in-the-loop annotation rather than independent ground truth. Given the measurement export and seven local reference sources, the agent must recover tissue sections, filter and cluster cells, and assign cell-type labels. Development uses two sections; evaluation uses the remaining four, and no test-section label or measurement was read before the harness was frozen.

This task extends harness adaptation to a scientific workflow in which success requires combining computational analysis with domain knowledge. Development traces showed that the agent left clusters unlabelled despite evidence for their lineage. Vestrum supplied eleven sourced annotation rules, marker and expression reference tables built from the development sections, and a nearest-neighbour helper before the agent planned its analysis. On held-out sections, agreement with the expert labels increases from 67.5% to 77.8%, and macro-F1 from 64.8 to 76.5 (Table 4), with two runs per system. This measures transfer to other sections of the same slide. Appendix K details cell matching, label mapping, and scoring.

Table 4: PathBench performance against expert annotations, averaged across execution runs ($n$). The rules and reference tables were written from the development sections; test sections were unseen until the harness was frozen.

| split | system | $n$ | ARI | NMI | accuracy | macro-F1 |
|---|---|---|---|---|---|---|
| development | Claude Code | 3 | 0.582 | 0.662 | 66.5% | 59.0% |
| | Vestrum | 2 | 0.670 | 0.717 | 82.2% | 80.4% |
| test | Claude Code | 2 | 0.642 | 0.713 | 67.5% | 64.8% |
| | **Vestrum** | 2 | **0.684** | **0.725** | **77.8%** | **76.5%** |

Table 5: Operations learned by the shared adaptation loop across four design dimensions. A dash indicates no admitted mechanism in that dimension.

| Setting | Verification | Retrieval | Decomposition | Knowledge synthesis |
|---|---|---|---|---|
| UltraHorizon | Replay evidence before commitment | Pin notebooks by rule class | Re-analyze rules; plan probes | Record evidence with provenance |
| LoCoMo | Check claims at evidence merges | Retrieve again when evidence is insufficient | Split questions selectively | Prioritize evidence; give a best-supported answer |
| AMA-Bench | Check notes against source records | Index events and object histories | – | Build notebooks during ingestion |
| PathBench | – | – | – | Supply annotation rules and reference tables; reassign small unlabelled clusters |
| Terminal-Bench | Require invariant and probe before edits | – | – | Supply repair procedures; choose raw or log scale |

## 4 What the Learned Harnesses Reveal

The shared loop produces mechanisms across the four design dimensions, adapting the information an agent uses, the procedures it follows, and the checks on its actions (Table 5). Each setting's bundle was learned separately. Appendix M describes the mechanisms in full, and Figure 2 shows the resulting harnesses.

These artifacts give the framework a concrete interpretation. A trace can motivate a change to the evidence available to the agent, the procedure that uses it, or the conditions under which the agent acts. Verification intersects with all three: a check needs evidence to consult, a place in the workflow, and a consequence when it fails.

### 4.1 Verification: placement, evidence, and scope

Our searches show a consistent pattern: verification helps when it is grounded in evidence, and it is cheapest where an intermediate result is committed. Each retained verifier acts at a point of commitment: a memory write on AMA-Bench, an evidence merge on LoCoMo, a rule commitment on UltraHorizon, or the first edit of a shipped file on Terminal-Bench. The checks repair an intermediate record or withhold an action, drawing on recorded evidence or a harness-enforced prerequisite. Table 6 summarizes the retained mechanisms. The UltraHorizon gate entered in the bundle that raised the development score from 16.7 to 30.0 and validation from 26.7 to 33.3. Appendix N records the development alternatives, models, and outcomes.

**Broad rewriting can undo correct work.** In development, an AMA-Bench checker that rechecked every final answer repaired 5 answers and broke 9. A LoCoMo answer-level checker similarly improved 2 and regressed 6. These comparisons motivate limiting when a checker intervenes and what it may change.

Table 6: Verification mechanisms retained in the evaluated harnesses.

| Setting | Checkpoint | Evidence or constraint | Action |
|---|---|---|---|
| AMA-Bench | memory write | extracts, cited turns, action ledger | repairs notebook entries |
| LoCoMo | evidence merge | spans retrieved for each claim | checks and revises claims |
| UltraHorizon | rule commitment | replay of logged observations | withholds commitment |
| Terminal-Bench | first edit of shipped code | invariant and probe must exist | blocks edits lacking prerequisites |

**Checks need evidence that supports the decision.** The retained UltraHorizon gate replays logged observations before accepting a rule. A different gate at the same stage, based on evidence sufficiency, blocked every development episode and lost 10 points. Placement alone therefore did not ensure a useful check: the evidence and acceptance condition must allow the agent to make progress. The learned mechanisms make these dependencies explicit, from cited source turns for notebook entries to executable prerequisites for code edits (Table 6).

**Selective intervention can preserve useful progress.** Restricting the AMA-Bench final-answer checker to counting and state questions, while preserving the draft unless contradicted, yielded 6 repairs against 4 regressions. This positive variant was later superseded by notebook verification. On LoCoMo, the pipeline checking only evidence merges scored highest overall (0.796), four answers net ahead of each alternative. It also changed routing; on a second conversation it led GAM (0.836 vs 0.809).

Held-out gains are measured for complete harnesses, and no verifier was evaluated alone at test; Appendix N reports each retained check's activity and failure record.

Taken together, evidence determines whether a check helps and placement determines what it costs: grounded checks at points of commitment protect the records later steps reuse, grounded final-answer checks repair answers selectively, and critics that rebuild finished work without independent evidence undo correct answers. This interpretation connects the harness setting to process and outcome supervision (Uesato et al., 2022; Lightman et al., 2024), unreliable self-correction without external feedback (Huang et al., 2024; Kamoi et al., 2024), and terminal checks that withhold release without rewriting accepted answers (Zhang et al., 2026f).

## 5 Discussion and Limitations

Vestrum connects an overhypothesis about recurring failures to scoped changes across four design dimensions. The evaluated harnesses improve held-out performance, and their development histories make verification's dependence on evidence and scope visible.

The evidence has several boundaries. One selected harness per setting was evaluated, so test repetitions do not measure variability across independent searches. The design dimensions were motivated by pilots, and the Terminal-Bench ablation (Table 3) uses one search per variant. Its test set is small, while PathBench measures agreement with laboratory annotations on sections of one slide. LoCoMo excludes unanswerable questions despite retaining a rule that encourages a best-supported guess; its gains therefore concern the evaluated answerable categories. The generalization check screens proposals without guaranteeing semantic separation or transfer.

Verification was evaluated in bundles, and preliminary alternatives sometimes used earlier task models. No verifier was ablated at test. GEPA uses a different editable surface and search model, and both its estimated evaluation comparison and the reported learning costs should be read under the accounting in Appendix L; the Terminal-Bench cost of $123.5 (Appendix D) covers the reported epoch, including its test runs. These limits leave controlled component comparisons, independent search repetitions, and broader transfer as the next tests of the framework.

Vestrum turns execution failures into scoped, reusable changes to agent software, providing a common procedure for improving verification, structure, and memory across different workloads.

## ETHICS STATEMENT

This work evaluates agents on existing benchmarks and spatial transcriptomics measurements. Terminal-Bench 4 Hard, LoCoMo (Maharana et al., 2024), AMA-Bench (Zhao et al., 2026), and UltraHorizon (Luo et al., 2025) are used under their released licenses. LoCoMo contains synthetic dialogues, and UltraHorizon uses simulated environments.

PathBench uses publicly available CosMx measurements from the HNSC slide in the AnchorR study (Morris et al., 2026), comprising six tissue sections. A bioinformatician's analysis pipeline supplies the expert reference annotation. Records used for evaluation are keyed by section and cell index and contain no patient identifier. The original data are available through the AnchorR preprint. An anonymized release of the code, harness configurations, and evaluation artifacts is provided as supplementary material. Each admitted mechanism is inspectable, and the appendix documents the alternatives considered during development.

## REPRODUCIBILITY STATEMENT

An anonymized release of the code, harness configurations, and loop records is provided as supplementary material. The appendices specify evaluation splits, judging and validation protocols, and detailed results. PathBench uses data publicly available through the AnchorR preprint (Morris et al., 2026); our release includes the annotation crosswalk, run outputs, and grader needed to reproduce the reported comparisons. Baseline implementation details are given in Appendix B.

## AI USE STATEMENT

Generative AI assisted the development of the conceptual framing, hypotheses, methods, and manuscript. AI agents proposed harness mechanisms from failure traces, provided feedback on experimental designs and screening criteria, and assisted with implementing harnesses, verifiers, benchmark adapters, and data-processing scripts. AI also supported failure categorization, interpretation of results, literature discovery and summarization, and drafting and editing the text and figures.

AI assistance with data cleaning and reformatting concerned processing code and benchmark adapters. We did not use generative AI to generate synthetic evaluation data for this study or to create the underlying spatial transcriptomics measurements or expert reference labels. AI was not used for translation. Formulating mathematical claims and writing proofs are not applicable, as the paper contains no theorems or proofs. The model-based grading used in the memory evaluations is described in Appendix I; PathBench uses expert annotations, and Terminal-Bench uses benchmark tests.

The authors directed the research, reviewed AI-assisted work, and retained responsibility for experimental design, data splits, admission criteria, and reported claims. Review included code testing, examination of failure analyses, and comparison of result interpretations with evaluation records. The authors take responsibility for the final content, including AI-assisted text, code, and analyses.

## 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 (ICLR)*, 2026.

Avrim Blum and Moritz Hardt. The Ladder: A Reliable Leaderboard for Machine Learning Competitions. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *PMLR*, pp. 1006–1014, 2015.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E.

Gonzalez, and Ion Stoica. Why Do Multi-Agent LLM Systems Fail? In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025.

Mert Cemri, Andrei Cojocaru, Melissa Pan, Shu Liu, Shubham Agarwal, Alexander Krentsel, Jay Tang, Kannan Ramchandran, Joseph E. Gonzalez, Matei Zaharia, Alex Dimakis, and Ion Stoica. Fantastic Adaptive Taxonomies and How to Use Them. *arXiv preprint arXiv:2607.16387*, 2026.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From Failed Trajectories to Reliable LLM Agents: Diagnosing and Repairing Harness Flaws. *arXiv preprint arXiv:2606.06324*, 2026a.

Mingguang Chen, Licheng Wang, and Bo Qu. The Horizon Gap: Planning, Memory, Execution, Training, and Evaluation for Long-Horizon LLM Agents. *arXiv preprint arXiv:2608.06663*, 2026b.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. *arXiv preprint arXiv:2504.19413*, 2025.

Lin Du, Jie Zhou, Yuxuan Cai, Kai Chen, Qin Chen, Xin Li, Bo Zhang, Wei Li, and Liang He. AutoMem: A Text-Gradient Recursive Self-Improvement Framework for Automated Memory Architectures Search. *arXiv preprint arXiv:2608.14621*, 2026.

Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. Preserving Statistical Validity in Adaptive Data Analysis. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing (STOC)*, pp. 117–126. ACM, 2015a. doi: 10.1145/2746539.2746580.

Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. The reusable holdout: Preserving validity in adaptive data analysis. *Science*, 349(6248):636–638, 2015b. doi: 10.1126/science.aaa9375.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017):625–630, 2024. doi: 10.1038/s41586-024-07421-0.

Nelson Goodman. *Fact, Fiction, and Forecast*. Harvard University Press, Cambridge, MA, 1955.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *International Conference on Learning Representations (ICLR)*, 2025a.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating Memory in LLM Agents via Incremental Multi-Turn Interactions. *arXiv preprint arXiv:2507.05257*, 2025b.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large Language Models Cannot Self-Correct Reasoning Yet. In *International Conference on Learning Representations (ICLR)*, 2024.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When Can LLMs Actually Correct Their Own Mistakes? A Critical Survey of Self-Correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024. doi: 10.1162/tacl_a_00713.

Charles Kemp, Amy Perfors, and Joshua B. Tenenbaum. Learning overhypotheses with hierarchical Bayesian models. *Developmental Science*, 10(3):307–321, 2007.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-End Optimization of Model Harnesses. In *Conference on Language Modeling (COLM)*, 2026.

Yifei Li, Weidong Guo, Lingling Zhang, Rongman Xu, Muye Huang, Hui Liu, Lijiao Xu, Yu Xu, and Jun Liu. Locomo-Plus: Beyond-Factual Cognitive Memory Evaluation Framework for LLM Agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25085–25100, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step. In *International Conference on Learning Representations (ICLR)*, 2024.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic Harness Engineering: Observability-Driven Automatic Evolution of Coding-Agent Harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

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*, 2026a.

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*, 2026b.

Haotian Luo, Huaisong Zhang, Xuelin Zhang, Haoyu Wang, Zeyu Qin, Wenjie Lu, Guozheng Ma, Haiying He, Yingsha Xie, Qiyang Zhou, Zixuan Hu, Hongze Mi, Yibo Wang, Naiqiang Tan, Hong Chen, Yi R. Fung, Chun Yuan, and Li Shen. UltraHorizon: Benchmarking Agent Capabilities in Ultra Long-Horizon Scenarios. *arXiv preprint arXiv:2509.21766*, 2025.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating Very Long-Term Conversational Memory of LLM Agents. *arXiv preprint arXiv:2402.17753*, 2024.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, et al. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces. In *International Conference on Learning Representations (ICLR)*, 2026.

Ludovico Mitchener, Angela Yiu, Benjamin Chang, Mathieu Bourdenx, Tyler Nadolski, Arvis Sulovari, Eric C. Landsness, Daniel L. Barabasi, Siddharth Narayanan, Nicky Evans, Shriya Reddy, Martha Foiani, Aizad Kamal, Leah P. Shriver, Fang Cao, Asmamaw T. Wassie, Jon M. Laurent, Edwin Melville-Green, Mayk Caldas, Albert Bou, Kaleigh F. Roberts, Sladjana Zagorac, Timothy C. Orr, Miranda E. Orr, Kevin J. Zwezdaryk, Ali E. Ghareeb, Laurie McCoy, Bruna Gomes, Euan A. Ashley, Karen E. Duff, Tonio Buonassisi, Tom Rainforth, Randall J. Bateman, Michael Skarlinski, Samuel G. Rodriques, Michaela M. Hinks, and Andrew D. White. Kosmos: An AI Scientist for Autonomous Discovery. *arXiv preprint arXiv:2511.02824*, 2025.

C. A. Morris, W. C. Bastian, Y. Cui, Z. Kurago, and E. F. Douglass. AnchorR: A QuPath and R interface for collaborative exploration of spatial transcriptomics and histology. *bioRxiv*, 2026. doi: 10.64898/2026.08.05.742985. Preprint, posted 11 August 2026.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing Instructions and Demonstrations for Multi-Stage Language Model Programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 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.

Pouya Pezeshkpour and Estevam Hruschka. AutoPyVerifier: Learning Compact Executable Verifiers for Large Language Model Outputs. *arXiv preprint arXiv:2604.22937*, 2026.

Huy Nhat Phan, Tien N. Nguyen, Phong X. Nguyen, and Nghi D. Q. Bui. HyperAgent: Generalist Software Engineering Agents to Solve Coding Tasks at Scale. *arXiv preprint arXiv:2409.16299*, 2024.

Zhenting Qi, Susanna Maria Baby, Stefanie Anna Baby, Kan Yuan, Andrew Tomkins, Tu Vu, Da-Cheng Juan, and Cyrus Rashtchian. On the Generalization Gap in Self-Evolving Language Model Reasoning. In *International Conference on Machine Learning (ICML)*, 2026.

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.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized Evolution for Image Classifier Architecture Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4780–4789, 2019.

Corby Rosset, Pratyusha Sharma, Andrew Zhao, Miguel Gonzalez-Fernandez, and Ahmed Awadallah. The Art of Building Verifiers for Computer Use Agents. *arXiv preprint arXiv:2604.06240*, 2026.

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.

Zihao Tang, Xin Yu, Ziyu Xiao, Zengxuan Wen, Zelin Li, Jiaxi Zhou, Hualei Wang, Haohua Wang, Haizhen Huang, Weiwei Deng, Feng Sun, and Qi Zhang. Mnemis: Dual-Route Retrieval on Hierarchical Graphs for Long-Term LLM Memory. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving Math Word Problems with Process- and Outcome-Based Feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Jonas Wallat, Maria Heuss, Maarten de Rijke, and Avishek Anand. Correctness is not Faithfulness in RAG Attributions. *arXiv preprint arXiv:2412.18004*, 2024.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-$\alpha$: Learning Memory Construction via Reinforcement Learning. *arXiv preprint arXiv:2509.25911*, 2025.

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 (ICLR)*, 2025.

Yiming Xiong, Shengran Hu, and Jeff Clune. Learning to Continually Learn via Meta-learning Agentic Memory Designs. In *ICLR 2026 Workshop on MemAgents*, 2026.

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 (NeurIPS)*, 2025.

B. Y. Yan, Chaofan Li, Hongjin Qian, Shuqi Lu, and Zheng Liu. General Agentic Memory Via Deep Research. *arXiv preprint arXiv:2511.18423*, 2025a.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing Large Language Model Agents to Manage and Utilize Memories via Reinforcement Learning. *arXiv preprint arXiv:2508.19828*, 2025b.

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.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive Language Models. *arXiv preprint arXiv:2512.24601*, 2025a.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, Yue Liao, Heng Chang, Xiaobin Hu, Qibing Ren, Wangchunshu Zhou, Chuanrui Hu, Yafeng Deng, and Shuicheng Yan. JIT-Agent: Scaling Harness Intelligence via Just-in-Time Harness Evolution. *arXiv preprint arXiv:2608.25593*, 2026a.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. MemEvolve: Meta-Evolution of Agent Memory Systems. In *International Conference on Machine Learning (ICML)*, 2026b.

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*, 2026c.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents. In *International Conference on Learning Representations (ICLR)*, 2026d.

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 (ICLR)*, 2025b.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. In *International Conference on Learning Representations (ICLR)*, 2026e.

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 *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, pp. 76583–76599, 2025c.

Yukun Zhang, Kemu Xu, and Yishen Chen. How Do Agent Harnesses Create Value? Planning Information and Release Control in Stateful LLM Agents. *arXiv preprint arXiv:2609.20474*, 2026f.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM Agents Are Experiential Learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 2024.

Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, Yuandong Tian, and Jishen Zhao. AMA-Bench: Evaluating Long-Horizon Memory for Agentic Applications. *arXiv preprint arXiv:2602.22769*, 2026.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, Jianghao Lin, and Weinan Zhang. Externalization in LLM Agents: A Unified Review of Memory, Skills, Protocols and Harness Engineering. *arXiv preprint arXiv:2604.08224*, 2026a.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. *arXiv preprint arXiv:2205.10625*, 2022.

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*, 2026b.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where LLM Agents Fail and How They can Learn From Failures. *arXiv preprint arXiv:2509.25370*, 2025.

Qiming Zhu, Shunian Chen, Rui Yu, Zhehao Wu, and Benyou Wang. From Lossy to Verified: A Provenance-Aware Tiered Memory for Agents. *arXiv preprint arXiv:2602.17913*, 2026a.

Xiaochen Zhu, Caiqi Zhang, Yizhou Chi, Tom Stafford, Nigel Collier, and Andreas Vlachos. Demystifying Multi-Agent Debate: The Role of Confidence and Diversity. *arXiv preprint arXiv:2601.19921*, 2026b.

# A Related Work

## A.1 Agentic memory

Memory systems distribute work differently between ingestion and answering. MemGPT pages a context window against external storage (Packer et al., 2023); other systems prepare notes, graphs, and indexes before a query arrives (Xu et al., 2025; Rasmussen et al., 2025; Gutiérrez et al., 2024; Chhikara et al., 2025; Tang et al., 2026; Zhu et al., 2026a). General Agentic Memory (GAM) keeps ingestion light and organizes information per query (Yan et al., 2025a). A further line learns memory-writing policies through reinforcement learning (Yan et al., 2025b; Wang et al., 2025). A comparison of twelve architectures reports no single winner (Zhou et al., 2026b). Vestrum treats memory construction and access as adaptable parts of the harness, alongside verification and control flow.

## A.2 Agentic harnesses

A harness comprises the prompts, tools, memory operations, verifiers, and control flow surrounding a model (Zhou et al., 2026a). Hand-designed generalist harnesses assign roles and routes between components (Phan et al., 2024). Just-in-time harnesses instead determine what to read, write, and execute for each task; models have also been trained to construct such harnesses (Zhang et al., 2026a). Trace-driven revision provides another route. Meta-Harness uses raw execution traces to guide changes (Lee et al., 2026), while Agentic Harness Engineering connects localized failures to proposed changes and predictions of their effects (Lin et al., 2026). Vestrum belongs to this line, carrying admitted mechanisms and search lessons between epochs, with proposals organized by design dimension and restricted by task-kind scope.

**Recursive language models.** Recursive language models hold a corpus as a variable in a Python interpreter. The root model receives the query and invokes further model instances on slices selected by its code (Zhang et al., 2025a). Decomposition on the memory benchmarks builds on this substrate and its decompose-and-integrate structure (Zhou et al., 2022). Vestrum can revise how these subcalls are used, together with the information prepared and retrieved for them.

## A.3 Long-horizon agents

UltraHorizon evaluates discovery tasks extending beyond four hundred tool calls (Luo et al., 2025); autonomous scientific workflows present similar challenges over longer interactions (Mitchener et al., 2025). Records preserved during an episode support later reasoning, while the decision to commit an answer remains a distinct harness operation. Memory-improvement loops propose changes from failures (Liu et al., 2026b; Du et al., 2026; Zhang et al., 2026b;c), and HarnessFix evaluates scoped repairs against regression criteria (Chen et al., 2026a). Vestrum shares failure-driven proposals and two benchmarks with this line of work, organizing remedies across four dimensions and evaluating composed bundles. A survey of 1,547 long-horizon papers identifies the separation of

model and system capabilities as an open problem (Chen et al., 2026b). Failure attribution remains difficult even when execution traces are available (Cemri et al., 2025; Zhu et al., 2025).

### A.4 Search over agent architectures

Automated design of agentic systems searches program space and retains promising candidates (Hu et al., 2025a; Real et al., 2019); agents have also been extended to modify their own source (Zhang et al., 2026d). Within memory, ALMA searches mechanism code (Xiong et al., 2026), while Evolve-Mem tunes retrieval settings (Liu et al., 2026a). Related approaches optimize prompts and coding-agent harnesses (Agrawal et al., 2026; Zhang et al., 2026e; Lin et al., 2026; Yuksekgonul et al., 2024; Khattab et al., 2023; Opsahl-Ong et al., 2024). AFlow searches connections among operators that include review, revision, and testing (Zhang et al., 2025b). These methods differ in editable surfaces, search procedures, and feedback; trace reflection is already central to GEPA and related work. Vestrum connects localized failure predicates and task kinds to four design dimensions, persistent lessons, generalization screening, and bundle admission.

The information retained between proposals further distinguishes these approaches. GEPA maintains a candidate-by-training-instance score matrix, selects from an instance-level Pareto front, and reflects on trajectories (Agrawal et al., 2026). Reflexion and ExpeL retain lessons from trajectories for later attempts (Shinn et al., 2023; Zhao et al., 2024), as Vestrum's lessons file does. AdaMAST induces and refines a failure taxonomy, passing taxonomy-coded diagnoses to its search mutator (Cemri et al., 2026). AdaMAST and HarnessFix are close methodological neighbors (Appendix F); we do not report empirical comparisons against them. All these approaches depend on attributing failures to steps in a trajectory (Zhang et al., 2025c). Vestrum uses a language model for localization and separately evaluates the proposed remedy, so a plausible diagnosis alone cannot justify admission.

### A.5 Verification in agents

The literature examines executable verifiers for model outputs (Pezeshkpour & Hruschka, 2026), attribution to retrieved evidence (Wallat et al., 2024), and uncertainty estimated from a model's distribution (Farquhar et al., 2024) or agreement among models (Zhu et al., 2026b). Vestrum's retained checks use recorded evidence or executable constraints, making both the material consulted and the consequence of failure explicit. Work on computer-use agents likewise emphasizes verifier design (Rosset et al., 2026). In Vestrum, placement, evidence, and scope are harness choices that can be revised during development. The LoCoMo checker applied to every answer, for example, improved two answers but regressed six (Section 4.1).

**Placement and permitted action.** Process supervision evaluates intermediate reasoning steps rather than only final outcomes (Uesato et al., 2022; Lightman et al., 2024). Our development record examines a related distinction between stages of an agent harness: retained checks occur at memory writes, evidence merges, rule commitments, and code edits. AutoPyVerifier searches compact executable verifiers for model outputs (Pezeshkpour & Hruschka, 2026); Vestrum treats checks within the harness as part of the editable workflow.

A concurrent study on $\tau^2$-bench reports that a read-only terminal verifier can block most false passes at under a cent per episode (Zhang et al., 2026f). Such a check withholds release while leaving accepted answers unchanged. This distinction helps interpret our mixed record: broad answer rewriting sometimes damaged correct answers, a selective answer checker was net positive, and some intermediate gates impeded progress. The record motivates considering permitted actions and evidence alongside placement, consistent with the difficulty of self-correction without external feedback (Huang et al., 2024; Kamoi et al., 2024). The record shows intermediate checks to be cheaper rather than universally more effective: gates whose evidence could not separate rival rules impeded progress.

### A.6 Evaluation validity

Repeatedly scoring candidates on the same data creates an adaptive-analysis problem. Reusable-holdout methods control information release to limit leakage (Dwork et al., 2015b;a; Blum & Hardt,

2015); closed-loop self-improvement can also plateau as the development–test gap widens (Qi et al., 2026). Concerns about LoCoMo's token-F1 metric (Li et al., 2026) motivate blinded multi-pass judging and evaluation across other memory workloads (Zhao et al., 2026; Wu et al., 2025; Hu et al., 2025b). Vestrum distinguishes development data available to search, optional validation data, and test data evaluated after freezing. Table 7 reports the data splits. UltraHorizon's fresh validation seeds limit repeated feedback from the same examples, but we make no formal reusable-holdout guarantee.

## B Evaluation Settings

Table 7 summarizes the five settings, their data splits, and the units graded by each benchmark.

Table 7: Evaluation settings. A check is the unit the benchmark grades. Development tasks are the ones the loop ran and read; validation tasks are disjoint from both development and test, and test tasks were run only after the harness was frozen. At test, GAM runs on Gemini 3.7 Flash, and Claude Code on Sonnet 5 (Terminal-Bench) or Opus 5 (PathBench).

| setting | check | baseline | development | validation | test |
|---|---|---|---|---|---|
| Terminal-Bench | test | Claude Code | 7 tasks | separate tasks | 8 tasks $\times$ 2 draws |
| PathBench | cell label | Claude Code | 2 sections | – | 4 sections $\times$ 2 episodes |
| UltraHorizon | episode score | GAM | 3 episodes | 21 episodes | 42 episodes $\times$ 3 seeds |
| LoCoMo | judged probe | GAM | 1 conversation | 1 conversation | 956 probes $\times$ 3 |
| AMA-Bench | judged probe | GAM | 10 episodes | – | 240 probes |

Table 8 specifies the editable surface, task kinds, failure evidence, and admission criteria in each setting. Appendix D gives the reported Terminal-Bench procedure and its principal prompts.

Table 8: How the loop was run in each setting.

| | UltraHorizon | LoCoMo | AMA-Bench | PathBench | Terminal-Bench Hard 4 |
|---|---|---|---|---|---|
| parts the loop may change | mechanism settings in the four dimensions | question-answering path and synthesis prompts; the store is unchanged | per-chunk extraction, notebook builds, deterministic views, synthesis prompts | what the agent reads before it plans | prompts, instruction files and mechanisms gated by task kind |
| task kind | environment | question category | probe axis | one kind, annotation | task mode $\times$ check style, tagged from the 66 instructions alone |
| failure classes from | graded development episodes | judged failures on the development conversation | judged probes on 10 development episodes | per-label disagreement with the reference on the development sections | failures on the 7 development tasks |
| development criterion | each mechanism's predicted behaviour appears on three development episodes, with no collapse | blinded judgement of paired runs on the development conversation (152 probes) | paired per-episode differences on judged batches | paired by seed against the current harness | gains checks the current harness fails in every draw; no task of another kind loses a check; cost under $1.5\times$ the baseline |
| generalization check | reviewer; leakage lint | reviewer; leakage lint | reviewer; leakage lint | reviewer; leakage lint | reviewer; leakage lint |
| validation set | 21 episodes on new seeds, each used once | one validation conversation (152 probes), opened during development | none | none | validation tasks, disjoint from the test set |
| epochs | four epochs | three epochs | four epochs | one epoch, one iteration | reported epoch, $123.5 |

**Baseline implementations.** The vendored GAM implementation (Yan et al., 2025a) is unchanged apart from the described adapter code. A-Mem (Xu et al., 2025) uses Gemini embeddings, the shared answer prompt, and repaired ingestion. Claude Code uses Sonnet 5 on Terminal-Bench and Opus 5 on PathBench.

## C TERMINAL-BENCH ABLATION

Table 3 ablates the loop of Section 3.2 by removing its components. The second row removes only the lessons file. The remaining rows remove, cumulatively, the ceiling probe; failure classes together with generalization screening and task-kind scope; turn-level localization; and trace access, leaving a loop that reads only final outputs. Each variant starts from Claude Code and proposes its own mechanisms. Pilot experiments motivated the four design dimensions because open-ended search rarely produced useful early changes; their individual contributions were not separately ablated. For rows reporting check counts, a gain is a check that Claude Code fails in every draw and the evaluated harness passes in every draw; a loss is the reverse. The variant without localization instead reports percentage-point improvement in mean check share from single draws. Cost is per-task spend relative to Claude Code, and validation counts checks gained on validation tasks, which are disjoint from the test tasks of Section 3.2; only the full loop's frozen harness was run on the test set. A validation entry of 0 includes variants that admitted no candidate to validate.

**Memory for the search.** Removing the lessons file while retaining all other loop components yields 10 development checks gained with no losses and 6 validation checks gained, at $1.6\times$ the baseline's cost. This candidate was never evaluated on the test set. The full loop gains 16 development checks against one loss and 11 validation checks, at $1.32\times$ cost. With the lessons file, the loop therefore records larger development and validation gains at lower cost, with one additional development check lost. The lessons file carries experience from rejected proposal lines into subsequent iterations (Section 2.6).

**The ceiling probe.** Removing the ceiling probe of Appendix D, which identifies development failures responsive to a supplied procedure, lowers the development gain from 16 checks to 3 and the validation gain from 11 checks to 5. With the probe, 9 of the 12 checks the admitted set predicted would pass did pass.

**The generalization check and scope by kind.** Without failure classes, generalization screening, and task-kind scope, the loop showed no validation gain. The first mechanism this variant tried lost two checks that Claude Code always passed on a task of a different kind, whose score fell from 7 to 5 of 10 checks, and three checks it gained on one development task in one draw did not reproduce in the next. With screening and scope, the loop still encountered a regression from an unscoped mechanism, but admission refused that proposal before retaining its scoped version (Section 2.5); this was the first configuration with a validation gain, at below the baseline's cost.

**Recognizable failure classes.** Without predicate failure classes, the variant used three fixed classes, but every failure fell into the same class. The labels therefore offered little discrimination between remedies, and a later iteration found a gain attributed to the wrong change. In the full loop, failure classes are predicates recognizable in traces. They specify behavior for the proposer to target and for subsequent trace analysis to inspect.

**Localization and trace access.** Without turn-level localization, the loop routed traces into failure clusters, leaving 55% of failures unassigned; when its second round produced no diagnosis, proposals bypassed the router. It raised its mean share of checks passed from 0.734 to 0.827 at 2.3 times the baseline's cost, with no validation gain. Without trace access, the loop read only final outputs. Its second round changed harness code and role prompts together, but the run report could not determine which change had acted; a repair round lowered a task of another kind from 17 to 14 of 20 checks, and nothing was admitted.

## D THE VESTRUM LOOP ON TERMINAL-BENCH 4 HARD: STEPS AND MAIN PROMPTS

This appendix describes the reported Terminal-Bench epoch, following Algorithm 1, and quotes the principal prompts, shortened where marked. Loop calls used Claude Fable 5.1 and Claude Opus 5; task executions used Sonnet 5, the same model as the Claude Code baseline. The quoted prompts retain their original terminology. The reported run is the loop's first epoch; the epoch, including its

test runs, cost $123.5. Development used seven tasks, validation used separate tasks, and testing used eight further tasks.

**Select consistently failed checks.** Claude Code runs each development task three times. A check contributes failure evidence only if it fails in all three draws. Intermittent failures, called coin checks in the original prompts, are excluded from this count but remain available for diagnosis.

**Localize and classify.** One call per failed task reads its Claude Code traces, writes the procedure required to pass every check, and identifies the missing step for each failure:

> "## Solved trace — the ordered checklist an agent would have to execute to pass EVERY check, as observable steps … Never quote test code or expected values … ## Gap — for each stable-red AND coin check: which solved-trace steps the bare draws lacked, as CAPABILITIES (KNOW · TEST · CORRECT · ORDER …)."

**Ceiling probe (Terminal-Bench only).** Before any mechanism was proposed, each development task was run once with its solved procedure given to the agent. On five of the seven tasks this run passed checks that Claude Code never passes, identifying failures responsive to procedural assistance. On the other two it did not, and proposals targeted only the five responsive tasks. This probe guides the search toward promising interventions; it does not establish an exhaustive boundary on the model's capability.

**Compose and screen.** One composer call proposes, selects, and composes the bundle. Generalization screening therefore follows selection but precedes candidate execution. The composer turns missing steps into mechanisms scoped to the task kind:

> "Your job is to make the harness DELIVER the DELIVERY steps to an agent that has only the request, on every task of the same shape, without naming the task. Start from the parent: keep every parent component that ever passed a stable-red check … Then add 2–4 new components."

Each mechanism then passes the generalization check: a reviewer answers seven pass/fail questions per mechanism, and a lint refuses any string or constant unique to one task.

> "Q1 fails if the decision is a fixed vocabulary/regex list … Q4 fails if anything task-derived is baked in (check the code, not the claim)."

**Evaluate, prune, and reevaluate.** The bundle is evaluated against the current harness on paired development tasks, and a trace reader labels each mechanism's behavior:

> "'not_fired' … 'fired_not_acted' (delivered or ran, but the agent's actions at the targeted point do not differ from a bare run), 'fired_acted' (the agent visibly did what the component asked …)"

Mechanisms judged inactive are disabled and those judged harmful are dropped before the pruned bundle is reevaluated:

> "a label is WORKED only if the stable-red check passed AND the read shows the component's own artifact/turn caused the behaviour … FIX if NOT-FIRED … DROP if HARMFUL … add new components ONLY along the lines of what worked."

In the reported epoch, the full bundle gained 16 checks that Claude Code consistently failed and lost one it consistently passed; 9 of the 12 checks predicted by its proposals passed. The pruned bundle gained 11 and lost 4, so only the full bundle advanced to validation. Development runs cost 1.32 times the baseline.

**Admission criteria.** A bundle is admitted when it passes checks that Claude Code fails in every draw, no check the current harness passed is lost on a task of another kind, and the per-task cost is under $1.5\times$ the Claude Code mean (Section 2.5); of the full and pruned bundles, development results determine which candidates advance to validation. A pruned bundle is validated only if it outperforms the full bundle on development; final admission follows validation.

**Validation and lessons.** Before the harness is admitted and frozen, it runs on the validation tasks, which are disjoint from the test set, as a check on the development result; the reported harness gained 11 checks over Claude Code there (Table 3). The epoch closes by recording successes and failures for subsequent search. A line of proposals that fails twice writes a lesson to a file that every later composition must read.

**Test.** The frozen admitted harness is evaluated on the eight test tasks, with two draws per system and no return of test outcomes to search.

On test, Vestrum passes 105.5 of 150 checks on average across draws, against Claude Code's 95.5. Four task means improve, four are unchanged, and none declines. Test cost is $1.03\times$ Claude Code's under the same per-task spend cap.

# E Overhypothesis, Categorization, and Generalization Screening

**The organizing expectation.** An overhypothesis guides what can be inferred from an individual example by expressing an expectation about a wider class (Goodman, 1955; Kemp et al., 2007). Vestrum uses the expectation that tasks of a shared task kind may exhibit recurring failure classes, whose remedies may transfer within that kind. This connects three operations: task categorization proposes a scope of transfer, failure categorization identifies behavior that may recur, and the generalization check examines whether a mechanism states a remedy at that level. The framework draws inspiration from overhypothesis theory without fitting a hierarchical Bayesian model or deriving a transfer guarantee.

A mechanism targets a failure class within the task kinds where it was observed. Its condition and content must remain meaningful beyond the originating example. Screening assesses this formulation; it does not establish that every task of the kind exhibits the failure, or that the mechanism improves any unseen task. Empirical admission separately determines whether the evaluated bundle is retained (Section 2.5).

**Kinds on Terminal-Bench 4 Hard.** A separate pass, with no access to solutions or tests, reads each of the 66 instructions once and records its mode (repair, build, configure or analyze), whether an oracle or a specification file is given, the languages involved, and the classes of knowledge the instruction presupposes. The kind of a task is its mode crossed with the style of its checks, where the check style is read from whether the instruction names a specification file, an oracle or measured values. Of a test task, a proposal sees only the kind its instruction was sorted into, as a count per kind; never its solution, test, trace or outcome. The Terminal-Bench evaluation therefore concerns unseen tasks of known task kinds. The categorization uses the public instruction list, including test instructions, while test solutions, traces, checks, and outcomes remain unavailable to proposals.

**Procedural knowledge.** Some failures reflect missing knowledge, such as a file-format convention, a finite-sample correction, or an estimator's precondition. On Terminal-Bench, the procedure that obtains such a fact may enter the harness, keyed on terms in the instruction; an answer fact taken from the development task may not. A knowledge class found in no other task is flagged, and a mechanism built on it is treated as fitted to an individual task.

## E.1 Permissible Supervision by Setting

The boundary between adapting a harness to a workload and compiling an answer into it is drawn per setting. In all five settings a procedure may be written, keyed on terms that appear in the request, and no task-specific name, symbol, path or constant taken from an instruction or a test may appear in a mechanism. The settings differ in what else is permitted.

- **UltraHorizon, AMA-Bench.** Nothing drawn from a development episode's board, rules or graded answer.
- **LoCoMo.** Two answer rules written from development failures. Nothing drawn from a development probe's answer.

- **PathBench.** Rules, reference tables, and a helper written from the failures and reference labels of the two development sections. Transfer is to held-out sections of the same slide. Nothing is taken from the four test sections.
- **Terminal-Bench.** Nothing drawn from the solution, test, trace or outcome of a test task, and of its instruction only its kind.

The leakage lint searches a mechanism's text for task-specific strings. A language-model reviewer also reads each mechanism to assess whether its content and conditions extend beyond the source example. Neither screen establishes semantic separation from the development tasks. The explicit supervision boundary and subsequent evaluation are therefore necessary parts of the protocol.

## F Nearest Neighbours: AdaMAST and HarnessFix

Table 9 compares Vestrum with two closely related loops. All three retain structured failure information between proposals. Vestrum connects failure classes to task-kind scope, evaluates composed bundles, and screens the generality of their mechanisms. The table describes methodological choices, not an empirical ranking.

Table 9: Vestrum beside AdaMAST (Cemri et al., 2026) and HarnessFix (Chen et al., 2026a), as we read each paper. A dash means we found no statement either way.

| | AdaMAST | HarnessFix | Vestrum |
|---|---|---|---|
| failure record | taxonomy of failure codes on three axes, induced from traces and refined during the search | flaw records consolidated across failed runs, kept in a repair memory | failure classes, task kinds, and lessons |
| task kinds | codes carry a domain-specific axis; tasks are not partitioned | flaws are grouped by harness layer, not by task | tasks partitioned into kinds; each mechanism targets a failure class within a task kind |
| frequency of a failure | how often each code occurs is reported | recurrence across runs required before repair | coverage per class ranks what is proposed |
| proposal | the mutator receives a taxonomy-coded diagnosis | a flaw is mapped to scoped repair operators | proposals under four dimensions, scored by a language model |
| mechanisms per evaluation | – | one repair per validation | several per run, each on its own kind |
| acceptance | evaluator score in the search | fixes the target flaw without unacceptable regressions; train, validation and test tasks | screened bundle meets the admission criteria; validation where used; test after freezing |
| contamination screen | held-out traces for taxonomy agreement | – | language-model reviewer and leakage lint |

## G Validation Protocol on UltraHorizon

An UltraHorizon draw is a complete long-horizon episode. Development episodes screen proposals, while validation decides admission. Across the three development episodes, a mechanism passes the screen when its predicted behavior appears, such as a commit gate withholding a commitment, and no score collapses relative to the current harness.

Validation samples new seeds outside every manifest and runs both the candidate and the current harness on each. Each seed is used once; only the decision and mean score return to the loop, with no transcript or per-episode score. Across four epochs and 21 validation seeds, the loop considered 59 proposals and admitted two bundles. Together they contain the UltraHorizon mechanisms in Table 5.

# H MEMORY BENCHMARK SETUPS

The benchmarks exercise different uses of retained information. UltraHorizon requires discovering environment rules through long interactions (Luo et al., 2025). Development uses three episodes; each of three test seeds uses 42 episodes across three environments. LoCoMo asks questions about extended conversations (Maharana et al., 2024). We evaluate multi-hop, temporal, open-domain, and single-hop questions using strict accuracy, requiring all key facts without contradiction; lexical F1 is reported as a secondary measure given its known limitations (Li et al., 2026). Development uses one conversation of 152 probes and validation a second; testing uses six others, containing 956 probes, in three replicates with fresh stores and judging runs. The adversarial, unanswerable category is excluded. AMA-Bench tests recall, causality, state updating, and state abstraction after tool-using trajectories (Zhao et al., 2026); ten development episodes precede twenty held-out episodes containing 240 probes.

# I JUDGE PROTOCOL

The memory benchmarks use blinded Sonnet judgments against a fixed rubric, with three independent passes and majority vote. The judge compares each answer with the reference key facts without learning which system produced it. Judge accuracy, called strict accuracy in the detailed tables, counts answers with all required facts present and no contradiction; lenient accuracy also counts partial answers. LoCoMo pass agreement was 0.955, 0.947, and 0.948 across its three runs. AMA-Bench recorded 663 unanimous judgments out of 720.

UltraHorizon uses the benchmark rubric for each environment and one canonical judge prompt. Every environment is graded on a 0–100 scale, and both systems are judged in one 42-episode batch per seed. Because the same rule library recurs across episodes, the statistical units are the rule configurations, not the 126 episode executions, so the main text reports paired directions and seed means without an episode-level $p$-value.

# J FULL RESULT TABLES

## J.1 ULTRAHORIZON: FULL SEED RECORD

Each seed contains the same 42 test episodes per system. Table 10 records the mean of every seed.

Table 10: UltraHorizon mean score per 100 over the 42 test episodes of each seed.

| Seed | GAM | Vestrum | Δ |
|---|---|---|---|
| 1 | 47.86 | **54.88** | +7.02 |
| 2 | 48.10 | **60.12** | +12.02 |
| 3 | 46.79 | **64.29** | +17.50 |
| pooled | 47.58 | **59.76** | +12.18 |

## J.2 LOCOMO: FULL SEED RECORD

The statistical unit on LoCoMo is the conversation. Probe-level $p$ values are optimistic because questions within a conversation share retrieved evidence; the main text therefore reports the direction of each replicate. Per run, GAM used 4,427/4,479/4,468 calls at \$15.39/\$15.51/\$15.55; Vestrum used 4,104/4,192/4,130 at \$21.20/\$21.73/\$21.34; A-Mem used 12,029/12,040/12,025 at \$17.63/\$17.49/\$17.46. On development, the planner split 12 of 152 questions and 11 of 152 on the validation conversation.

## J.3 AMA-BENCH: FULL THREE-WAY AND REPLICATION RECORD

The frozen harness was evaluated twice to measure generation and judging variability. Table 12 gives the second run used for the three-way comparison, also reported in the upper block of Table 1.

Table 11: LoCoMo strict and lenient accuracy by full run, 956 test probes per system. Net is Vestrum's probe wins minus losses against GAM; $p$ is an exact sign test over probes. Agreement is the fraction of grades unanimous across the three judge passes, computed over the whole replicate and therefore the same for every system in it.

| Replicate | System | Strict | Lenient | Lexical F1 | Agreement | Net | $p$ |
|---|---|---|---|---|---|---|---|
| 1 | Vestrum | 0.741 | 0.881 | 0.667 | 0.955 | $+24$ | 0.038 |
| 1 | GAM | 0.715 | 0.859 | 0.618 | 0.955 | — | — |
| 1 | A-Mem | 0.497 | 0.647 | 0.477 | 0.955 | — | — |
| 2 | Vestrum | 0.749 | 0.897 | 0.679 | 0.947 | $+39$ | $5.6\times10^{-4}$ |
| 2 | GAM | 0.708 | 0.865 | 0.612 | 0.947 | — | — |
| 2 | A-Mem | 0.497 | 0.651 | 0.480 | 0.947 | — | — |
| 3 | Vestrum | 0.756 | 0.901 | 0.679 | 0.948 | $+34$ | $2.9\times10^{-3}$ |
| 3 | GAM | 0.721 | 0.865 | 0.623 | 0.948 | — | — |
| 3 | A-Mem | 0.486 | 0.632 | 0.473 | 0.948 | — | — |

Table 12: AMA-Bench held-out three-way result, 240 shared probes under one blinded three-pass judgment. Vestrum and GAM use their second answer-generation run; A-Mem has one generation. Per-axis counts are StateAbstraction 40, StateUpdating 67, Causal 58 and Recall 75.

| System | Overall | StateAbs. | StateUpd. | Causal | Recall |
|---|---|---|---|---|---|
| Vestrum | **0.742** | **0.78** | **0.78** | **0.76** | **0.68** |
| GAM (Yan et al., 2025a) | 0.642 | 0.57 | 0.75 | 0.67 | 0.56 |
| A-Mem (Xu et al., 2025) | 0.562 | 0.45 | 0.57 | 0.60 | 0.59 |

Table 13 preserves both paired run comparisons. The final three-way run gives Vestrum against A-Mem $+0.179$ strict, 13/4/3 episode wins, losses and ties, $p = 0.0490$ two-sided.

### J.4 Model usage

Table 14 records model usage for the indicated evaluation runs. UltraHorizon seeds 2 and 3 used 5,053 and 4,872 model calls and cost $49.30 and $45.90. The AMA-Bench Vestrum runs used 713 and 724 calls. LoCoMo used 61,894 calls and $163.30 across all systems and seeds.

## K PathBench Grading

PathBench asks an agent to recover tissue sections from a spatial transcriptomics export, filter cells, cluster their expression profiles, and assign names from sixteen cell types. It supplies seven local sources and no network. The reference is the laboratory annotation produced by a bioinformatician's pipeline for the AnchorR data (Morris et al., 2026). Development uses two sections of a head-and-neck squamous cell carcinoma slide, with 55,366 reference cells; testing uses the other four, with about 129,000 cells scored. No test-section label or measurement was read before the harness was frozen.

Agent labels are joined to the laboratory reference by cell identifier, matching 99.0 to 99.9 percent of cells in the agent's output. The four test sections contain 136,338 reference cells; about 129,000 remain after the agent's quality filter and are scored. Confusion matrices are computed at three hierarchy levels, containing four, eleven, and eighteen labels. The task brief requests sixteen names, while the finest scoring level contains eighteen labels, seventeen of which occur in the reference.

A hand-audited crosswalk maps finer agent labels into the reference vocabulary without splitting coarse labels into finer ones. A label unresolved at a scoring level is counted as wrong, so declining to subtype earns no credit through abstention. The endpoint is agreement with an expert-in-the-loop laboratory annotation (Morris et al., 2026), rather than independently established ground truth. The underlying data are publicly available through the AnchorR preprint; the accompanying code release includes the annotation crosswalk, run outputs, and grader.

Table 13: AMA-Bench Vestrum–GAM replication. Each row is a within-run comparison on the same 20 episodes and 240 probes. Every $p$ is a two-sided exact sign test over episodes.

| Run | Vestrum | GAM | $\Delta$ | Episode W/L/T | sign-test $p$ |
|---|---|---|---|---|---|
| 1 | 0.787 | 0.667 | $+0.121$ | 12/3/5 | 0.0352 |
| 2 | 0.742 | 0.642 | $+0.100$ | 11/4/5 | 0.118 |

Table 14: Recorded model usage. A dash means cached input was not recorded separately.

| Benchmark/run | Model | Input | Cached input | Output | Total |
|---|---|---|---|---|---|
| UltraHorizon seed 2 | Gemini 3.7 Flash | 49,839,008 | 3,817,446 | 3,103,475 | 56,759,929 |
| UltraHorizon seed 3 | Gemini 3.7 Flash | 45,952,793 | 3,523,674 | 2,979,775 | 52,456,242 |
| AMA Vestrum run 1 | Gemini 3.7 Flash | 3,198,075 | — | 286,253 | 3,484,328 |
| AMA Vestrum run 2 | Gemini 3.7 Flash | 3,224,191 | — | 293,294 | 3,517,485 |

ARI and NMI compare clustering independently of label names; accuracy and macro-F1 compare cell-type assignments. We additionally count section–class pairs with F1 $\geq 0.5$: on development, Claude Code and Vestrum average 18.0 and 27.5 of 30 pairs; on test, they average 43.5 and 51.5 of 60 pairs.

## L GEPA Comparison and Resource Accounting

**Search configuration.** GEPA (Agrawal et al., 2026), version 0.1.4 with stock reflective evolution, optimized GAM's question-time prompts on LoCoMo and AMA-Bench and its action protocol on UltraHorizon, with memory construction fixed. Its task model was Gemini 3.7 Flash, as for every system at test, and its reflection model was Claude Opus 5. Vestrum searched using the earlier task models specified in Section 3. Evaluation counts therefore compare numbers of task-agent executions, not executions of the same model. GEPA received Vestrum's development data and used its own Pareto set for selection. On UltraHorizon, reflection received the judge's per-rule verdicts but never the ground-truth rules, and every candidate passed Vestrum's leakage lint. An earlier run whose candidate embedded the answer key in its prompt was discarded.

**Test and judging repetitions.** The frozen GEPA program was run on three test seeds. On LoCoMo, each seed was judged blind together with GAM, Vestrum, and A-Mem. AMA-Bench used three judging runs: the third paired GEPA's third generation with GAM's and Vestrum's first-generation answers. UltraHorizon used the canonical judge for each seed. Under Appendix I, each answer is graded against the reference answer's key facts; judge accuracy is the share graded fully correct.

These evaluations form a separate comparison from the upper block of Table 1. That block reports UltraHorizon means over three seeds (126 paired executions), LoCoMo mean and standard deviation over three replicates, and AMA-Bench's second answer-generation run (240 probes), with one A-Mem generation using Gemini embeddings. The lower block reports the separately judged GEPA comparison. Scores and differences should be interpreted within each block.

**Performance.** Vestrum scores above GEPA in every reported paired evaluation (Table 15). On AMA-Bench, GEPA improves over GAM by a pooled net $+37$ questions. Vestrum's further margin is concentrated in state abstraction, averaging 81% against 57% across the three judging runs. On UltraHorizon, GEPA completed four iterations without improving GAM's prompt on its Pareto set and stopped at a plateau. Its test results therefore re-execute GAM's program; the 44 wins, 38 losses, and 44 ties against GAM across 126 episodes describe execution variability.

**Learning tokens and stopping conditions.** Table 16 distinguishes budget caps from actual usage. On LoCoMo, GEPA's cap was 78.2M tokens, but it stopped at a plateau after 22.8M. On AMA-Bench its cap was raised to 25M, and it used 15.7M before reaching its time limit, compared with Vestrum's 9.5M. On UltraHorizon it also stopped at a plateau. Token counts sum usage over all

Table 15: GEPA against GAM and Vestrum over three test seeds (AMA-Bench: three judging runs). Judge accuracy on LoCoMo and AMA-Bench, score per 100 on UltraHorizon. Net counts questions or episodes won minus lost, pooled over seeds, ties excluded; $p$ is a two-sided exact sign test over questions, optimistic because questions in one conversation or episode share evidence.

| | GAM | GEPA | Vestrum | Vestrum − GEPA |
|---|---|---|---|---|
| LoCoMo, seeds 1/2/3 | 72.9 / 71.7 / 72.9 | 74.0 / 73.2 / 74.7 | **75.5 / 76.6 / 76.7** | net +15 / +32 / +19 |
| mean ± s.d. | $72.5 \pm 0.7$ | $74.0 \pm 0.8$ | $\mathbf{76.3 \pm 0.7}$ | pooled net +66, $p = 5.7 \times 10^{-4}$ |
| AMA-Bench, runs 1/2/3 | 65.0 / 66.7 / 65.4 | 72.5 / 70.4 / 69.6 | **77.1 / 75.4 / 76.7** | net +11 / +12 / +17 |
| mean ± s.d. | $65.7 \pm 0.9$ | $70.8 \pm 1.5$ | $\mathbf{76.4 \pm 0.9}$ | pooled net +40, $p = 6.3 \times 10^{-4}$ |
| UltraHorizon, seeds 1/2/3 | 47.9 / 48.1 / 46.8 | 47.1 / 46.4 / 50.1 | **54.9 / 60.1 / 64.3** | +7.8 / +13.7 / +14.2 |
| mean ± s.d. | $47.6 \pm 0.7$ | $47.9 \pm 2.0$ | $\mathbf{59.8 \pm 4.7}$ | 68 / 26 / 32 episodes (W/L/T) |

models called by each loop; they do not imply equal computational or monetary cost. Vestrum uses more learning tokens on LoCoMo and UltraHorizon and fewer on AMA-Bench.

Table 16: Learning budget. Tokens are summed over every model each loop called.

| | Vestrum tokens | GEPA cap | GEPA used | GEPA reflection calls | GEPA stopped at |
|---|---|---|---|---|---|
| LoCoMo | ∼30M | 78.2M | 22.8M | 10 | plateau |
| AMA-Bench | 9.5M | 25M | 15.7M | 8 | time limit |
| UltraHorizon | ∼74M | – | 21.9M | 4 | plateau |

**Estimated task-agent evaluations.** Task-agent evaluations measure a different resource: executions of the program being improved. GEPA evaluates the current and proposed candidates on batches of 32 development probes, or six UltraHorizon episodes, and reevaluates improving candidates on its full validation set. Vestrum evaluates composed bundles, reuses prior current-harness runs where available, and reads existing traces. Its totals use an allowance of 1.5 passes per epoch for pruning and retesting. This is an accounting convention, not an exact execution log or a requirement to validate both bundles. A pruned bundle reaches validation only if it outperforms the full bundle on development. Table 17 gives the calculation.

Table 17: Estimated task-agent evaluations during learning under the stated accounting convention: probe answers on LoCoMo and AMA-Bench, episodes on UltraHorizon. Vestrum's totals use the 1.5× allowance for pruning and retesting.

| | GEPA | Vestrum | Vestrum / GEPA |
|---|---|---|---|
| LoCoMo | 64 iterations × 64, plus 9 validation passes × 81: **4,825** | 3 epochs × 1.5 × (152 development + 152 validation): **1,368** | 0.28× |
| AMA-Bench | 42 iterations × 64, plus 5 validation passes × 48: **2,928** | 4 epochs × 1.5 × 120 development: **720** | 0.25× |
| UltraHorizon | 54 batch episodes, plus 6 validation: **60** | 4 epochs × 1.5 × 3 development, plus 21 validation seeds × 2: **60** | 1.00× |

**Interpretation and uncertainty.** Under this accounting, Vestrum uses approximately a quarter of GEPA's task-agent evaluations on LoCoMo and AMA-Bench and the same number on UltraHorizon, while achieving higher scores in the evaluated configurations. The comparison describes a tradeoff between task execution and other learning work, rather than an advantage in total compute. Search spaces, task models, actual token use, and stopping conditions differ.

The pooled question-level sign tests favor Vestrum on LoCoMo and AMA-Bench ($p = 5.7 \times 10^{-4}$ and $6.3 \times 10^{-4}$), but questions within a conversation or episode share evidence, making these val-

ues optimistic. AMA-Bench's third judging run also reuses GAM and Vestrum answers. Each setting evaluates one selected program per method; repetitions quantify test execution and judging variability, not variation across independent optimization runs. The comparison does not isolate the contribution of structural changes or establish superiority over methods with the same editable surface.

## M ADMITTED MECHANISMS IN FULL

Table 18 lists the admitted mechanisms by setting. Table 5 in Section 4 groups them by design dimension, and Appendix N records verification alternatives. Admission applies to the evaluated bundles and does not establish the contribution of each mechanism.

**From development failures to mechanisms.** The following summaries connect each development diagnosis to the corresponding admitted bundle.

**Terminal-Bench 4 Hard.** Claude Code changed shipped code without recording its invariants. The edit gate requires an invariant and a probe script before the first change. Two repair procedures specify what to record; a scale decision records the chosen branch before execution.

**UltraHorizon.** GAM scored zero on the medium and hard development boards and committed answers without replaying their supporting observations. The learned harness organizes observations into evidence rows and a notebook for each rule class. A separate model call re-analyzes contradictory evidence, probe plans guide investigation, and a commit gate requires replay before accepting a rule where the gate is enabled (Appendix N).

**LoCoMo.** A planner decides whether to split the question; split questions receive one further retrieval round when evidence is insufficient, then a claim judge at answer merge. Two answer rules place the strongest evidence first and require a best guess rather than refusal.

**AMA-Bench.** Development failures implicated notebook entries used by later probes. The admitted bundle preserves notebook history, checks entries during construction, and provides deterministic views including an event index, an action record, and object timelines.

**PathBench.** Claude Code assigned thousands of cells to Unknown despite markers indicating their lineage. Eleven sourced rules, beginning with naming each cluster by its lineage, marker and expression reference tables built from the development sections, and a nearest-neighbour helper for small unlabelled clusters supply guidance before the agent plans.

A verifier was admitted in four of the five settings; on PathBench the admitted change is knowledge synthesis.

**Additional development observations.** LoCoMo retained GAM's store and retrieval channels; two further verification variants tied. On AMA-Bench, removing synthesis cost 10 points in an earlier development comparison; alternatives using a memory split, boundary decider, merge step, and ontology were not retained. PathBench's bundle was admitted in one epoch with one iteration.

## N THE VERIFICATION RECORD

This appendix supports the verification analysis in Section 4.1. It records each retained verifier and the alternatives considered in preliminary experiments, including the Terminal-Bench ablation variants of Table 3 and earlier harness versions. Those experiments used GPT-4o-mini on AMA-Bench, Claude Haiku 4.5 on LoCoMo and UltraHorizon, and Claude Sonnet 5 on Terminal-Bench. Each comparison is paired within its own run using the same task model; comparisons across rows may differ in model, harness version, and development history. Table 19 preserves both favorable and unfavorable outcomes.

Table 18: Admitted mechanisms. Verifiers specify inputs, checks, evidence, and responses; other rows describe the operation. A dash indicates an unspecified field.

| Setting | Mechanism | Dimension | Input | Proposition checked | Evidence | On failure |
|---|---|---|---|---|---|---|
| Ultra-Horizon | Commit gate | verification | a drafted answer and its rule class | every logged observation of that class has been replayed against the answer | the pinned per-class notebook | the answer is held, up to three times |
| | Re-analysis sub-call | decomposition | the observation rows of one rule class | the rows of the class do not contradict each other | the provenance-stamped rows | a nested call re-analyses that one rule |
| | Per-class notebook | retrieval | Observation rows and a confound matrix, pinned per rule class. | | | |
| | Probe plans | decomposition | The agent plans deliberate re-probes instead of waiting for evidence to arrive. | | | |
| | Evidence rows | knowledge synthesis | Provenance-stamped, differenced observation rows written as evidence arrives. | | | |
| LoCoMo | Split-or-keep planner | decomposition | Decides whether to split the question; two rules written from development failures keep counterfactual and single-entity-set questions whole. | | | |
| | Claim judge at merge | verification | the claims of an answer synthesised across sub-queries | each claim is supported by the spans retrieved for it | the retrieved spans | – |
| | Re-research round | retrieval | One further retrieval round when the selection for a sub-query is insufficient. | | | |
| | Two answer rules | knowledge synthesis | Order claims by strength and drop none; commit to a best-supported guess instead of refusing. | | | |
| AMA-Bench | Verify-on-build notebooks | verification | a notebook entry as it is integrated | the entry is supported by the chunk it was extracted from | its extracts, cited raw turns and the action ledger | the note is repaired |
| | Deterministic views | retrieval | An event index, an action record, tallies, object timelines and an entity index, always on and compacted. | | | |
| | Write-time notebooks | knowledge synthesis | Per-notebook buffers of per-chunk extractions, from which notebooks are integrated as they fill. | | | |
| PathBench | Rule file | knowledge synthesis | Eleven rules, each a statement with its sources, delivered once before the agent plans; for example, every cluster with a lineage is named by that lineage, and *Unknown* is one specific cluster. | | | |
| | Reference tables | knowledge synthesis | A marker table by label and expression profiles by label and by reference cluster, built from the two development sections. | | | |
| | Nearest-neighbour helper | knowledge synthesis | A script that reassigns the cells of small unlabelled clusters to their nearest labelled neighbours. | | | |
| Terminal-Bench 4 Hard | Edit gate | verification | the first change to a shipped file on a repair task | an invariant and a probe harness exist | the workspace | the change is refused |
| | Repair procedure | knowledge synthesis | A procedure for repair tasks, delivered before the first turn. | | | |
| | Data-repair procedure | knowledge synthesis | A procedure for repair tasks with shipped data, delivered before the first turn. | | | |
| | Scale decision | knowledge synthesis | A decision with two fixed branches, delivered on analysis tasks with a measured-values test. | | | |

Table 19: The verification record. Stage is where the check acts. Split: "dev" is the development set of that run, "val" its validation set. The first row of each setting is the verifier in the tested harness; the other rows were not kept.

| Setting | Check | Stage | Grounded in | Result | Split |
|---|---|---|---|---|---|
| AMA-Bench | verify-on-build (kept) | memory write | extracts, cited raw turns, action ledger | repairs the note, never the answer | dev |
| | every answer vs raw files | final answer | raw trajectory files | rewrote 82/120; 5 repaired, 9 broken; strict 0.263 vs 0.297 | dev |
| | answer rewriter | final answer | draft's evidence | 17/48 → 15/48 (1 rescued, 3 corrupted) | dev |
| | selective, conservative answer check | final answer, gated | raw trajectory files | fired 44/120, changed 12; 6 repaired, 4 broken; superseded by design | dev |
| LoCoMo | claim judge at merge (kept) | evidence merge | cited retrieved spans | best of three pipelines (Table 20) | dev |
| | post-answer checker, every answer | final answer | retrieved material | 22 answers changed: 2 improved, 6 regressed; its "grain" test alone 2 improved, 0 regressed | dev+ 2nd |
| | claim audit widened beyond merges | claims of any answer | retrieved material | 17 answers changed: 0 improved, 3 regressed | dev+ 2nd |
| UltraHorizon | replay-gated commit (kept) | commit | replay of logged rows | dev 30.0 vs 16.7; val 33.3 vs 26.7 | val |
| | replay gate, first form | write + commit | logged rows | alone, committed 2/6 episodes; clean rerun 36.0 vs 36.0 | dev |
| | rival slate, withheld-row replay | write + commit | withheld rows | val 26.7 vs 53.3 | val |
| | sufficiency gate | commit | distinct values per predicate | held every episode; 26.7 vs 36.7 | dev |
| Terminal-Bench | edit gate (kept) | before an edit | hook: invariant and probe exist | admitted within the set of four | dev |
| | constraint checks after each call | after each call | deterministic predicates | admitted by its loop; no validation gain | dev |
| | audit checklist; two-phase audit | finished work | separate auditor call | rewards 1/10 and 0/10; two grader-confirmed fixes; refused | dev |
| | tool-using auditor | finished work | executed adversarial cases | one task 13–15/15 vs 11; within noise on six; 3.2× tokens | dev |
| | adversary on delivered code | finished work | executable probes | two tasks improved; 18–43× tokens | dev |
| | non-author reviewer | finished work | separate model call | no hidden check moved; new loss; 2.1× | dev |
| | self-graded check per requirement | finished work | agent's own checks | gains on two tasks, new losses; 1.9× | dev |
| | second draw and reconcile | finished work | an independent draw | reconcile starved of budget; 2.2–6× | dev |
| | once-per-session stop gates | at stop | hook over agent's ledger | acted once; moved nothing | dev |
| | pre-fix self-graded probes | before an edit | agent's own probes | identical to Claude Code; 1.3× | dev |
| | ledger required before first write | before an edit | hook | no gain over Claude Code | dev |

**AMA-Bench.** The retained verifier checks each notebook entry against its source extracts, cited raw turns, and the deterministic action ledger. A failed check repairs the note, allowing later questions to use the corrected record. Its activity depends on the writer model. With GPT-4o-mini as the writer, it flagged and repaired 17 of the 18 notes it checked; with Gemini 3.7 Flash, the task model of Table 1, all 107 notes it checked across two runs passed, so it made no repairs, at 7 to 8% of tokens. It was not ablated at test. The refused alternative re-checked every final answer against the raw trajectory files, with the instruction that the raw record overrides the draft. It rewrote 82 of 120 development answers, repairing 5 and breaking 9 (turning them wrong or partial); in the record's own words, "always-verify second-guesses correct answers". The same check, gated to counting and state questions and told to keep the draft unless the raw record contradicts it, fired on 44 questions, changed 12, and was net positive (6 repaired, 4 broken). It was not refused on evidence: the final design moved verification from the answer to the notebook, and the gated check was not evaluated again.

**LoCoMo.** Table 20 compares three development pipelines that perform span selection, span checking, and claim judging on every answer, on none, or only at sub-query evidence merges. The merge-only pipeline also contains planner rules derived from the same conversation, so it split 12 questions while the other two split 48 to 53. Judging was blind and three-pass; the ranges come from two judging runs. The merge-only pipeline leads each alternative by four answers net, close to judge noise. On a second conversation it scored 0.836, against 0.829 for a routing baseline and 0.809 for GAM, and the harness built on it gained over GAM on the test conversations (Table 1). The kept claim judge acts rarely: on the test conversations it ran on 12 to 13% of questions, edited claims on 4 to 5%, and dropped at most four claims per replicate. The refused answer-level checkers were

measured within each question: the same question was answered from the same evidence with and without the check, and both answers were graded blind. The checker run on every answer changed 22 answers, improving 2 and regressing 6; three of the six regressions replaced a correct answer with a hedge such as "This cannot be confirmed from the material provided". One of its tests, on the grain of the answer, was positive on its own (2 improved, none regressed). The claim audit widened beyond the merge points changed 17 answers, regressing 3 and improving none. At the level of whole runs, the variant carrying both checkers scored slightly above the kept harness (0.822 against 0.803, pooled over two conversations), a difference within noise; the within-question comparison isolates the checkers' own effect.

Table 20: LoCoMo placement comparison on the development conversation: three pipelines that differ in where span selection, span checking and the claim judge run. Strict accuracy by question path; $n$ is the number of questions on that path in that pipeline.

| Checking runs | single path | split path | overall |
|---|---|---|---|
| on every answer | ≈0.79 | 0.736 ($n$=53) | 0.763–0.770 |
| on no answer | **0.856** ($n$=104) | 0.562 ($n$=48) | 0.763–0.770 |
| where evidence merges (kept) | 0.807–0.856 | 0.667–0.736 ($n$=12) | **0.796** |

**UltraHorizon.** The retained commit gate replays every logged observation of a rule class against the drafted rule and withholds commitment while a replay fails, up to three times. It entered in the second admitted bundle, alongside a per-class confound matrix and a probe plan (development 30.0 vs 16.7; validation 33.3 vs 26.7). On a diagnostic run of 15 frozen episodes on unseen seeds of the development environment, it held answers in 12 of the 15. On the 42-episode test it acts mainly in the sequence environment, rarely in grid, and is disabled in the biology environment. The same loop refused other checks at the commit point. An earlier form of the replay gate, tested in an earlier program with its own six development episodes, let the agent commit in only 2 of 6 when used alone, since the agent could not write rules the gate would accept; with observation tables beside it the agent committed in 5 of 6, and a clean re-measurement tied the baseline at 36.0. A bundle with rival-slate and withheld-row replay checks lost 26.7 points on validation, and a sufficiency gate held answers in every development episode and lost 10 points. The record's summary: "verification rejects, it cannot identify". A gate needs evidence that separates rival rules if it is to support progress in these runs.

**Terminal-Bench.** Preliminary experiments examined several forms of verification: a separate auditor of finished work, a tool-using auditor executing adversarial cases, an adversary writing failing probes against delivered code, a read-only non-author reviewer, and checks written and graded by the agent itself. The variant without localization (Table 3, fifth row) also admitted deterministic constraint checks run after each call, with no validation gain. Several of these did real work. The audits made two fixes that the grader confirmed; the tool-using auditor raised one development task to 13 to 15 of 15 checks against at most 11 for Claude Code, but stayed within noise across six tasks at 3.2 times the tokens; the adversary on delivered code raised two development tasks and, in one run on an earlier loop's held-out tasks, disjoint from the test tasks of Section 3.2 and against a different baseline, gained one check, at 18 to 43 times the baseline's tokens; and self-graded checks gave reproducible gains on two tasks while losing checks elsewhere. None was admitted. The records state the lesson the authors drew from these runs: "the author cannot grade its own adversary: every self-graded check mirrored the fix, fed the code's own formula as EXPECTED, or re-labelled a failing check as a test artefact; a non-author reviewer found real bugs and moved no hidden check." These observations motivated checks supported by an independent draw or a deterministic hook. A second independent draw with a reconcile step was tried, but its reconcile step ran out of budget in most trials, so that remedy was not tested fairly; hooks that fired once at the agent's stop acted as advice. The admitted edit gate is a hook that acts before the first edit of a shipped file. On development, the repair procedure already had the agent write the invariant and probe first, so the gate had nothing to refuse; in development attribution it did not fire on two tasks whose edits went through scripts, and it was judged harmful on one. Its separate effect was not measured.

# O FIGURES

Figure 2 shows the admitted harness in each setting, with mechanisms colored by design dimension. Scores summarize the evaluated bundles. LoCoMo reports three-replicate means; AMA-Bench reports the second generation run, matching the upper block of Table 1.

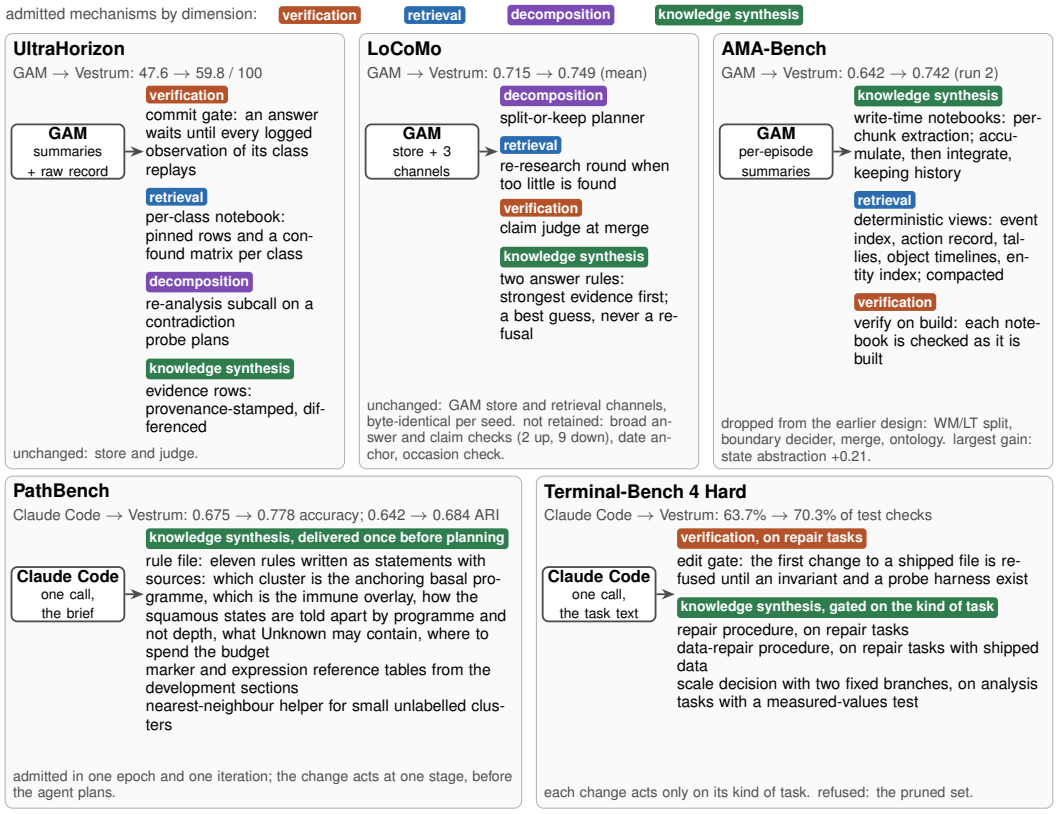



Figure 2: The admitted harnesses. The top row starts from GAM and the bottom row from Claude Code. Each panel shows the mechanisms by design dimension, with retained context and earlier alternatives below. Scores belong to whole bundles; they do not measure individual mechanism effects.