Context Engineering and Loop Engineering for LLM Agents: A Systematic Review, Unified Taxonomy, and Use-Case Fit Framework

Thanatchon Chongmankhong

November 2026
Preprint

Abstract

Large language model (LLM) agents increasingly succeed or fail on two engineered layers around the model rather than on the model itself. *Context engineering* decides what the model sees at each step; *loop engineering*, a term that spread among practitioners in June 2026, decides what keeps an agent working when no person types the next prompt. The two literatures have grown apart, although every loop iteration begins by building a context. This paper places both on one footing. We (i) define both fields with boundary tests against prompt and harness engineering; (ii) model context as the agent's state and the loop as its control, and derive a verifier ceiling and a context-cost law; (iii) build and audit a taxonomy of seven context operations and eight loop levels on a multivocal corpus of 1,395 sources drawn from 5 curated reading lists (544 academic papers, 274 practitioner write-ups and 577 repositories), with held-out coding agreement of $\kappa = 0.86$; (iv) measure the diffusion of the terms across GitHub; (v) re-analyse published results of the LoopArena benchmark; and (vi) run six simulation experiments. Loop papers made up 42% of dated academic sources in 2026, against 30% across 2020–2026, yet only 1.7% of loop-layer sources are benchmarks or empirical studies. Of 56 context-by-loop pairings, 6 co-occur above chance after correction, 4 of them predicted in advance. In LoopArena, the best controller raised strict success over an uncontrolled worker by 8.8 points on full tasks (27.5% to 36.3%) at 13.3$\times$ the cost per run, and no conservative interval excluded zero. Simulation shows that retries converge to the verifier's precision, that appending history wins on short horizons and file-backed resets usually win on cost for long ones, and that self-critique fails most where context policies lose information. We condense the evidence into a use-case fit framework keyed to task horizon, verifiability and risk, with an evidence grade for every recommendation, and release all data and code.

Keywords: context engineering; loop engineering; harness engineering; LLM agents; long-horizon tasks; verification; agent memory; multivocal literature review

1 Introduction

The unit of engineering for LLM systems has moved outward four times in about four years: from the wording of a prompt, to the contents of the context window, to the harness that runs one agent, and to the loop that runs many agents without a person in the chair (Figure 1). *Context engineering* became a common name in mid-2025 [1–3] and received a dedicated survey within weeks [4]. *Harness engineering* followed in early 2026 [5–7]. *Loop engineering* is the newest: a practitioner guide dates its spread to a single week in June 2026, when several prominent engineers described the same shift, namely to stop prompting agents

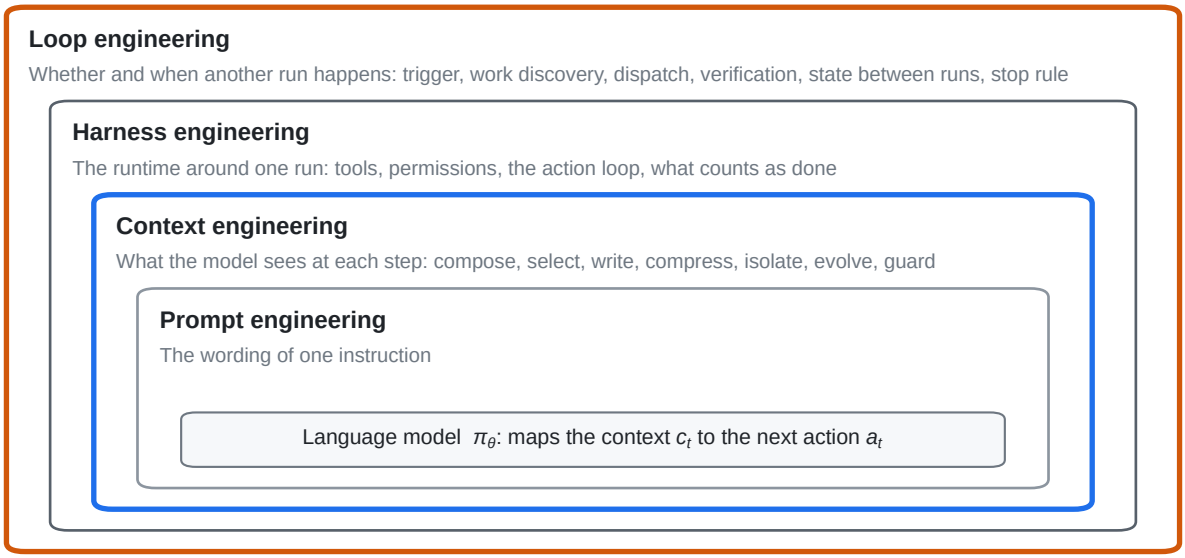


Figure 1: The engineering stack around a language model. Each layer contains the one below it. This paper studies the two outlined in colour, context engineering and loop engineering, and treats harness engineering as the layer where they meet.

and to design the system that prompts them [8, 9]. A recent survey extends the stack further, to graph and ontology engineering for multi-agent systems [10].

Three developments make a joint study timely. First, the harness now matters as much as the model on several agent tasks: benchmark scores move when only the harness changes [11, 12]. Second, tasks have outgrown the window. Agents run for hours across many context windows, so state must live outside the window and be reloaded [5, 13]. Third, the outer loop has become a product feature [14] and an object of academic study [15–17].

Yet the two literatures barely meet. Context studies typically measure answer quality per token inside one run. Loop practice optimises unattended throughput across many runs. The layers are nonetheless coupled: each loop iteration begins by building a context, and each context decision changes what feedback the loop can use. A design that is best for one layer alone can be wrong for the pair. We know of no study that places both on one taxonomy, measures how they interact, and maps their combinations to use cases.

Research questions. We ask six questions. **RQ1** How do academic and practitioner sources define context and loop engineering, and where are the boundaries with prompt, harness and graph engineering? **RQ2** Which method families exist, how has output shifted between 2020 and 2026, and who produces it? **RQ3** How strong is the evidence behind each family? **RQ4** Do context and loop choices substitute for or complement each other in quality, cost and reliability? **RQ5** Which combinations suit which use cases? **RQ6** Which choices keep their value as models improve?

Contributions.

1. Operational definitions of both fields with boundary tests (Section 2).
2. A formal model of context as state and loop as control, with two closed-form results: a verifier ceiling on delivered quality and a quadratic cost law for append-only context (Section 3).

Table 1: Working definitions and boundary tests.

Term	Working definition	Boundary test
Prompt engineering	Design of the wording of one instruction: role, task statement, examples, output format.	Does the change alter only how one instruction is phrased?
Context engineering	Design of the policy that composes, selects, stores, compresses, isolates, evolves and guards the tokens a model receives at each step.	Does it change what enters the window, but not whether or when the model is called?
Harness engineering	Design of the runtime around one agent run: tools, permissions, the action loop, and what counts as done.	Does it change the environment of a single run?
Loop engineering	Design of the control structure that decides when the model is called again, with what goal, on what evidence, and when to stop.	Does it change whether another step or run happens?

3. An open multivocal corpus of 1,395 coded sources, a taxonomy of seven context operations and eight loop levels, and an empirical test of a pre-specified coupling matrix (Sections 4–6).
4. Two secondary analyses: the diffusion of the three terms across 6,278 GitHub repositories, and a re-analysis of the published LoopArena results (Sections 5–6).
5. Six simulation experiments that test six hypotheses on coupling, horizon, verification and model strength (Section 7).
6. A use-case fit framework in which every recommendation carries an evidence grade, and a research agenda (Sections 8–9).

Scope of evidence. Our analysis environment could not reach arXiv or bibliographic databases. We therefore sampled the literature through five community-maintained reading lists hosted on GitHub, which list title, link and usually an arXiv identifier for each source. We did not read the full texts of the papers we code; codes rest on titles, list annotations and the curators' section headings. This is a real limitation, discussed in Section 10; it also makes the study fully reproducible from public files.

2 Definitions and boundaries

Table 1 gives working definitions. Each comes with a boundary test: a question that decides which layer a design choice belongs to.

Loop engineering has a narrow and a broad sense. The narrow sense, from June 2026 practitioner writing, is the outer system that discovers work, dispatches agents, verifies results, keeps state between runs and runs on a timer or an event [8, 9]. One pattern library adds an autonomy ladder: loops start report-only, become assisted, and run unattended only after the verifier has proved reliable [9]. The broad sense covers every iteration level from the action loop of ReAct [18] outward. Our taxonomy uses the broad sense and reports the narrow sense as two of its levels, the session loop and the scheduled loop.

The harness is the bridging layer. It holds the action loop (a loop concern) and the window that the action loop fills (a context concern). Our audit confirmed that this boundary is the hardest to code: most disagreements between coders fell between harness and loop (Section 4.4).

3 A formal model of context and loop

We treat context as the agent's state and the loop as its control. At step t a context policy Φ builds the window from history h_t, external memory M_t, knowledge sources K and tools $\mathscr{T}$, under a token budget B:

$$c_t = \Phi(h_t, M_t, K, \mathscr{T}; B), \qquad |c_t| \le B. \tag{1}$$

The model acts on that context, $a_t \sim \pi_\theta(\cdot \mid c_t)$. A loop policy Λ then reads the evidence e_t, the persisted state s_t and the remaining budget b_t, and chooses a control move:

$$u_t = \Lambda(e_t, s_t, b_t) \in \{\text{CONTINUE}, \text{VERIFY}, \text{BRANCH}, \text{DELEGATE}, \text{RESET}, \text{ESCALATE}, \text{STOP}\}. \tag{2}$$

Design seeks quality Q at low cost C within a harm limit and a cap on human oversight effort H:

$$\max_{\Phi,\Lambda} \mathbb{E}[Q] - \lambda\,\mathbb{E}[C] \quad \text{s.t.} \quad \Pr(\text{harm}) \le \varepsilon, \quad \mathbb{E}[H] \le H_{\max}. \tag{3}$$

Context engineering designs Φ and the rule that writes to M. Loop engineering designs Λ, its triggers, its verifiers and the state s. A harness is the tuple $(\Phi, \Lambda, \mathscr{T}, \text{verifiers}, \text{permissions})$ for one run.

3.1 Two closed-form results

Proposition 1 (Verifier ceiling)**.** *Let one attempt be correct with probability P. A verifier wrongly rejects a correct result with probability f_n and wrongly accepts an incorrect one with probability f_p. A loop that retries until acceptance, up to n attempts, delivers a correct result with probability*

$$\Pr(\textit{correct delivered}) = \frac{a(1-r^n)}{a+b}, \qquad a = P(1-f_n),\ b = (1-P)f_p,\ r = 1-a-b. \tag{4}$$

The yield $1-r^n$ rises with n, but the precision of what is delivered is fixed at $a/(a+b)$. As $n \to \infty$, delivered quality tends to the verifier's precision and cannot exceed it.

Proof. Each attempt independently ends the loop with a correct delivery (probability a), an incorrect delivery (b) or a rejection (r). The probability that the first acceptance happens at attempt $k \le n$ and is correct is $r^{k-1}a$; summing the geometric series gives the result. □

When attempts on one task are correlated, for example because the same context misleads every attempt, the gain from retries is smaller still. We model this with a task-level P drawn from a Beta distribution (Section 7).

Proposition 2 (Context cost)**.** *An append-only window that starts at c_0 tokens and grows by s tokens per step consumes $C_{append}(T) = Tc_0 + \frac{1}{2}sT(T+1)$ input tokens over T steps, which is quadratic in the horizon. A window bounded at B' tokens consumes at most TB'.*

Prompt caching lowers the price of re-read tokens, by a factor $\kappa < 1$, but not their effect on attention. Policies that rewrite history, such as masking and compaction, also invalidate the cache from the rewritten point. The simulation therefore prices fresh and cached tokens separately.

3.2 Taxonomy

Context engineering is described by seven operations (Table 2). Four (write, select, compress, isolate) follow a grouping common in practitioner guides [2]; we add compose, evolve and guard. Loop engineering is described by eight levels, ordered from the innermost to the outermost (Table 3), and by ten design dimensions (Appendix A).

Table 2: Seven context operations. The share column gives the proportion of context-layer sources in the corpus coded with each operation (a source can have several).

Operation	Question it answers	Typical methods	Share
Compose	What standing instructions, examples and tool descriptions sit in the window, and in what order?	System prompts, few-shot examples, repository instruction files, cache-stable prefixes	19%
Select	What is pulled in for this step?	Retrieval, reranking, graph retrieval [19], agentic search, tool and skill retrieval	53%
Write	What is saved outside the window for later?	Scratchpads, notes, state files, memory systems [20, 21]	44%
Compress	What is shortened or dropped?	Summarisation, observation masking, pruning, learned compression [22, 23]	18%
Isolate	What is kept in a separate window?	Sub-agents, branch contexts, recursive calls	–
Evolve	How does the context policy improve from experience?	Evolving playbooks [24], learned curation [25, 26]	3.8%
Guard	What is trusted, attributed or withheld?	Provenance, trust boundaries, injection defences [27]	–

Isolate and Guard are rare inside the context layer because sources that isolate or guard are usually coded to the loop or harness layer; their counts across all layers appear in Figure 5.

Table 3: Eight loop levels. The share column gives the proportion of loop-layer sources coded with each level.

Level	What repeats	Representative work	Share
1 Action	Model call, tool call, observation	ReAct [18]; CodeAct [28]; SWE-agent [29]	–
2 Plan	Plan, execute, replan	StateFlow [30]; AdaPlanner [31]; Plan-and-Act [32]	28%
3 Verification	Produce, check, revise	Self-Refine [33]; Reflexion [34]; CRITIC [35]; Proof-or-Stop [16]	15%
4 Search	Branch, evaluate, select	Tree of Thoughts [36]; LATS [37]	4.3%
5 Orchestration	Delegate, collect, merge	Magentic-One [38]; controller and worker [17]	49%
6 Session	Fresh run, reload state, continue	Long-running harnesses [5]; the “Ralph” loop [39]	15%
7 Scheduled	Trigger, discover work, dispatch, verify, persist	Pattern libraries [9]; [15]	16%
8 Improvement	Run, score, change prompts, context, harness or weights	ADAS [40]; Meta-Harness [12]; Self-Harness [41]	24%

Level 1 sits mostly inside the harness layer, so its share among loop-layer sources is not informative and is omitted.

Hypotheses. The model suggests six hypotheses, stated before analysis in the study protocol. **H1** (verifier ceiling): more loop iterations raise delivered quality only up to the verifier’s precision. **H2** (complementarity): the value of a loop mechanism depends on the context policy it runs with. **H3** (horizon switch): append-only context wins within a few windows of work, and resets with file-backed state win beyond. **H4** (substitution): with a strong verifier, a reset loop with minimal context management matches elaborate context management. **H5** (independent verification): an evaluator with its own context beats self-critique in the working context. **H6** (scaffold decay): the benefit of context and loop scaffolds shrinks as models improve.

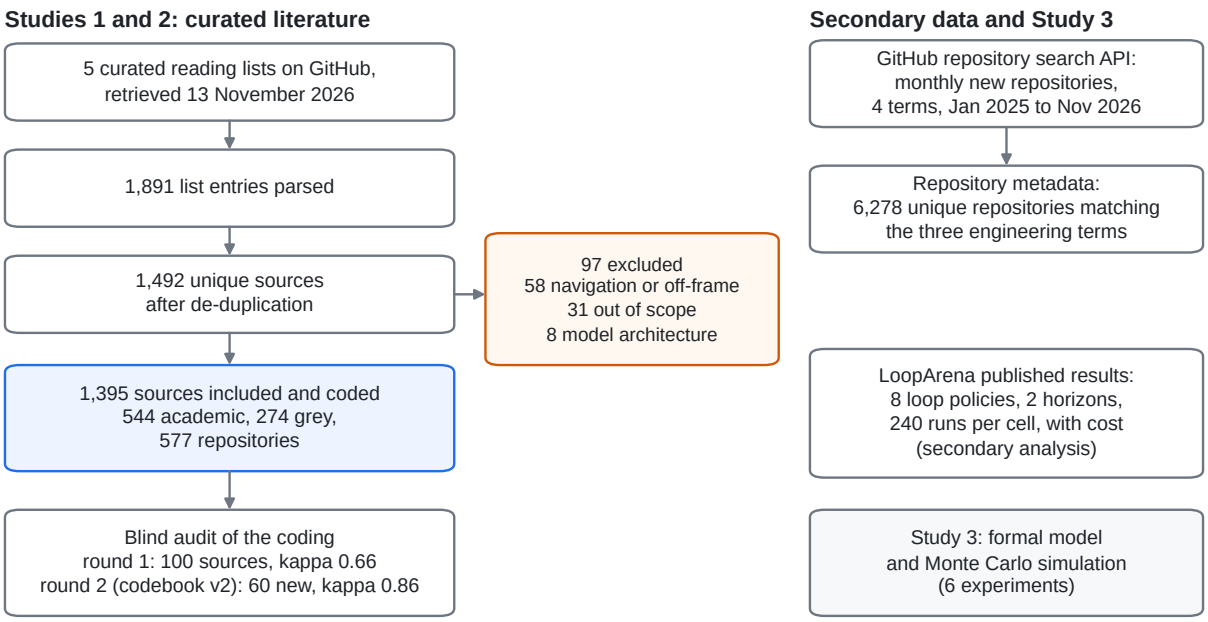


Figure 2: Study design and source flow. Left: the curated-literature stream behind Studies 1 and 2. Right: the secondary data and the simulation study.

4 Method

The study combines a multivocal review with a formal model and two secondary data sets (Figure 2). Grey literature is included because loop engineering began in practitioner writing, which a search of peer-reviewed venues alone would miss [42]. Reporting follows PRISMA 2020 where it applies [43], and procedure follows software-engineering review guidance [44]. All steps run from scripts in the accompanying repository.

4.1 Deviations from the protocol

The study protocol, written before data collection, planned database searches, a factorial experiment with live models, an affiliation map and a validation against production case reports. Four deviations followed from the analysis environment and the chosen scope.

1. **Sampling frame.** arXiv, OpenAlex and similar services were unreachable from the analysis environment. We therefore sampled through 5 community-curated reading lists on GitHub (Section 4.2). This trades recall for reproducibility.
2. **No live experiment.** The factorial experiment (Tier C in the protocol) was not run. In its place we re-analyse the published results of LoopArena [17], which varies loop control with model and task fixed, and we test the hypotheses in simulation.
3. **Geography by proxy.** Author affiliations were not available. We describe practitioner communities through repository metadata instead (Section 4.6).
4. **No case-survey validation.** The fit framework is graded by the evidence behind each recommendation (Section 8) but has not been checked against production case reports.

4.2 Sampling frame and parsing

Five lists form the frame: one centred on context engineering [45], one covering the whole stack up to graph engineering [46], and three centred on harness engineering [47–49]. Their README files were retrieved on 13 November 2026. A parser handles the four entry formats the lists use (HTML items, venue-tagged bullets, Markdown links and Markdown tables) and records the curator's section heading for each entry. Dates come from the arXiv identifier where present, otherwise from venue tags, badges, text or URL. De-duplication merges entries by arXiv identifier, then by normalised URL, then by normalised title. The parser found 1,891 entries and 1,492 unique sources; 180 sources appear in more than one list.

4.3 Screening and coding

Sources were screened by three rules: entries in navigation or off-frame sections (pre-training, prompt wording, ontology engineering, course lists) were excluded; model-architecture work was excluded unless its title also concerned agents, retrieval, memory or compression; and entries with no in-scope term were excluded. 97 sources were removed (58, 31 and 8 by the three rules), leaving 1,395.

Each included source was coded on five dimensions by a rule-based codebook that is released with the data. The *primary layer* (context, loop, harness or evaluation) uses the curator's section heading as a prior and overrides it when the source's own title or annotation carries strong evidence. The seven *context operations* and eight *loop levels* are multi-label codes triggered by terms in the title and annotation, plus codes implied by the curator's heading. The *contribution type* (method, benchmark, empirical study, survey, practitioner write-up, tool) and the *application domain* complete the record. Codes rest on titles, list annotations and headings, not on full texts.

4.4 Audit of the coding

Coding quality was measured against blind manual coding, in which the second coder saw each source's title, annotation and heading but not the rule output. A development sample of 30 sources was used while writing the rules and excluded from both audits. In round 1, codebook v1 coded 100 random sources; agreement on the primary layer was $\kappa = 0.66$ [50], below the protocol's threshold of 0.70. Most disagreements fell between harness and loop. We revised the rules once, placing the action loop inside the harness in line with the framework (Table 3), and drew a fresh sample of 60 sources that had played no part in the revision. Codebook v2 reached $\kappa = 0.86$ on that held-out sample (90% raw agreement; confusion matrix in Appendix C) and $\kappa = 0.76$ on contribution type. The manual coder was the AI assistant that also wrote the rules; the audit therefore measures the consistency of the rules with careful reading, not agreement between independent human experts (Section 10).

4.5 Coupling analysis

For each of the 56 pairs of loop level and context operation we compared the number of sources carrying both codes with the number expected under independence. We report the lift (observed over expected) and a one-sided Fisher exact test, with Holm correction across all pairs [51]. The protocol's draft coupling matrix named 19 pairs in advance, which allows a test of prediction rather than only description.

4.6 Term diffusion on GitHub

We queried the GitHub repository search API [52] for repositories whose name, description or topics contain each quoted term: "context engineering", "harness engineering", "loop engineering" and, as a reference, "prompt engineering". Monthly counts of newly created repositories cover January 2025 to 13 November 2026, with yearly counts for 2023 and 2024. For the three engineering terms we also downloaded metadata for every matching repository, splitting creation-date windows to stay under the API's 1,000-result cap, which gave 6,278 unique repositories. Community proxies are the share of descriptions containing Han characters, the share of repositories owned by organisations, and the share with at least 100 stars.

4.7 LoopArena secondary analysis

LoopArena [17] compares loop policies that steer one fixed coding worker (the same worker model, tools, budget and tasks in every arm). Two reference policies give the worker one instruction and let it run (*no control*) or re-issue a fixed instruction until the worker reports completion (*fixed control*); six controller models instead read a report of the worker's progress and issue the next assignment. Type II tasks are condensed slices of a task, and Type III tasks start from the original, full task state. Each cell has 240 runs (80 tasks, 3 repeats). We used the published aggregate file (successes, runs and normalised cost per run). We report Wilson intervals for success [53] and Newcombe intervals for differences from no control [54]. Because repeats of a task are not independent and per-task data were not published, we also report a conservative interval that counts each task once.

4.8 Simulation

Study 3 extends the closed-form results with a Monte Carlo simulator, specified in Section 7 with parameters in Appendix B. Six experiments (E1–E6) use 3,000 simulated tasks per cell (1,500 in the E6 grid and 600 per policy in the 300 random draws), fixed seeds and common random numbers across policies.

5 Results I: the landscape and its diffusion

5.1 What the corpus contains

Table 4 shows the included corpus by primary layer and source type. Harness sources are the largest group (602), followed by loop (303), context (290) and evaluation (200). This partly reflects the frame, since three of the five lists centre on harnesses. Academic sources dominate the context layer (190 of 290), whereas 82% of harness sources and 50% of loop sources are practitioner write-ups or repositories.

Evidence is thin where practice is busiest (Figure 3). Only 3.1% of context sources, 1.7% of loop sources and 1.7% of harness sources are benchmarks or empirical studies; most academic work proposes methods (55% of context sources, 47% of loop sources). Software engineering is the most common named domain (429 sources), far ahead of research automation (59), web and GUI agents (56) and dialogue (38).

5.2 How the academic literature has shifted

Of academic sources, 95% carry a date. Figure 4a shows the balance moving from context to loop. Context papers peaked in 2024 (67) and numbered less than half that in 2026 (29); loop papers rose from 19 in 2024 to 75 in 2026. Loop work accounts for 42% of dated academic sources in 2026, against 30% across all years,

Table 4: Included sources by primary layer and source type (codebook v2). The last column is the share coded as a benchmark or an empirical study.

Layer	Academic	Grey	Repository	Total	Evaluative
Context	190	39	61	290	3.1%
Loop	152	51	100	303	1.7%
Harness	110	146	346	602	1.7%
Evaluation	92	38	70	200	44.5%
Total	544	274	577	1395	

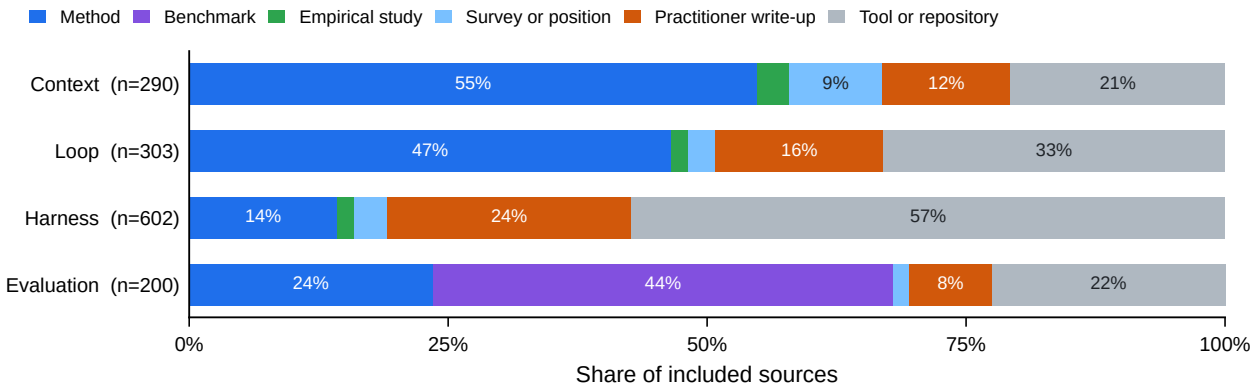


Figure 3: Contribution type by primary layer. Outside the evaluation layer, benchmarks and empirical studies are rare; loop and harness knowledge sits largely in tools and practitioner write-ups.

while context work fell from 37% overall to 16% in 2026. The lists were curated in 2025 and 2026, so older work is under-sampled; the shift within recent years is the more reliable signal.

The terms reached academic titles within weeks of practitioner uptake. A survey of context engineering appeared as arXiv 2507.13334 [4], one month after "context engineering" first reached ten new GitHub repositories in a month (2025-06). Inside the corpus, the first arXiv source naming harness engineering in its title or annotation has identifier prefix 2604 (April 2026), two months after that term reached ten repositories a month (2026-02), and the first naming loop engineering has prefix 2607 (July 2026), two months after 2026-05.

5.3 Families of context engineering

Selection and writing dominate the context layer (Table 2): 53% of context sources select what enters the window (retrieval, reranking, graph and agentic search) and 44% write state outside it (scratchpads, notes, memory systems). Composition (19%) and compression (18%) are less common, and learned or evolving context is still rare (3.8%), although it is the most active recent line, from evolving playbooks [24] to context curation trained with reinforcement learning [25, 26]. Isolation and guarding seldom appear as the primary topic of a context source; they appear instead in loop and harness sources, where sub-agents and trust boundaries are designed.

Three findings recur across the families. First, longer inputs degrade performance even on simple tasks, which practitioners call context rot [55]. Second, compression and memory trade cost against the loss of facts

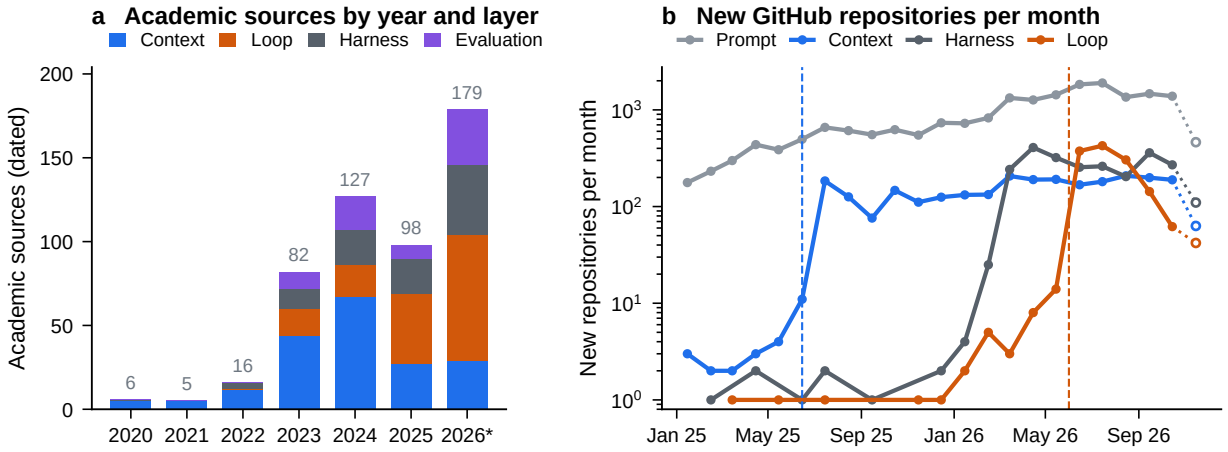


Figure 4: (a) Dated academic sources in the corpus by year and primary layer. (b) GitHub repositories created per month whose name, description or topics contain each term (log scale). Dashed lines mark mid-2025, when "context engineering" spread, and June 2026, when "loop engineering" did.

Table 5: Term diffusion on GitHub. Counts are repositories created in each month whose name, description or topics contain the quoted term (search API, 13 November 2026). Community columns describe the full set of matching repositories: share with Han characters in the description, share owned by organisations, and share with at least 100 stars.

Term	New repos Jan 25–Nov 26	First month ≥10	Peak month (count)	Repos (all)	Han script	Org. owner	≥100 stars
"context engineering"	2,655	2025-06	2026-08 (208)	2,656	3.5%	9.5%	2.3%
"harness engineering"	2,469	2026-02	2026-04 (408)	2,279	11.4%	6.9%	2.4%
"loop engineering"	1,389	2026-05	2026-07 (426)	1,387	8.7%	5.0%	1.2%
"prompt engineering"	19,744	2025-01	2026-07 (1898)	–	–	–	–

a later step needs, and recent work learns that trade-off rather than fixing it by hand [23]. Third, retrieval is moving from a fixed pre-built index towards search tools the agent calls when it needs them [1]. Each of these is a statement about state, and each affects what a loop can do with it.

5.4 Families of loop engineering

Orchestration is the most common loop level among loop sources (49%), followed by planning (28%) and self-improvement (24%); scheduled (16%), session (15%) and verification loops (15%) follow, and search loops are rare (4.3%). The narrow, practitioner sense of loop engineering, the scheduled outer loop, is the least studied in academic venues: it appears mainly in pattern libraries and guides [8, 9] and in a handful of 2026 papers [15, 16, 56]. Two failure modes now have their own studies: loops that never stop [57] and coordination failures in multi-agent systems [58]. Self-improvement loops have turned on the harness itself, with several 2026 systems that rewrite their own harness from traces [12, 41, 59–61].



Figure 5: Co-occurrence of loop levels and context operations across 1,395 sources. Cell values are lifts (observed over expected under independence) with counts beneath; an asterisk marks pairs above chance after Holm correction; outlined cells are the pairs predicted by the protocol's draft matrix. Cells with fewer than three observed and expected sources read "few".

5.5 Diffusion of the terms on GitHub

The three terms spread in sequence (Figure 4b, Table 5). "Context engineering" crossed ten new repositories a month in June 2025, jumped to 184 in July, and has since stayed between about 75 and 210 a month. "Harness engineering" took off in February 2026 and peaked at 408 in April. "Loop engineering" had almost no repositories before May 2026, peaked at 426 in July, the month after it was named, and fell to 62 by October. By October 2026 the monthly figures were 189 (context), 270 (harness) and 62 (loop), against roughly 1,400 for "prompt engineering". The term communities barely overlap: only 43 of the 6,278 repositories mention two or more of the terms.

The repositories are mostly small personal projects. Organisations own 10%, 7% and 5% of them, and the median repository has no stars. Han characters appear in 11% of harness and 9% of loop repository descriptions, against 4% for context engineering, which points to a larger Chinese-language practitioner community around the two newer terms. These are proxies: GitHub is not the whole practitioner population, and description language is not author location.

6 Results II: coupling and loop control

6.1 Which context operations travel with which loops

Loop levels and context operations are not independent (Figure 5, Table 6). 6 of 56 pairs co-occur above chance after correction. 4 of them were among the 19 pairs predicted in advance: improvement loops with evolving context (lift 9.0), action loops with compression (2.53), session loops with writing state (1.84) and

Table 6: Loop level and context operation pairs that co-occur above chance (one-sided Fisher exact test, Holm-adjusted across 56 pairs, $p < 0.05$). "Draft" marks pairs predicted by the protocol's coupling matrix.

Loop level	Context operation	Observed	Expected	Lift	p_{Holm}	Draft
Improvement	Evolve	20	2.2	8.99	< 0.001	yes
Scheduled	Isolate	11	3.8	2.93	0.039	no
Session	Isolate	35	13.8	2.54	< 0.001	no
Action	Compress	20	7.9	2.53	0.002	yes
Session	Write	65	35.3	1.84	< 0.001	yes
Orchestration	Isolate	48	26.9	1.79	< 0.001	yes

Table 7: LoopArena re-analysis. Success is the strict success rate over 240 runs (80 tasks $\times$ 3 repeats). Cost is LoopArena's normalised no-cache estimate in US dollars. The gain column gives Type III percentage points over no control with a 95% Newcombe interval that treats runs as independent.

	Success (%)		Type III cost (USD)		
Loop policy	Type II	Type III	per run	per solved	Gain, Type III (pp)
No control	49.2	27.5	2.61	9.51	–
Fixed control	55.0	31.3	9.47	30.32	+3.8 [−4.4, 11.8]
GPT-5.5	56.7	36.3	34.68	95.67	+8.8 [0.4, 16.9]
Qwen3.7-Plus	56.7	34.2	12.69	37.13	+6.7 [−1.6, 14.8]
Qwen3.6-35B-A3B	55.8	33.8	11.58	34.31	+6.3 [−2.0, 14.4]
Claude Opus 4.8	58.8	33.8	20.63	61.12	+6.3 [−2.0, 14.4]
DeepSeek-V4-Flash-0731	50.8	28.3	32.66	115.25	+0.8 [−7.2, 8.8]
GLM 5.2	40.4	24.2	11.97	49.55	−3.3 [−11.1, 4.5]

orchestration with isolation (1.79). Predicted pairs had a mean lift of 1.81, against 0.97 for all other pairs, and 12 of 19 had a lift above one. The two unpredicted significant pairs both involve isolation: session loops (2.54) and scheduled loops (2.93). In practice, sessions and scheduled runs are given fresh, separate contexts, so isolation belongs to the outer loops as much as to orchestration. The draft's prediction that verification loops lean on isolation and composition was not borne out in the corpus; the data cannot say whether that reflects practice or how sources describe themselves.

6.2 What loop control buys on coding tasks

Loop control helps a little and costs a lot (Table 7, Figure 6). On full tasks, an uncontrolled worker solved 27.5% of runs. Fixed control raised this to 31.3%, and controller models reached 31.7% on average and 36.3% at best (GPT-5.5). The best gain, 8.8 points, has a 95% interval of 0.4 to 16.9 points if runs are independent, and of −5.6 to 22.7 points if each task counts once; no gain is distinguishable from zero under the conservative interval. Controlled policies cost 3.6 to 13.3 times as much per run, so the cost per solved task rose from $9.51 without control to between $30.32 and $115.25 with it. The pattern is the same on condensed tasks, where the best controller (Claude Opus 4.8) reached 58.8% against 49.2%.

Two further patterns bear on the model in Section 7. First, moving from condensed to full tasks cut success by 36% to 44% in relative terms for every policy, including no control (44%), which is consistent with error that grows with the length of the work. Second, controllers that chose better loop contracts in

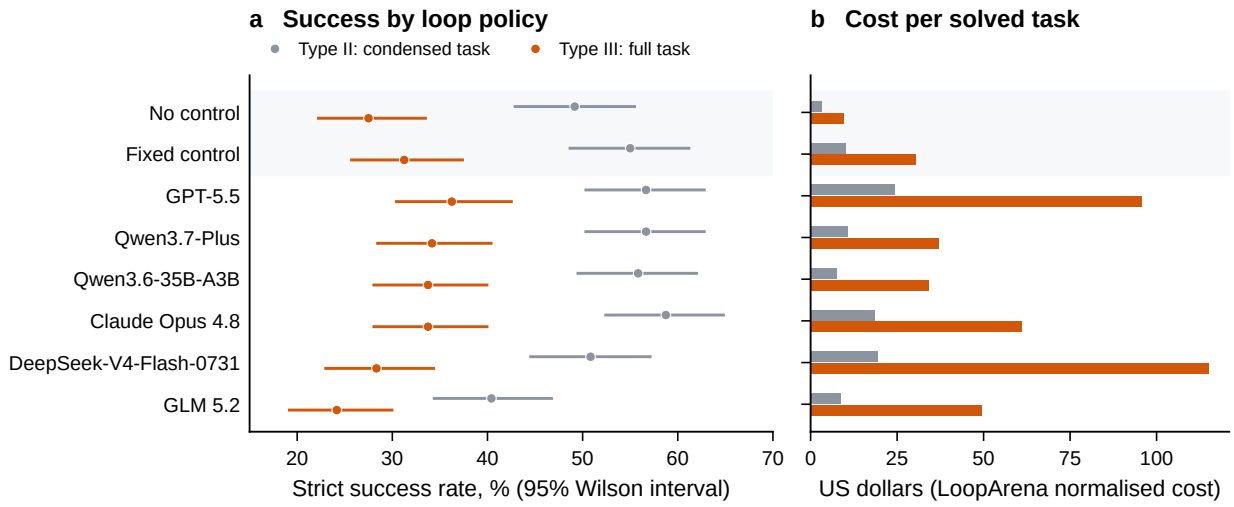


Figure 6: LoopArena re-analysis. (a) Strict success with 95% Wilson intervals on condensed (Type II) and full (Type III) tasks. (b) Cost per solved task. The shaded rows are the two reference policies without a controller model.

LoopArena's single-decision test (Type I) tended to do better on full tasks (Spearman $\rho = 0.70$, $p = 0.12$, $n = 6$), a weak hint that loop control quality is a skill that can be measured in isolation. Relative gains of the best controller over no control were larger on full tasks (32%) than on condensed ones (20%), so whatever control buys grows with the horizon.

These results come from one benchmark, one worker model and one domain, and from aggregate data published by the benchmark's authors. They are evidence about loop control on coding tasks, not about loops in general.

7 Results III: simulation of context and loop

Model. A task needs T successful steps. At step t the context policy sets the window load L_t, and the step succeeds with probability

$$q_t = \left[1 - \varepsilon\,(1 + \gamma L_t / W)\right](1 - m_t), \tag{5}$$

where ε is the model's base step error (drawn per task from a Beta distribution, so attempts on one task are correlated), γ is the factor by which a full window of size W raises the error rate, and m_t is the probability that the step needs a fact the policy has lost. Four context policies are compared. *Append* keeps the whole history and silently truncates at the window limit. *Mask* replaces old observations with short placeholders in blocks of k steps. *Compact* summarises the history whenever it reaches a share θ of the window; each summary drops a needed fact with probability λ_c. *Reset* starts a fresh window every H steps and reloads a state file that lacks a needed fact with probability λ_r. On the loop side, a failed step is caught with probability d and retried up to R times; a retry re-reads files with probability ζ, which recovers lost information. Self-critique shares the working context and so cannot see information that the context lost. Fresh input tokens cost one unit and cached tokens $\kappa = 0.1$; append keeps the cache valid, whereas mask and compact invalidate it at each rewrite. Parameters are listed in Appendix B. They are assumptions chosen to be plausible, not estimates, which is why E6 varies them.

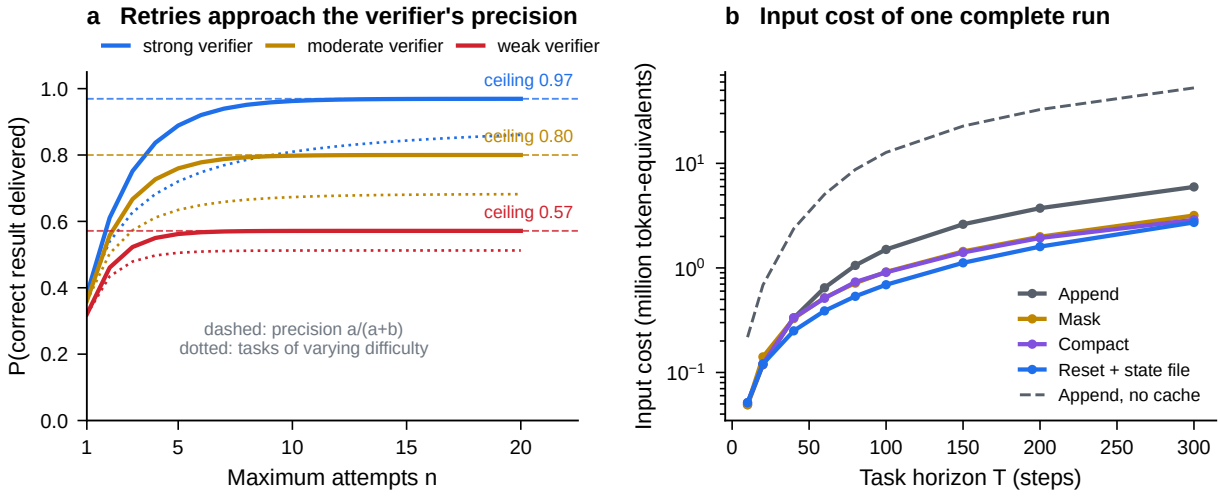


Figure 7: (a) Probability that a correct result is delivered when a loop retries until a verifier accepts, for three verifiers, with base attempt accuracy 0.4. Dashed lines mark the verifier's precision; dotted lines show tasks whose difficulty varies (Beta-distributed accuracy). (b) Input cost of a complete run by context policy, with prompt caching (solid) and, for append, without it (dashed).

7.1 E1 and E2: the verifier ceiling and the cost of context

Retries buy yield, not precision (Figure 7a). With a moderate verifier, one attempt delivers a correct result 0.36 of the time and twenty attempts 0.80, which is exactly the verifier's precision (0.80) as Proposition 1 predicts. When difficulty varies across tasks, hard tasks fail on every attempt and twenty attempts reach only 0.68. A strong verifier lifts the ceiling to 0.97 and a weak one caps delivery at 0.57: the verifier, not the number of retries, sets the quality of an unattended loop.

Caching changes the cost picture more than the choice of policy does (Figure 7b). Over 300 steps, append costs 52.7 million token-equivalents without caching but 6.0 million with it, about 8.9 times less. Reset with a state file remains the cheapest policy (2.7 million), because it both bounds the window and keeps most calls cacheable.

7.2 E3 and E6: which context policy wins, and why

At default parameters the answer depends on the horizon (Figure 8a). Append wins short tasks, and reset with a state file wins long ones. The switch comes earlier as context rot grows: at $T = 150$ steps with no rot ($\gamma = 0$), at 60 with $\gamma = 1$, and at 40 for $\gamma \geq 2$. This supports H3, the horizon switch.

The ranking of long-horizon policies, however, is not robust (Figure 8b). Across 300 random parameter settings at $T = 100$, reset had the highest success in 40% of settings, masking in 25%, append in 21% and compaction in 15%. At $T = 30$ append won 71% of settings. What decides the long-horizon winner is how much information each policy loses per handoff: the state-file loss λ_r, the masking loss δ_m and the compaction loss λ_c were the most important splits in a depth-three classification tree (accuracy 64%), and a one-line rule (choose whichever of reset, mask and compact loses least) picked the winner in 66% of the 238 settings that append did not win, against 33% by chance. Cost is more clear-cut: reset had the lowest cost per solved task in 73% of settings at $T = 100$. In short, no context policy is best for long tasks in general, but resetting from a well-kept state file is usually the cheapest, and the decisive quantity, retention per handoff, is one the literature rarely measures.

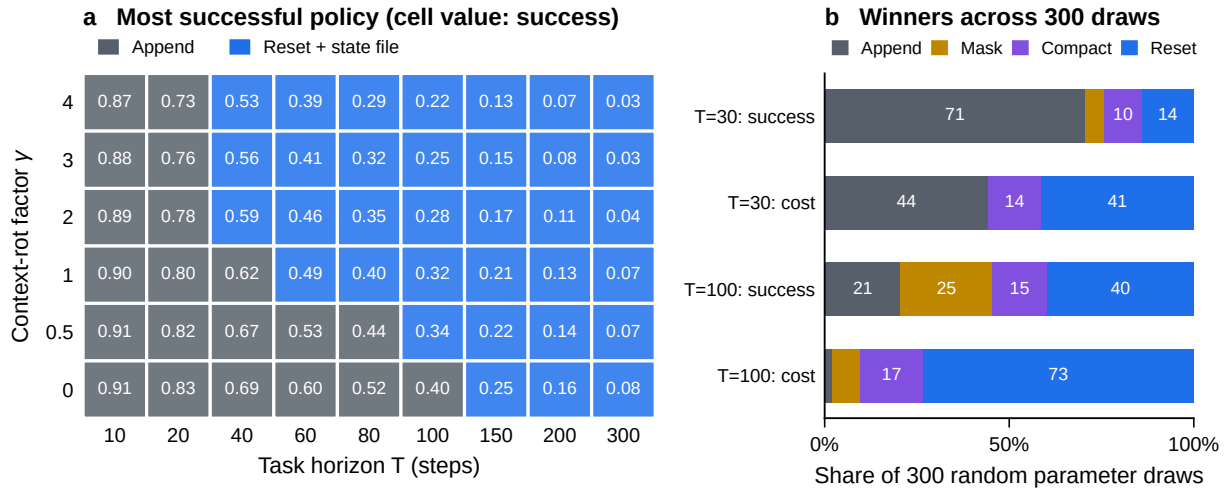


Figure 8: (a) The most successful context policy by task horizon and context-rot factor at default parameters, with the success rate in each cell. (b) Share of 300 random parameter settings in which each policy has the highest success or the lowest cost per solved task, at horizons of 30 and 100 steps.

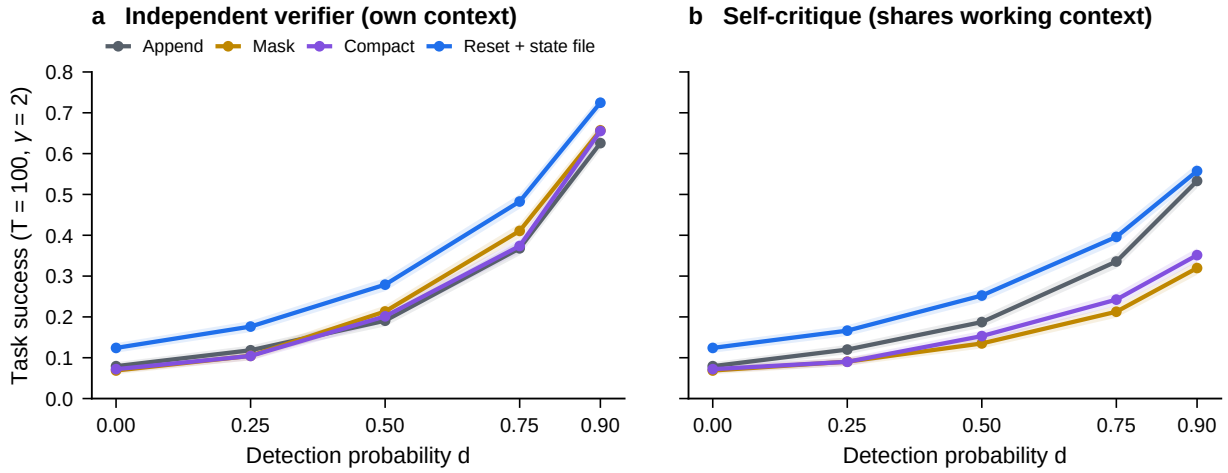


Figure 9: Task success by detection probability for each context policy ($T = 100$, $\gamma = 2$), with (a) an independent verifier that has its own context and (b) self-critique that shares the working context. Bands show 95% Monte Carlo intervals.

7.3 E4: verification and context interact

Stronger detection lifts every policy (Figure 9). With an independent verifier, the gap between reset and append stays almost constant on the log-odds scale as detection rises (0.50 at $d = 0$, 0.45 at $d = 0.9$), so verification strength and context policy act roughly additively; verification does not make the context choice irrelevant, contrary to the strong form of H4. Reset with minimal context management does, however, match or beat the more elaborate policies at every detection level (at $d = 0.9$: reset 0.72, compaction 0.66, masking 0.66, append 0.63).

The type of verifier interacts strongly with the context policy. Replacing the independent verifier with

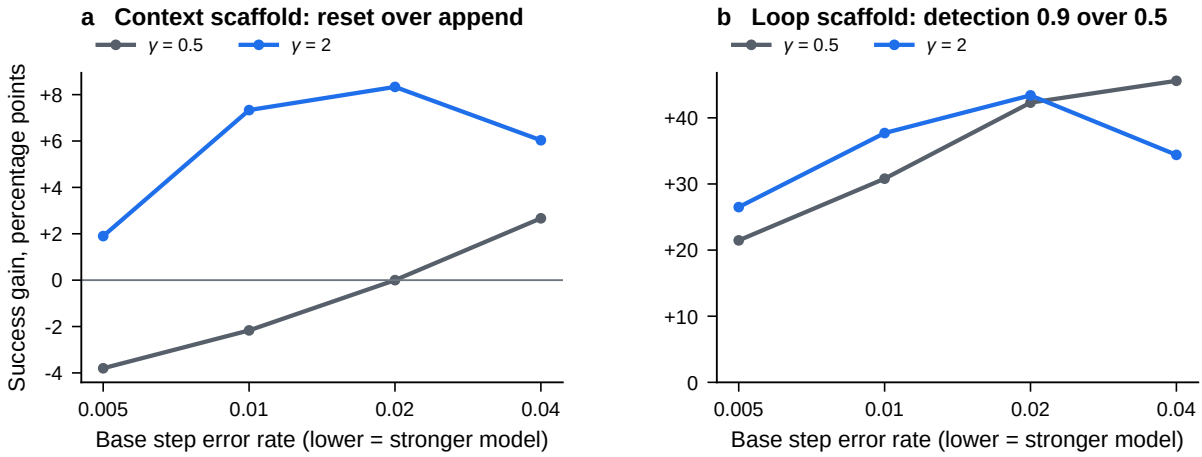


Figure 10: (a) Success gain of reset over append as the base step error falls ($T = 100$, $d = 0.5$). (b) Success gain from raising detection from 0.5 to 0.9 under append.

self-critique at $d = 0.9$ costs append 9 points and reset 17 points, but compaction 30 and masking 34 points, because self-critique cannot detect errors caused by information its own context has lost. This supports H5, and it gives H2 a concrete form: lossy context policies need an independent verifier.

7.4 E5: what happens as models improve

Scaffolds lose value as models get stronger (Figure 10). With mild context rot ($\gamma = 0.5$), reset's advantage over append moves from +2.7 points for the weakest model to -3.8 points for the strongest: the context scaffold turns from help to hindrance. With strong rot ($\gamma = 2$) reset stays ahead, peaking at +8.3 points for mid-range models and shrinking to +1.9 for the strongest. The verifier's gain also shrinks, from 46 to 22 points, but remains large. This supports H6 in the model; whether real model generations follow it is an empirical question that LoopArena, with a single worker, cannot answer.

8 A use-case fit framework

Two properties of a use case predict most of the design choice: how long the task runs and whether a machine can check the result. Risk then sets the ceiling on autonomy. Figure 11 places thirteen use cases on these two axes, and Table 8 gives the recommended context and loop design for each, with an evidence grade.

Evidence grades follow a simple scale. **A**: a controlled comparison with model and task held fixed exists for the loop or context choice in that use case. **B**: the recommendation follows from the formal model or simulation and agrees with academic work in the corpus. **C**: the recommendation rests on practitioner reports, tools or design reasoning only. Only one row reaches grade A, and only for loop control on coding tasks.

The evidence condenses into a decision procedure.

1. **Measure the verifier before adding iterations.** Delivered quality cannot exceed the verifier's precision (Proposition 1). If false acceptances cannot be estimated, keep people in the loop and invest in context instead.
2. **If the task fits one window, append and cache.** Append-only history with a stable prefix won short

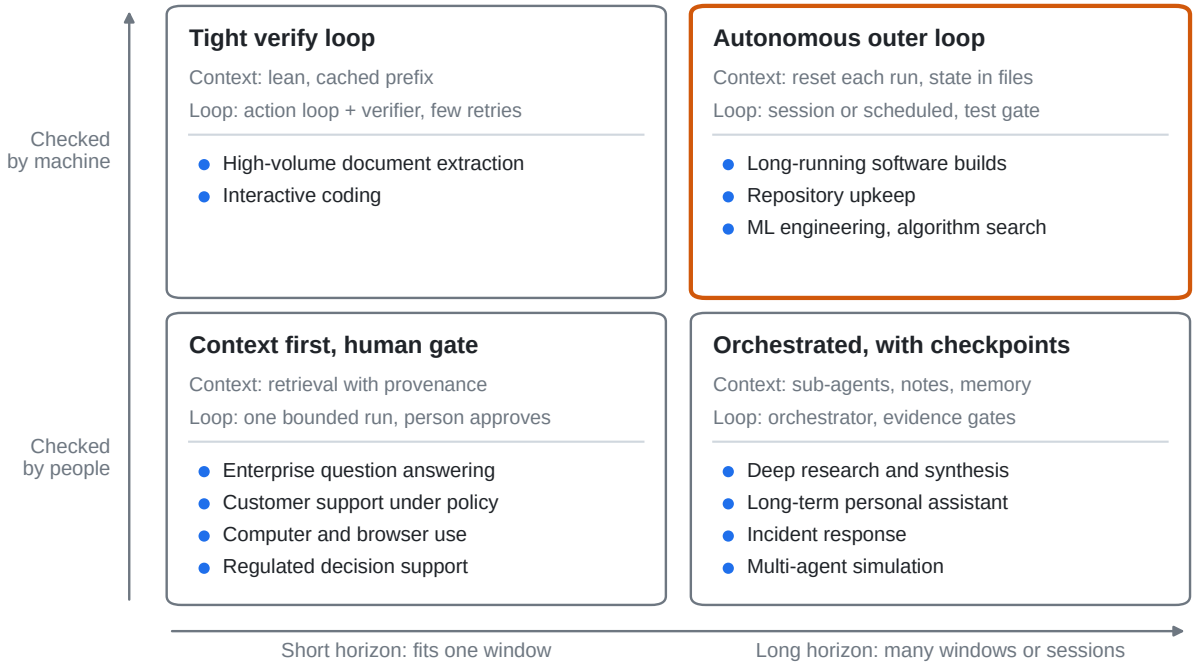


Figure 11: Use-case fit by task horizon and by how results are checked. The top row can rely on machine checks and so supports more autonomy; the bottom row keeps people in the loop. Use cases are placed by quadrant, not by exact position.

horizons in simulation and is cheap once cached (E2, E3).

3. **If it does not, choose the handoff that loses least.** No long-horizon policy wins in general; retention per handoff decides (E6). Reset with a state file is the cost default.
4. **Pair lossy context with an independent verifier.** Self-critique misses the errors that compaction and masking create (E4).
5. **Judge loop control by cost per solved task.** On coding tasks it added at most a few points of success at several times the cost (Section 6.2).
6. **Automate recurring work in steps.** Scheduled loops should move from report-only to unattended only after the verifier has proved reliable [9].
7. **Re-test scaffolds when the model changes.** Context scaffolds that help a weaker model can hurt a stronger one (E5).

9 Discussion

9.1 Answers to the research questions

RQ1, definitions. Context engineering is well defined around what enters the window; loop engineering is defined by practitioners in a narrow sense (the unattended outer loop) and used more broadly in research. The

Table 8: Use-case fit matrix. Horizon: S short (fits one window), M medium, L long (many windows), R recurring, E event-driven. Check: who or what verifies results. Grade: strength of evidence (see text).

Use case	Horizon; check	Context design	Loop design	Autonomy ceiling	Grade
Document extraction at volume	S; schema validator	Lean instructions, examples, cache-stable prefix	Action loop with validator; few bounded retries	Unattended, sampled audits	B
Enterprise question answering	S; people, citations	Hybrid retrieval with reranking; graph retrieval for corpus-wide questions	One run, at most a short retrieve-and-refine loop	Assisted	B
Customer support under policy	S; system state, people	Policy text; account and policy retrieval; small tool set	Fixed workflow around a bounded action loop; escalation to a person	Assisted	C
Interactive coding	S–M; tests, linters	Repository instruction file; agentic search; compress when needed	Action and verification loops with a test gate; person approves the plan	Assisted	B
Long-running software builds	L; tests, browser checks	Reset with a state file, progress log and version control	Session loop with an independent evaluator and budget caps	Unattended in a sandbox; person reviews merges	A, B
Repository upkeep (triage, CI, dependencies)	R; CI status	Run log and state file; guard untrusted issue text	Scheduled loop climbing an autonomy ladder	Report-only first	C
Deep research and synthesis	L; people, citation checks	Parallel isolated searchers; notes; condensed returns	Orchestration with evidence gates and an independent citation check	Assisted	C
ML engineering, algorithm search	L; numeric metric	Experiment journal; program archive	Search and improvement loops	Unattended within a compute budget	B
Computer and browser use	M; page state, people	Prune screenshots and page structure; keep recent steps	Action loop with step checks; confirmation before irreversible actions	Assisted	C
Long-term personal assistant	Many sessions; people	Profile, episodic and semantic memory; scheduled consolidation	Conversational loop; background consolidation loop	Assisted	B
Regulated decision support	S–M; people, audit	Retrieval with provenance; data minimisation; logged context	Bounded loops; mandatory human gate; no self-modifying loop	Report-only or assisted	C
Incident response and operations	E; health checks, people	Runbooks; filtered logs	Event-triggered loop; approval before remediation	Assisted	C
Multi-agent social simulation	L; none	Memory stream with reflection	Orchestration on a simulation clock	Unattended, offline	C

boundary that coders found hardest is harness against loop, because the action loop sits inside the harness. Placing it there, as our framework does, raised coding agreement from $\kappa = 0.66$ to $\kappa = 0.86$.

RQ2, landscape. The corpus holds 1,395 sources. Research attention has moved from context to loop: loop sources make up 42% of dated academic sources in 2026. On GitHub the three terms spread in sequence, from mid-2025 (context) to February 2026 (harness) to June 2026 (loop), and academic titles followed within

Table 9: Status of the six hypotheses.

Hypothesis	Status	Evidence
H1 Verifier ceiling	Supported	Proposition 1; E1: twenty retries reach the verifier's precision and no more, less when difficulty varies across tasks.
H2 Complementarity	Partly supported	E4: self-critique costs lossy policies 30–34 points against 9 for append; the corpus shows above-chance coupling. Not tested on real systems.
H3 Horizon switch	Supported in simulation	E3: append wins short horizons; the switch to reset moves from 150 to 40 steps as context rot grows. LoopArena's 36%–44% relative drop on full tasks is consistent.
H4 Substitution	Not supported in strong form	E4: the reset–append gap is stable on the log-odds scale across detection levels; context choice still matters with a strong verifier. Reset matches or beats elaborate policies, as the weak form predicts.
H5 Independent verification	Supported in simulation	E4: at $d = 0.9$ the independent verifier beats self-critique for every policy, by up to 34 points.
H6 Scaffold decay	Supported in simulation	E5: reset's gain over append falls from +2.7 to -3.8 points under mild rot; the verifier's gain falls from 46 to 22 points. Untested across real model generations.

one to two months each time. The newer terms have a larger Chinese-language practitioner community than context engineering.

RQ3, evidence. Evidence is thinnest where practice is busiest. Benchmarks and empirical studies make up 1.7% of loop and 1.7% of harness sources, and half or more of those layers' sources are practitioner write-ups or tools. Software engineering dominates the named domains.

RQ4, coupling. The layers are coupled in the literature: 6 level–operation pairs co-occur above chance, mostly as predicted. In simulation, verification strength and context policy combine roughly additively, but verifier *type* and context policy interact: lossy context policies lose most when the verifier shares their context. Loop control on real coding tasks bought little success for its cost, so loops do not substitute for a sound context and a good worker.

RQ5, fit. Section 8 gives the framework. Most rows rest on grade B or C evidence.

RQ6, durability. In simulation, context scaffolds lose value and can become harmful as base error falls, while verification keeps a large benefit. We could not test this across real model generations.

9.2 Implications

For practitioners, the most robust results are also the simplest. A loop is only as good as its verifier, so the first investment in an unattended loop is a verifier whose false-acceptance rate is known. Caching, not compression, is the largest lever on the cost of short tasks. For long tasks, the quality of the state that survives a handoff matters more than which handoff mechanism is used. Loop control should be bought by the solved task, not by the run.

For researchers, the gaps are measurable. Few studies report how much information survives a summary, a mask or a reset, although E6 shows that this single quantity decides which long-horizon policy wins. Few studies vary context and loop design independently with the model fixed, as LoopArena does for loop control alone. And no instrument in the corpus measures the human side of a loop: review time, oversight load, or whether people still understand what the loop ships.

9.3 Research agenda

1. **Retention benchmarks.** Measure the probability that a fact needed later survives each handoff policy, across task types.
2. **Factorial experiments.** Vary context policy and loop policy independently, with model and task fixed, beyond coding.
3. **Verifier calibration.** Report false-acceptance and false-rejection rates for the verifiers used in loops, and the precision ceiling they imply.
4. **Costs of autonomy.** Measure human review time and comprehension alongside success and tokens.
5. **Durability.** Re-run the same scaffolds across model generations to test scaffold decay.
6. **Reporting standards.** Report tokens, cost, cache use and repeated runs with every agent result, so that gains such as those in Table 7 can be judged by cost per solved task.
7. **Coverage.** Extend the corpus to Chinese-language sources and to full texts, which this study could not reach.

10 Threats to validity

Construct validity. Codes rest on titles, list annotations and curators' headings, not on full texts. The harness–loop boundary is fuzzy by nature. The manual coder in the audit was the AI assistant that wrote the rules, so $\kappa = 0.86$ measures how well the rules match careful reading of the same evidence, not agreement between independent human experts. A second human coder is the most important addition for a future version.

Internal validity. The codebook was revised once after the first audit. The second audit used a fresh sample that played no part in the revision, but the revision itself was informed by the first sample, whose in-sample agreement is therefore optimistic.

External validity. The frame is five curated lists, three of them about harness engineering, curated in 2025 and 2026 and written in English. The corpus over-represents harness work and recent work, and under-represents Chinese-language and older literature. GitHub counts depend on how the search API matches text, include forks of tutorials and abandoned projects, and stand in for a practitioner population that is larger than GitHub. LoopArena is one benchmark in one domain, with one worker model, published as aggregates by its authors; model names are reported as the benchmark gives them.

Conclusion validity. Simulation results hold for the model as specified. Its parameters are assumptions, which E6 varies, but its structure is also a choice: steps fail independently given the task, verifiers cost nothing, and partial credit does not exist. LoopArena intervals that treat runs as independent overstate precision; the conservative intervals bound the effect of clustering.

Bibliographic validity. Bibliographic details come from the curated lists and from README files opened on 13 November 2026, and are recorded with their provenance in the repository. Full texts were not opened, and author names marked in the provenance file as entered from knowledge should be checked against the original papers.

11 Conclusion

Context engineering and loop engineering are two halves of one design problem: what an agent knows at each step, and whether another step happens. Treating them together yields results that neither literature shows alone. A loop cannot deliver better results than its verifier allows. At long horizons, the information that survives a handoff decides which context policy wins. An independent verifier matters most where context loses information. On real coding tasks, loop control has so far bought little success for its cost. The field's evidence base is thin exactly where its practice is busiest, and the most useful next studies are controlled ones that vary context and loop together, measure retention, and report cost per solved task.

Declarations

Use of AI assistance. This study was carried out with substantial assistance from an AI system (Claude, by Anthropic), which wrote the analysis code, ran the analyses and simulations, performed the manual coding in the audit, and drafted the text under the author's direction. The author takes responsibility for the content.

Data and code availability. All data, code, figures and the LaTeX source are released with this preprint as a single archive intended for a public GitHub repository. Every number in the paper is generated from the processed data by the scripts in the archive.

Competing interests and funding. The author declares no competing interests. The study received no specific funding.

References

[1] Effective Context Engineering for AI Agents. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents, n.d.. Practitioner or vendor write-up.

[2] Martin et al. Context Engineering for Agents. https://rlancemartin.github.io/2025/06/23/context_engineering/, 2025. Practitioner or vendor write-up.

[3] Context Engineering for AI Agents: Lessons from Building Manus. https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus, n.d. Practitioner or vendor write-up.

[4] Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. A Survey of Context Engineering for Large Language Models. https://arxiv.org/abs/2507.13334, 2025. arXiv:2507.13334.

[5] Effective Harnesses for Long-Running Agents. https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents, n.d.. Practitioner or vendor write-up.

[6] Harness Design for Long-Running Application Development. https://www.anthropic.com/engineering/harness-design-long-running-apps, n.d.. Practitioner or vendor write-up.

[7] Agent Harness Engineering: A Survey. https://openreview.net/forum?id=eONq7FdiHa, 2026.

[8] Loop Engineering Orange Book (practitioner guide, English edition v260615). https://github.com/alchaincyf/loop-engineering-orange-book, 2026. Retrieved 13 November 2026.

[9] Loop Engineering: patterns, starters and command-line tools. https://github.com/cobusgreyling/loop-engineering, 2026. Retrieved 13 November 2026.

[10] Yuyuan Feng, Zhishang Xiang, Chaobin Yang, Qichao Ma, Zerui Chen, Yujing Zhang, Ke Huang, Chuanjie Wu, Zhaoxu Liu, Yili Wang, et al. Graph Engineering in the Era of LLM Agents: From Individual Intelligence to System Intelligence. https://arxiv.org/abs/2608.21156, 2026. arXiv:2608.21156.

[11] Harness-Bench: Measuring Harness Effects across Models in Realistic Agent Workflows. https://arxiv.org/abs/2605.27922, 2026. arXiv:2605.27922.

[12] Meta-Harness: End-to-End Optimization of Model Harnesses. https://arxiv.org/abs/2603.28052, 2026. arXiv:2603.28052.

[13] Building a C Compiler with a Team of Parallel Claudes. https://www.anthropic.com/engineering/building-c-compiler, n.d.. Practitioner or vendor write-up.

[14] Getting started with loops. https://claude.com/blog/getting-started-with-loops, 2026. Practitioner or vendor write-up.

[15] Stop Hand-Holding Your Coding Agent: Engineering the Loops That Replace Step-by-Step Prompting. https://arxiv.org/abs/2607.00038, 2026. arXiv:2607.00038.

[16] Proof-or-Stop: Don't Trust the Agent, Trust the Evidence – Loop Engineering for Verifiable Evidence-Gated Lifecycle Control. https://arxiv.org/abs/2607.14890, 2026. arXiv:2607.14890.

[17] DreamX Team. LoopArena: Benchmarking Models as Runtime Controllers for Loop Engineering. https://github.com/AMAP-ML/LoopArena, 2026. arXiv:2608.28281; results from the project repository.

[18] Yao et al. ReAct: Synergizing Reasoning and Acting in Language Models. https://arxiv.org/abs/2210.03629, 2022. arXiv:2210.03629.

[19] Edge et al. From Local to Global: A GraphRAG Approach to Query-Focused Summarization. https://arxiv.org/abs/2404.16130, 2024. arXiv:2404.16130.

[20] Packer et al. MemGPT: Towards LLMs as Operating Systems. https://arxiv.org/abs/2310.08560, 2023. arXiv:2310.08560.

[21] Xu et al. A-MEM: Agentic Memory for LLM Agents. https://arxiv.org/abs/2502.12110, 2025. arXiv:2502.12110.

[22] Jiang et al. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. https://arxiv.org/abs/2310.06839, 2023. arXiv:2310.06839.

[23] ACON: Optimizing Context Compression for Long-horizon LLM Agents. https://arxiv.org/abs/2510.00615, 2025. arXiv:2510.00615.

[24] Zhang et al. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. https://arxiv.org/abs/2510.04618, 2025. arXiv:2510.04618.

[25] Escaping the Context Bottleneck: Active Context Curation for LLM Agents via Reinforcement Learning. https://arxiv.org/abs/2604.11462, 2026. arXiv:2604.11462.

[26] Learning Agent-Compatible Context Management for Long-Horizon Tasks. https://arxiv.org/abs/2605.30785, 2026. arXiv:2605.30785.

[27] Debenedetti et al. Defeating Prompt Injections by Design. https://arxiv.org/abs/2503.18813, 2025. arXiv:2503.18813.

[28] Wang et al. Executable Code Actions Elicit Better LLM Agents. https://arxiv.org/abs/2402.01030, 2024. arXiv:2402.01030.

[29] Yang et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. https://arxiv.org/abs/2405.15793, 2024. arXiv:2405.15793.

[30] StateFlow: Enhancing LLM Task-Solving through State-Driven Workflows. https://arxiv.org/abs/2403.11322, 2024. arXiv:2403.11322.

[31] AdaPlanner: Adaptive Planning from Feedback with Language Models. https://arxiv.org/abs/2305.16653, 2023. arXiv:2305.16653.

[32] Plan-and-Act: Improving Planning of Agents for Long-Horizon Tasks. https://arxiv.org/abs/2503.09572, 2025. arXiv:2503.09572.

[33] Madaan et al. Self-Refine: Iterative Refinement with Self-Feedback. https://arxiv.org/abs/2303.17651, 2023. arXiv:2303.17651.

[34] Shinn et al. Reflexion: Language Agents with Verbal Reinforcement Learning. https://arxiv.org/abs/2303.11366, 2023. arXiv:2303.11366.

[35] Gou et al. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. https://arxiv.org/abs/2305.11738, 2023. arXiv:2305.11738.

[36] Yao et al. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. https://arxiv.org/abs/2305.10601, 2023. arXiv:2305.10601.

[37] Zhou et al. Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models. https://arxiv.org/abs/2310.04406, 2023. arXiv:2310.04406.

[38] Fourney et al. Magentic-One: A generalist multi-agent system for solving complex tasks. https://arxiv.org/abs/2411.04468, 2024. arXiv:2411.04468.

[39] Supervising Ralph Wiggum: Exploring a Metacognitive Co-Regulation Agentic AI Loop for Engineering Design. https://arxiv.org/abs/2603.24768, 2026. arXiv:2603.24768.

[40] Hu et al. Automated Design of Agentic Systems. https://arxiv.org/abs/2408.08435, 2024. arXiv:2408.08435.

[41] Self-Harness: Harnesses That Improve Themselves. https://arxiv.org/abs/2606.09498, 2026. arXiv:2606.09498.

[42] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, 106:101–121, 2019.

[43] Matthew J. Page et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ*, 372:n71, 2021.

[44] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-01, Keele University and Durham University, 2007.

[45] Awesome Context Engineering (curated list). https://github.com/Meirtz/Awesome-Context-Engineering, 2026. Retrieved 13 November 2026.

[46] Awesome Graph Engineering (curated list). https://github.com/DEEP-JLU/Awesome-Graph-Engineering, 2026. Retrieved 13 November 2026.

[47] Awesome Harness Engineering (curated list). https://github.com/ai-boost/awesome-harness-engineering, 2026. Retrieved 13 November 2026.

[48] Awesome Harness Engineering (curated list). https://github.com/walkinglabs/awesome-harness-engineering, 2026. Retrieved 13 November 2026.

[49] Awesome Agent Harness (curated list). https://github.com/Picrew/awesome-agent-harness, 2026. Retrieved 13 November 2026.

[50] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[51] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[52] GitHub REST API: search repositories. https://docs.github.com/en/rest/search/search, 2026. Retrieved 13 November 2026.

[53] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[54] Robert G. Newcombe. Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in Medicine*, 17(8):873–890, 1998.

[55] Diagnosing and Mitigating Context Rot in Long-horizon Search. https://arxiv.org/abs/2606.29718, 2026. arXiv:2606.29718.

[56] ResearchLoop: An Evidence-Gated Control Plane for AI-Assisted Research. https://arxiv.org/abs/2605.28282, 2026. arXiv:2605.28282.

[57] When Agents Do Not Stop: Uncovering Infinite Agentic Loops in LLM Agents. https://arxiv.org/abs/2607.01641, 2026. arXiv:2607.01641.

[58] Cemri et al. Why Do Multi-Agent LLM Systems Fail? https://proceedings.neurips.cc/paper_files/paper/2025/hash/b1041e52d3be19f0a9bc491657488e4a-Abstract-Datasets_and_Benchmarks_Track.html, 2025.

[59] Agentic Harness Engineering: Observability-Driven Automatic Evolution of Coding-Agent Harnesses. https://arxiv.org/abs/2604.25850, 2026. arXiv:2604.25850.

[60] HARBOR: Automated Harness Optimization. https://arxiv.org/abs/2604.20938, 2026. arXiv:2604.20938.

[61] Evolving Agents in the Dark: Retrospective Harness Optimization via Self-Preference. https://arxiv.org/abs/2606.05922, 2026. arXiv:2606.05922.

A Ten loop design dimensions

Table 10: Design dimensions of a loop. Any loop at any level can be described by one option on each dimension.

Dimension	Options
Trigger	Person, schedule, event, another agent
Iteration unit	Tool call, turn, task, session, generation
State carrier	Window history, summary, files, version control, database, weights
Feedback source	Environment, deterministic check, judge model, person
Controller	Fixed code, the working model, a separate controller model, a person
Stop rule	Self-declared done, verifier pass, budget, convergence, sign-off
Recovery	Retry, backtrack, rollback, reset, escalate
Topology	Single thread, fan-out, tree, population, hierarchy
Budget governance	Caps on steps, tokens, cost and time
Oversight	Before action, at checkpoints, or after the fact; autonomy from report-only to assisted to unattended

B Simulation parameters

Experiment E1 uses a base attempt accuracy of 0.4 and three verifiers: strong ($f_n = 0.05$, $f_p = 0.02$), moderate (0.10, 0.15) and weak (0.20, 0.40); the heterogeneous variant draws accuracy from a Beta distribution with mean 0.4 and concentration 2. E3 crosses nine horizons with six values of γ. E4 uses $T = 100$, $\gamma = 2$ and five detection levels. E5 crosses four base error rates, two values of γ and two detection levels at $T = 100$. E6 has two parts: a grid of compaction and state-file loss rates at $T = 100$, and 300 random draws of γ, $\bar{\varepsilon}$ (log-uniform), ρ, δ_m, λ_c, λ_r, d, k, H and s, each evaluated at $T = 30$ and $T = 100$.

The depth-three classification tree fitted to the $T = 100$ draws (winner by success) is reproduced below; `lam_r`, `delta_mask` and `lam_c` are the per-handoff loss rates and `s` is the tokens added per step.

```
|--- lam_r <= 0.17
|   |--- s <= 1764.83
|   |   |--- class: append
|   |--- s >  1764.83
|   |   |--- delta_mask <= 0.13
|   |   |   |--- class: mask
|   |   |--- delta_mask >  0.13
|   |   |   |--- class: reset
|--- lam_r >  0.17
```

Table 11: Simulation parameters and default values. Experiments vary the parameters named in the text; E6 draws ten of them at random.

Symbol	Meaning	Default
T	steps the task needs	60
W	window size (tokens)	200,000
c_0	fixed prompt, tools and task (tokens)	8,000
s	tokens added per step	2,500
γ	error-rate rise at a full window	2
$\bar{\varepsilon}$	mean base step error	0.02
κ_ε	Beta concentration of step error across tasks	100
ρ	P(step needs an older fact)	0.05
k	mask: observations kept verbatim (block size)	10
p_m	mask: tokens per masked observation	60
δ_m	mask: P(masked fact cannot be re-derived)	0.3
θ	compact: trigger as share of W	0.6
σ	compact: summary size as share of W	0.08
λ_c	compact: P(summary drops a needed fact)	0.2
H	reset: steps per session	20
u	reset: state-file tokens per step	150
λ_r	reset: P(state file lacks a needed fact)	0.1
d	detection probability	0.5
–	detection blind to lost information	no
R	retries per step	2
ζ	P(retry re-reads files and recovers)	0.8
κ	price of a cached token (fresh = 1)	0.1

```
|   |--- delta_mask <= 0.27
|   |   |--- s <= 2387.65
|   |   |   |--- class: append
|   |   |--- s >  2387.65
|   |   |   |--- class: mask
|   |--- delta_mask >  0.27
|   |   |--- lam_c <= 0.11
|   |   |   |--- class: compact
|   |   |--- lam_c >  0.11
|   |   |   |--- class: reset
```

C Audit of the coding

Codebook v2 made four changes to v1, all motivated by round-1 disagreements: section headings about agent loops, runtime orchestration, interaction protocols and harness architecture map to the harness layer, because the action loop sits inside the harness; skill composition maps to context selection; meta-harness generators and workflow profiles map to the loop layer; and explicit evidence of search, scheduled or self-improvement loops in a source's own title or annotation overrides a generic section prior. One spelling rule ("tree-search") and one evolve rule (evolving skills) were also added. Round-1 agreement for v1 was $\kappa = 0.66$ on layer and

Table 12: Held-out audit (round 2, codebook v2, $n = 60$): rule-based primary layer (rows) against blind manual coding (columns).

Rule \ manual	Context	Evaluation	Harness	Loop
Context	14	0	0	1
Evaluation	0	6	1	0
Harness	0	1	24	2
Loop	0	0	1	10

$\kappa = 0.78$ on contribution type.