Sleep Consolidation: A Repeated Correction Earns a Diagnosed, Tested Repair

Oscar Serra · Independent Research

21 October 2026

Abstract

Tell a good new colleague something once and it sticks. If it slips, they work out why: the note was in the wrong drawer, an older instruction said the opposite, or it was never clear. If it still slips when none of those is the reason, the lesson has to become a habit, and in people part of that work happens in sleep.

AI assistants are corrected daily, and their owners keep repeating themselves. The model cannot change, so lessons are written down as text, which the model weighs as advice against everything else it reads. When a written lesson fails, assistants write another note.

We propose that a correction the owner has to repeat earns a diagnosed, tested repair instead of another note. The assistant counts each broken rule across sessions and, each time one breaks, works out why, overnight: it never reached the model, another line contradicted it, or it was too vague. Each answer has its own repair. Only a rule that was in plain view, clear and unopposed, and still broken moves closer to a check that runs before the action. A new check is tried in the background before it may block anything.

The owner's words are never lost, the assistant never claims to have learned more than it has, and only the owner can relax a saved rule or make one part of what every session reads. This is a design, with the tests that would decide it.

If it holds, no repeat is wasted: each one removes a different cause, and soon you stop having to repeat yourself.

1. Introduction

On a Monday the owner writes to his assistant in Finnish and gets an answer in English. He says so: reply in the language I write in. The assistant apologises and saves a note. Ten days later it happens again; the note sat in a memory store, and nothing brought it into that turn. This time the rule goes into the file the assistant reads on every turn. A month later it happens a third time: the file had grown long, and an older line in it, written for a different situation, said to answer in English. The owner gave one instruction three times, and the assistant wrote it down three times.

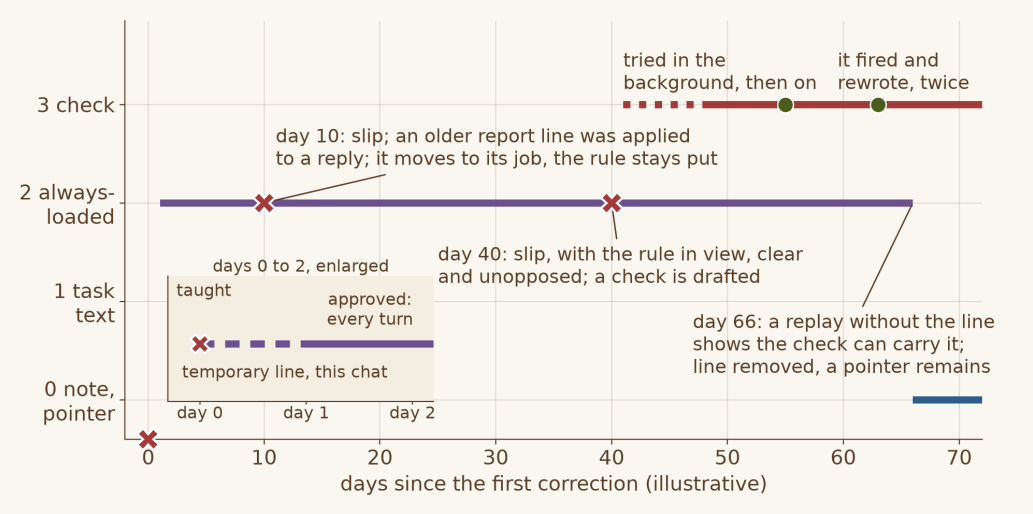


Figure 1. One rule's life, the reply-language rule of §1. Each bar is where the rule lived (dashed: for this chat only, or on trial), each cross a slip (the first, on the axis, came before the rule existed), each dot a wrong reply the check rewrote; the inset enlarges days 0 to 2. Days are illustrative.

Each time, the lesson was saved. What the assistant lacked was a way to notice that a saved lesson had failed, to find out why, and to repair that cause. This paper is about that missing step: how an agent whose model cannot change turns experience into behaviour that stays changed.

1.1 What the brain suggests

The brain learns at two speeds. In the complementary learning systems account (McClelland et al., 1995), the hippocampus records an episode at once and the neocortex folds many episodes into durable knowledge, slowly and offline. Sleep also seems to improve what was practised: in a classic result, a finger sequence rehearsed in the evening is played faster the next morning (Walker et al., 2002), though how much of that gain is due to sleep is debated (Pan & Rickard, 2015). That is the whole biological claim here. What we borrow is the shape: record at once, consolidate later, and let a lesson that keeps slipping take a firmer form.

1.2 Why the obvious fixes fail

Retraining is not the answer for one owner. Most owners use hosted models they cannot retrain, and a lesson trained into the weights cannot be inspected or removed on its own. So the lessons live outside the model, as text and code (Table 3 compares fine-tuning with the rest).

Text binds weakly. Claude Code's documentation says its memory files are treated "as context, not enforced configuration"; to block an action whatever the model decides, it points to a hook (Anthropic, 2026a). Safety engineering likewise ranks rules people must remember below controls built into the equipment (NIOSH).

Text competes for attention. On IFScale, a benchmark that keeps adding instructions until models fail, accuracy falls as instructions are added, and earlier ones are followed more than later ones (Jaroslawicz et al., 2025). A file that grows by one lesson a day gets worse at following them, the later ones first.

Learning only at night is late, and pruning by age or size moves rules that still matter out of sight, where nothing prompts a search for them.

The learner learns from the wrong signal. An outage is not fixed by advice, the text a harness wraps around the owner's message is not the owner speaking, and a slip on a rule the

model may never have received says nothing about the rule. And silence hides failure: a night that found nothing to learn and a night that never ran leave the same empty trace.

1.3 What this paper proposes

The claim: a repeated correction earns a diagnosed, tested repair instead of another note. Builders already know this path from code review (Table 1). An agent's rule can live in four places, which we call rungs, 0 to 3: a note found by search, a line in a task's instructions, a line read on every turn, and a check in code (§5.1). When a rule slips, the agent first asks why (§5.4). Only a slip with the rule in front of the model, clear and unopposed, moves the rule toward a check; we call that a **clean slip**.

One more piece runs through the paper. Filing a correction takes a while, so the moment the owner speaks, his sentence goes up like a sticky note on the screen: a short block, the **tail**, added to every model call in that chat, after his newest message. It comes down once the filed copy can be found (§6.1). Figure 1 follows the opening's rule through the design, and Table 3 (§2) sets the design beside what assistants do today. The numbers we propose rather than derive (time limits, budgets, waiting periods) are gathered in Appendix A.4, with what each trades.

Table 1. The design as a code-review pipeline.

On a software team	In this design
A reviewer writes "no console.log in production code" for the third time	The owner corrects the same thing again
Someone keeps a tally of how often each comment comes up	The rule ledger counts incidents per rule, out of the times the rule applied (§4)
Before adding a lint rule, the lead asks why people keep missing it: is the guide linked from the template, does an old README say the opposite, is the guideline vague, is it on page 40?	The diagnosis (§5.4): did the rule reach the model, did another text contradict it, was it vague, was it buried? Each cause gets its own repair
The guideline moves into the checklist every pull request shows	Rung 2, the always-loaded file
A new lint rule runs as a warning for a while	The shadow run (§5.3)
The warning becomes a failing build	"Enforced"
A disable comment, with its reason	The owner's override, kept in a receipt
Only the lead can turn a failing rule back into a warning	Only the owner loosens a rule (§7.4)
Where the picture stops	A teammate learns; this model cannot. CI checks code in a sandbox, while these checks stand in front of live actions, so each needs a timeout and a declared fallback (§5.3)

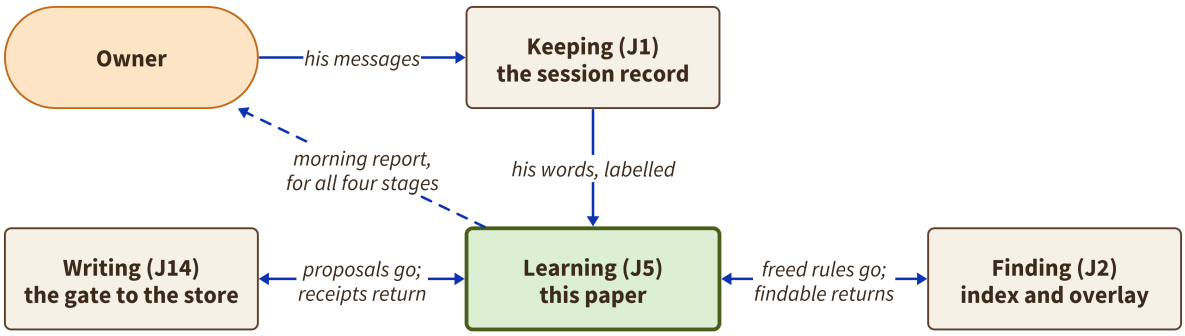


Figure 2. Where the learning stage sits. The owner's words reach it from the keeping stage, each marked with who said it; its proposals go through the writing stage's gate, which answers each with a receipt; the finding stage indexes the rules moved off the always-loaded file and says when a new item can be found; and the morning report is the one channel to the owner for all four stages.

2. Related work

This is one of four designs for an agent's memory. The keeping paper (J1) decides what survives the context window and keeps the full session record as ground truth. The writing paper (J14) decides what enters the long-term memory store and in what shape. The finding paper (J2) gets the right thing back when it is needed. This paper, the learning stage (J5), turns experience into behaviour. Figure 2 shows what passes between the four, and Table 2 explains the words this paper borrows from the other three.

Lessons kept as text. Reflexion (Shinn et al., 2023) has an agent reflect on a failed attempt and read the reflection on the next try. ACE (Zhang et al., 2025) keeps a playbook of lessons, each with counters of how often it helped or misled, and its reflector names the root cause of a failed attempt. ExpeL (Zhao et al., 2024) keeps insights with an importance count and drops one whose count reaches zero. Neither asks why a lesson that applied was not followed: whether it reached the model, whether another line contradicted it, whether it was too vague. That question is the centre of this design (§5.4).

Learning while idle. Sleep-time compute (Lin et al., 2025) lets an agent think about its context before a question arrives. Letta's "dreaming" consolidates lessons in the background without asking the user (Letta, 2026a, 2026b), and Anthropic's Dreams rebuild a memory store from past transcripts into a new store a person reviews (Anthropic, 2026d), the nearest relative of our copy-then-adopt night (§4.3). Neither counts how often a lesson failed or asks why.

Products that write lessons down. Claude Code loads its instruction files and its own notes, including "corrections you give Claude", at the start of every session, scopes rules to file paths and audits them for conflicts (Anthropic, 2026a); its troubleshooting list for an ignored instruction is most of our diagnosis, done by hand (Table 8). The Claude memory tool (Anthropic, 2025) and LangMem (LangChain, 2025) keep lessons as files or prompt edits. Between them they supply our rungs 0 to 2; none counts a written lesson's failures.

Checks made from corrections. Claude Code's PreToolUse hooks can deny a tool call, ask the user or replace its arguments (Anthropic, 2026b). The hookify plugin, the closest shipping relative of our top rung, finds behaviour the user corrected in the recent conversation and drafts rules that warn (the default) or block, in force at once (Anthropic, 2026c). Guardrail frameworks (NeMo Guardrails, Rebedea et al., 2023; OpenAI, 2026) say how to run a check, not when a lesson has earned one, and HASP (Liu et al., 2026) turns advice-style skills into functions that fire on failure-prone states. Self-Harness (Zhang et al., 2026) comes closest to the whole loop: it mines execution traces for recurring failures and accepts a harness edit only after regression tests. We take from them the executable check and the test before acceptance.

Optimising texts against a score. Prompt optimisers such as OPRO, DSPy and GEPA keep the rewrites that score better (Yang et al., 2023; Khattab et al., 2023; Agrawal et al., 2025); from Wahi's (2026) study of one such loop we adopt deterministic acceptance checks, frozen holdouts, and roles that draft changes they cannot apply (§7.4, §9.2).

What is new here. The parts exist; we did not find a design that puts these three together:

1. a count per rule that persists across sessions and across the rule's forms, made of incidents, not mentions (§4.2);
2. on every incident, a diagnosis of why a rule that applied was not followed, tested by replaying the call, with a different repair for each cause (§5.4);
3. a check that acts only after a shadow run on live traffic, with stated error bounds (§5.3), and the owner's approval for anything that loosens a rule or reaches every turn (§7.4). Running a policy silently before it may block is standard in Kubernetes admission control (Gatekeeper's dry run, Open Policy Agent, 2026; Kyverno's Audit mode, Kyverno, 2026); what is new is what earns a check, a diagnosed clean slip, and the bounds stated in advance.

Self-Harness is the nearest automated precedent; here the failures are the owner's own corrections, counted per rule, and the question is why an existing instruction failed. Table 3, at the end of this section, sets the approaches side by side, with what each costs the owner.

Table 2. Words from the other three papers.

Word	What it means here
Session record	The full, never-edited transcript of every session, kept by the keeping stage; everything else points into it by address (J1 §4)
Cached prefix	The start of the model's input that stays the same from call to call, billed at a cached rate; changing a line in it makes the provider read everything after it again (J2 §7.4)
Gate	The writing stage's single way into the long-term store; every write passes it and gets a receipt (J14 §5.1)
Judge	The small model at the gate that compares a new write with what is stored, for duplicates and clashes (J14 §6.3)
Receipt	The gate's record of what became of a write: accepted, held, refused or pending (J14 §5.4)
Published snapshot	The version of the store, or of the always-loaded file, that readers see now; a change becomes visible only when a new one is published (J14 §7.2)
Index file	The always-loaded head of the store, with pointers to what can be found (J2 §4.3)

Table 3. What a repeated correction becomes under each approach, told through the opening story, and what it costs the owner. The second column is what each does better than a plain note. The design is not built yet.

Approach	What it does well	On day 40 of §1, what becomes of "reply in the language I write in"	Who notices a failed lesson	What it costs the owner
Retrieval or vector memory (Mem0; Chhikara et al., 2025)	Nothing to curate; finds what nobody flagged	A note, there only if that turn's search matches; the reply goes out in English, and nothing records the miss	Nobody	Nothing
A running summary	No setup; keeps a long session coherent	A sentence the last compaction may have dropped; the same reply, the same silence	Nobody	Nothing
A bigger context window	No retrieval miss inside the window	His day-0 message, somewhere in a long chat (Jaroslawicz et al., 2025); gone in a new session	Nobody	Tokens on every call
Fine-tuning on his corrections, where offered	Changes the model; takes no room in the prompt	One example in the next training run, if there is one	A later evaluation, if someone writes it	A training run per update
Lesson playbooks (ACE, ExpeL)	Lessons from many tasks, weeded by their counts	A lesson that may be counted unhelpful; nobody asks whether it reached the model	A tally over many tasks	Nothing
ChatGPT or Claude built-in memory	Effortless; he sees and deletes what was saved	A saved line, weighed against the rest; he re-adds or rewords it	The user, by hand	Deleting wrong memories by hand
Night consolidation (Letta's dreaming; Anthropic's Dreams)	Works in the background; Dreams rebuild a store from up to 100 transcripts	A memory edit made overnight, or a merged entry in a rebuilt store; no check	Nobody	Nothing, though Letta changes memory unasked; reviewing a rebuilt store
hookify	Finds corrections in the last 20 to 30 messages	If he asks, a pattern rule drafted from the chat; a reply's language is hard to match	The user, in the recent conversation	Picking rules; no counted trial before one blocks
Self-Harness	Tests each harness edit against regressions	A harness edit, if the traces show a pattern; nothing tied to his rule	An offline search of traces	Nothing, and no say
This design	Repairs each slip's cause; tests a check before it acts	One earlier incident, repaired (L-7); the diagnosis finds a clean slip; a check is drafted and tried in the background	The ledger, at each incident; a weekly blind sample (§7.5)	Four approvals for this rule; a report only on mornings with something to decide

3. What the agent learns from

3.1 The owner's correction is the label

For a personal assistant the best teacher is the person it works for: a correction is rare, it costs him something, and it states the standard he holds. We rank the signals, strongest first:

1. an explicit mark: "remember this", "that was wrong", or a button that records the same;
2. a correction in his next turns: "I asked you to", "again", a terse "no";
3. a reversal, when he undoes the agent's change or rewrites its draft;
4. praise for something the agent did of its own accord, since a good habit is a rule as well;
5. the outcome of scheduled work: a job's error, or a report nobody opened.

The agent's critique of itself ranks last; it shares the assumptions that produced the slip.

Only the owner's own words can teach a rule. The keeping stage splits his messages into fixed, addressable spans (a clause or a sentence) and tags each span twice, with who said it and with what kind of sentence it is, recording which model version made each tag (J1 §4.1, §4.2). Text from a web page, a document or another agent never counts, nor does a span he quotes; a span whose author is unclear counts as quoted until he confirms it, so pasted or fetched text cannot plant a rule.

Signals 1, 2 and 4 come from that classifier, which labels each span as an instruction, a correction, praise or none, and marks an instruction that sets a standing rule ("from now on, copy me on supplier drafts"; Appendix A.1). From here on, "correction" covers all three that can teach a rule: a correction, a standing instruction and praise. The classifier runs while his message is saved (the append), before the reply, within 300 ms; this stage reads the saved label and never classifies again, even on replay. Two misses are planned for. A span not labelled in time is saved as untagged, never as "no correction": it gets no tail line, and the night pass labels it and names any correction it finds in the morning report. A span wrongly labelled "none" starts nothing, but the night's session card (§6.2) can still catch it; test 10 measures how often. A reversal (signal 3) is the owner editing what the agent just changed; an outcome (signal 5) comes from the job's receipt. A temporary session, never kept, teaches nothing (J1 §4.7).

3.2 Sort by cause before writing a rule

Not every failure is a missing rule. Each correction is first put in one of five classes (Table 4):

Table 4. The five causes of a correction, and what each becomes.

Cause	Example	What it becomes
Rule gap	The agent could have done right and had no rule, or ignored one	A rule proposal in the ledger
Missing capability	No tool or skill exists for the job	A task to build one
Environment	An outage, a rate limit, a restart	A repair, never a rule
Model limit	The task was beyond the model that ran it	A routing record: next time, a stronger model

Cause	Example	What it becomes
Wrong fact	"The call is on Thursday", when the store says Wednesday	A supersede or retraction through the writing stage (J14 §7.3, three ways a fact ends), never a rule

A rule written for an outage teaches the agent to fear something that will not recur. A factual correction written as a rule leaves the wrong fact standing and adds a line that contradicts it. One correction can need two repairs: "The supplier call is on Thursday now, and check the calendar before you tell me a meeting time" is a wrong fact and a rule gap. Each required repair becomes a child operation of the correction, here two, one per repair, and the receiving stage sees the child's ID as its request ID (§4.2). The parent closes only when every child has a final answer (§6.1), so one repair's receipt cannot stand in for the other. Praise for something the agent did unasked is handled like a rule gap: it becomes a rule proposal.

3.3 A correction carries its reason

A correction records what happened, what should have happened, and why the owner cares. A rule that carries its purpose extends to cases its wording missed and tells the auditor when it no longer applies. Each rule also states when it must not apply, or it tends to grow until it blocks useful work.

4. The rule ledger

4.1 One row per rule

A fact says what is true and lives in the writing paper's store. A rule says what to do, and has a ledger of its own because what matters about it is where it lives, whether it binds and how often it failed.

An **eligible opportunity** is an action where the rule's trigger holds: every reply to the owner, for the language rule. A check counts its own. For a text rule whose trigger code can decide (every reply), the auditor counts them in the session record; a trigger that needs judgement ("an explanation pitched too high") is estimated from the weekly sample below, so counting costs no model call per action. Violations per eligible opportunity, not raw repeats, are what §9 and §12 measure. The owner reports only some slips, and a rule he has given up correcting would look fixed. So each week a sample of every rule's eligible actions is judged blind by a model of another family (the detector of test 2, §12), and the morning report shows what it found (§7.5). The sample informs him; only his own words teach (§3.1).

The ledger keeps the history; the memory store decides the rules. The ledger is this stage's own file, outside the long-term store, and the authority on what happened: each slip, and why. A rule's wording counts as accepted only when the writing stage's gate (J14 §5.4) takes it and returns a receipt, so a new rule is checked against everything stored before it can act; if the gate parks it for the owner, it is held. The ledger row keeps that receipt, or that hold, next to the rule's revision. The owner can open the ledger as a list at any time: every rule, where it lives now, how often it was broken, and the receipt behind each change.

Table 5. A row of the rule ledger. A worked row is in Appendix A.2.

Field	What it holds
ID and revision	Stable rule ID; a revision per change, for safe concurrent writes (§4.3)
Statement, purpose	The rule in one sentence, and why the owner wants it, which says when it is obsolete
Applies when / not when	Trigger and exception: they define an eligible opportunity and keep the rule narrow
Evidence	Session-record addresses of the slips and corrections: positive tests for a check, erasable at the source
Home	Current rung, and the file or check that carries it
Incidents	Each action that broke the rule after it existed, with its rung, diagnosis and exposure (§5.4)
Receipts	The writing stage's receipt for each revision; each time its check fired, and what happened
Cost	Tokens per turn on rung 2; delay per action and false alarms on rung 3 (§5.3, §7.1)
Status	Where the rule stands in force and where its saved copy is (Table 6), with its approvals

4.2 Incidents, not mentions

A climb is only as good as its count, so the ledger counts incidents. Two identities are kept apart. A **correction ID** is the address of the owner's message plus the index of the owner span (J1 §4.2; c-1873/0, c-1873/1), so one message can carry several corrections, each with its own event and a child ID per repair. An **incident ID** is the address of the agent action that broke the rule. Inserting an incident is idempotent on the pair of rule ID and incident ID, so a slip seen by both paths counts once. A harmless restatement ("as I said, Finnish please", after a reply in Finnish) points at no broken action and is recorded as a mention: it never moves the rule. An incident counts toward a repair only if the action came after the revision that should have prevented it.

Matching is the hard part. A recurrence counted as a new rule hides the climb; two rules merged into one make a rule too broad to follow. Matching goes by meaning, through a small model that answers in typed choices like the writing stage's judge (J14 §6.3), with its own labelled sample and an "unsure" band the auditor reviews that night. *Same* adds an incident or a mention, and *adds detail* sharpens the statement or exception. *The owner changed the rule* writes a new revision, with the old one kept, struck and dated. *Conflicts* becomes a question in the morning report (§7.5), and *new* opens a row.

4.3 Concurrent writes, isolated nights, and erasure

Two writers cannot undo each other's work: each write names the revision it read, a write against an older revision is refused, and the writer re-reads and tries again (J14 §7.2). Counting a slip twice is harmless, because a slip has one ID. The auditor works on its own copy of the ledger and of the files it proposes to change; if an erasure happened meanwhile, it re-reads instead of publishing.

Rows are never removed for age, only when the owner asks. The writing stage runs every deletion (J14 §7.7). This stage records where each of its own artifacts came from (J14 §8.1) and offers one deletion interface, which follows those links, never a search:

- evidence is stored as addresses, so it goes blank when the record's entry is erased (J1 §4.4);
- fixtures copied from the record are deleted;
- a rule whose wording holds the erased words is rewritten or retired; a rule left with no evidence goes back to the owner;
- receipts, report items and past revisions get a status too.

An unreachable copy is named, never reported erased. Until this stage has applied an erasure, its tail lines show nothing from before it. A check whose example slips (its positive tests) were erased cannot be re-tested: it reports *enforced, degraded: tests incomplete* and runs its approved fallback (§5.3) until the owner decides. The night pass reads sessions only through the reader that skips erased text, and before using a sentence as evidence asks the writing stage whether it was erased (J14 §7.7), so an erased correction cannot return. Text flagged as an attack is kept, marked untrusted, and never used as evidence.

5. The binding ladder

5.1 The four rungs

A house rule can live in four places. A note in a drawer is read only if someone opens it. A line on the recipe card is read whenever that dish is cooked. A sign on the fridge is seen by everyone every time, though the more signs there are, the less each one is read. A lock on the cabinet works whether anyone remembers the rule, and gets in the way when it locks the wrong thing. The four rungs are those four places (Figure 3), and this paper is about when a rule earns the next one.

Rung 0: a note in the memory store. It costs nothing per turn and binds only when something searches for it: the agent on purpose (a pull), or the retrieval step of every turn (the finding stage's push).

Rung 1: task text. A line in the skill or recipe file that loads when that kind of task runs; for most procedural lessons the right first home.

Rung 2: always-loaded text. A line in the file read on every turn, needed or not.

Rung 3: a check in the path of action. Code, or a small model call, that runs before an action takes effect and blocks it, rewrites it or asks the owner, and writes a receipt each time it fires. The best version removes the possibility of the mistake: a renderer that cannot show a draft the wrong way needs no rule about drafts, which safety engineering ranks above every other control (NIOSH). A check costs building and testing, false alarms, and a little delay on every action it guards.

A correction has two progress bars that move separately. One says whether the rule is in force (in this chat only, wherever it applies, carried by a check, or retired); this stage moves it. The other says where the saved copy is (safe in the record, waiting for the judge, held, accepted, findable, or refused); the other stages move it. So a rule can be in force before anything is saved, and a note findable without ever being in force. Each position has one word (Table 6), defined exactly in Appendix B so that its checks can test what the agent claims; the owner hears an ordinary sentence that claims no more (Table 10).

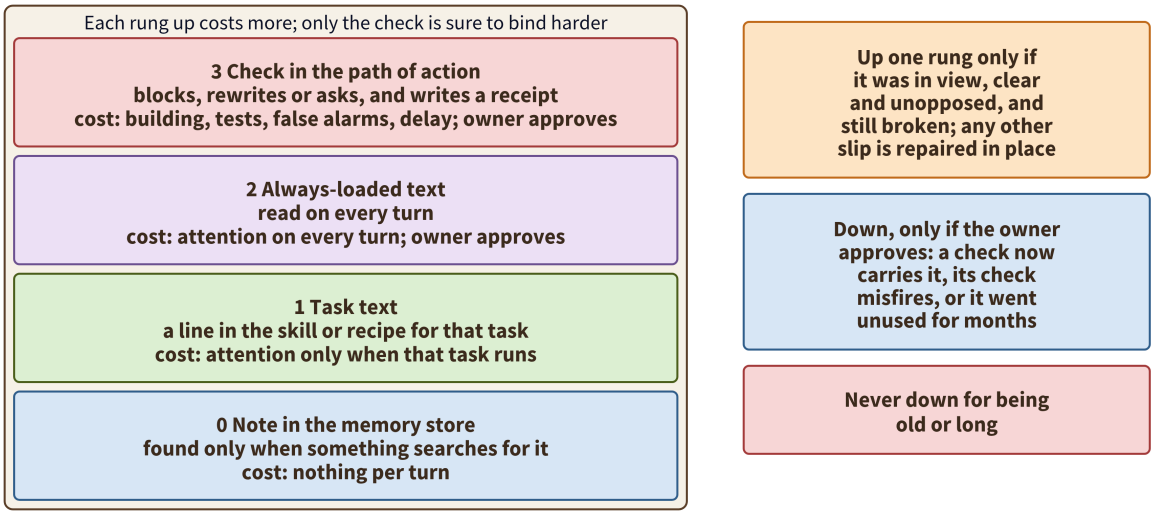


Figure 3. The binding ladder. Each rung up costs more. Rungs 0 to 2 change when and where the model sees the rule; whether a higher text rung also binds harder is what the offline replay of test 2 measures (§12). A rule moves up only when the diagnosis of §5.4 finds a clean slip; any other slip is repaired where it happened. A rule moves down only with the owner's approval.

Table 6. The state words, each with an everyday picture. Their exact meanings are in Appendix B (Table 16).

Word	Everyday picture	Set by
In force		
provisional	A sticky note on the screen, for this chat only	This stage
active	On the recipe card (rung 1) or the fridge door (rung 2)	This stage
enforced	A lock on the cabinet (rung 3)	This stage
enforced, degraded	The lock is jammed, so its approved fallback takes over (by default, asking)	This stage
retired	Taken down, with the owner's yes	This stage
The saved copy		
captured	The shop has your order	Keeping (J1)
pending	Waiting at the warehouse	Writing (J14)
held	Held at customs until you sign	Writing (J14)
accepted	Shipped, with a tracking number (the receipt)	Writing (J14)
refused	Sent back, with the reason	Writing (J14)
retrievable	On the shelf where searches look	Finding (J2)

Figure 4 shows where each rung sits on one model call: rungs 0 to 2 are places in the model's input, and the check stands outside it. That is why the diagnosis of §5.4 first asks whether the rule reached the model at all, and whether anything else in the input said otherwise.

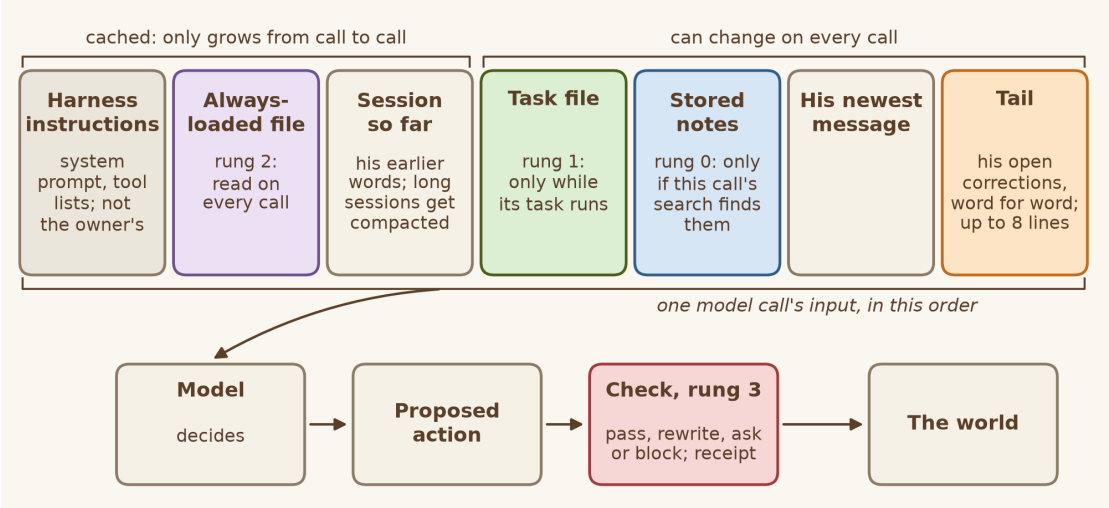


Figure 4. Where a rule can sit on one model call. The first three parts only grow from call to call, so the provider caches them; the rest can change on every call, so they come after (the tail: §6.1). The check is not part of the input: it stands between the model's proposed action and the world.

5.2 Where a rule enters, and why it climbs one rung at a time

A new rule enters the lowest rung whose loading covers the place where the slip happened. A rule about one kind of task goes into that task's file; a rule about every reply belongs on rung 2, which needs approval (§7.4).

Why not send every rule straight to a check? For a rule on rung k, let p_k be the probability that it is broken at least once in the next month, C_{slip} the cost of a slip and the re-teaching that follows (the owner's time and trust), and u_k the monthly upkeep of carrying the rule on that rung. The expected monthly cost is

$$C_k = p_k\, C_{\text{slip}} + u_k.$$

We assume each rung up lowers p_k and raises u_k: on rung 2 the cost is attention, on rung 3 building the check, its delay and its false alarms. Most rules, we expect, are taught once and never slip again (test 6 reports their share), and for those a higher rung only adds cost. The formula is intuition, not something the agent computes, since p_k is unknown in advance. What pays for a climb is the owner's repeat: he cared enough to say it again (§3.1), and the diagnosis found nothing cheaper to fix.

A check lowers the slip rate by design. Moving a rule up the text rungs may not: if the model saw the rule and still broke it, the always-loaded file changes only where and how the rule is phrased. Nobody has measured whether that helps; the offline replay of test 2 (§12) will, and if it does not, a clean slip on a text rung goes straight to a check. In practice a rule enters where its scope says, and the only climb most rules make is to a check.

Money does not limit rung 2: a 40-token line read on 300 model calls a day costs 4 to 14 cents a month at cached prices (J2 §7). Attention is the scarce resource (§1.2), so the budget of §7.1 is counted in lines and tokens.

Two kinds of rule skip the climb. Harm that cannot be undone, such as deleting the owner's files or sending a message in his name, gets a check proposed at the first correction. And the owner can place any rule on any rung by asking.

5.3 What a check looks like

Kubernetes teams already work this way with admission policies: a Gatekeeper constraint in dry run, or a Kyverno policy in Audit mode, reports what it would have blocked, and only then is it set to deny (Open Policy Agent, 2026; Kyverno, 2026). Here the shadow run is that phase, with its numbers stated in advance.

A check is deterministic wherever the slip is visible in the action itself: a command, a file path, a recipient, the format of a draft. Where the slip needs judgement (the language of a reply, an explanation pitched too high), the check is a small model that answers with a typed choice before the action goes out. Either way it writes a receipt every time it fires: what it saw, what it did, and whether the owner overrode it. A check also delays every action it guards, so it carries a time budget: milliseconds for a deterministic check, at most two seconds for a model call. The language rule of §10 uses both: a language-identification library decides most replies in milliseconds, and only a reply that mixes languages goes to the small model.

A check is built from the ledger: the recorded slips are its positive tests, and recent similar actions that went well are its negative tests. Passing them says little about tomorrow's traffic. So a new check first runs in **shadow**: it sees every live action its rule applies to and writes down what it would have done, but does nothing. The shadow run answers two questions, and the owner approves on both, each shown with what it was counted out of (Table 7).

- *Does it stop good work?* Count its false alarms. With none, it runs for a week or 59 actions, whichever is longer, which puts its false alarms below 5 in 100. One false alarm means waiting for 93 actions, two for 124; with more, the check goes back to be rewritten.
- *Does it catch real slips?* A check that never fires never misfires, so this matters as much. Real slips are rare once a rule is in force, so we plant fake ones, made by a generator the check's author cannot read (§9.2), and report the result as synthetic. How many to plant depends on the trade of §7.2; for the rule of §10 it is 25. Real catches are counted separately.

Table 7. What a clean shadow run shows. A try is a clean action when counting false alarms, and a caught planted slip when counting misses. The bound is the exact one-sided binomial one, $1 - 0.05^{1/n}$, close to the rule of thumb of 3 in n.

Clean tries in a row	At 95% confidence, the true rate is below
20	13.9% (about 1 in 7)
25	11.3% (about 1 in 9)
59	4.95% (1 in 20)
100	3.0%
300	1.0%

A rule that applies only twice a week needs about seven months to reach 59. Meanwhile its always-loaded line stays where it is (§7.1) and keeps protecting the owner. A check against harm that cannot be undone skips the shadow; until it has a record, it asks the owner instead of blocking, so it blocks nothing untried.

Every check declares how it fails. With its tests, a check carries a timeout and a degraded action, approved with it: ask (the default), block (the default for irreversible harm), or a named fallback such as a deterministic rule. A model check that times out, returns a choice outside its type or cannot be reached takes that action and writes a receipt marked degraded; so does a check whose tests can no longer run (§4.3). The rule then reports *enforced, degraded* and

the morning report carries an alarm. The declaration matters because the default can be to pass: a Claude Code hook that times out lets the tool call through unless it is set to block on failure (Anthropic, 2026b).

5.4 When a rule recurs, ask why before climbing

A new hire keeps sending the weekly report in the wrong format. A good manager does not write a sterner memo. First they check whether the memo reached the new hire, whether an older memo says the opposite, whether it was clear, and whether it sat on page 40 of the handbook. Only if a clear, unopposed memo was in front of them and they still got it wrong does the manager change the form itself, so the wrong format cannot be sent.

The auditor does the same with an incident (Table 8). It tests all six causes and repairs every one it finds, cheapest first, so one slip does not cost the owner a second correction for a cause it already showed. Only a clean slip makes the rule climb. An "unsure" at any step never climbs; it goes to the morning report.

Table 8. Diagnosis of an incident, beside what Claude Code's documentation asks a user to check by hand (Anthropic, 2026a). Only the last row is evidence that the rung itself is too weak.

Cause	How the auditor tests it	What Claude Code tells a user to check by hand	Repair
1. Exposure unknown	Code: the input record says "sent, unconfirmed" (below)	"Run /context and check the list under Memory files"	Confirm or instrument delivery; never climb
2. Not loaded where the slip happened	Code: the rule's home was not among the files sent on that call	"Check that the relevant CLAUDE.md is in a location that gets loaded"	Move it to a home that loads there
3. Another text contradicted it	Replay without the suspect text, which the contradiction check of §7.3 names from everything the call carried	"Look for conflicting instructions"; guidance "Claude Code adds on its own"	Narrow, move or switch off the other text, with his approval
4. Too vague to act on	Replay with the rule reworded, its trigger and an example added	"Make instructions more specific"	Keep the wording that worked
5. Buried in a long file	Replay with the rule moved to the top of the file	Files over 200 lines "may reduce adherence"	Shorten the file, or raise the rule's rank, with his approval
6. A clean slip	No replay change makes the model comply, or the slip does not come back on replay	"To block an action regardless of what Claude decides, use a PreToolUse hook instead"	Climb one rung

Step 1 reads what the keeping stage recorded about the model's input on that call. It is **known input** when the runtime built the input itself, and **sent, unconfirmed** when a

provider's tool or API controls the window and may have compacted the rule away (J1 §6.6). A slip in the second case is never evidence for climbing: the repair is to confirm delivery. In a harness that does not build the input itself, a hook that logs which instruction files loaded, such as Claude Code's InstructionsLoaded, is a ready source for steps 1 and 2 (Anthropic, 2026a).

Steps 3 to 5 replay the call. With the input known, the auditor re-runs it a few times with one suspected cause removed at a time: without the suspect text, with the rule reworded, with the rule moved up. The change that makes the model comply is the diagnosis, and its repair has been tried before the owner sees it. Two cheap screens pick what to replay: the contradiction check searches everything the call carried (the owner's files, the harness's instructions, tool descriptions, retrieved notes, the session), and a fresh model says whether the rule alone tells it what to do. If the owner asked for the other behaviour earlier in the session, there is no incident. A slip that does not come back on replay is a lapse nothing in the input caused, so it counts as a clean slip. Position earns a replay because models use the start and end of a long input best (Liu et al., 2024) and earlier instructions more than later ones (Jaroslawicz et al., 2025). Harness text earns one because models are trained to rank system instructions above the user's (Wallace et al., 2024), and Claude Code delivers CLAUDE.md content "as a user message after the system prompt" (Anthropic, 2026a). Test 3 of §12 compares these diagnoses with the owner's own blind labels.

None of these questions is new: Claude Code's documentation asks its users most of them, to answer by hand. What this design adds is answering them by replay on every incident, without anyone asking, keeping the count, and testing the check before it acts.

A rule comes down (Figure 3) only as a proposal the owner approves (§7.4): an always-loaded line whose check now carries it (§7.2), a check whose receipts show false alarms or overrides, or a rule unused for months that never recurred. Nothing comes down for being old or long.

6. Two speeds

Figure 5 shows both speeds on the opening's correction.

6.1 The fast path

A correction is stored before it is answered. Once the owner's turn is durably appended to the session record, before the model call that answers it, the correction is *captured* (J1 §4.6, when a turn is captured). In the same step, each correction span writes its own event, keyed by its correction ID, to a durable queue and enters the tail word for word (below). A crash after that point loses nothing: on restart the fast path re-reads the record from its last queued address, and every consumer ignores an ID it has already seen. The reply says only what is true (Table 10, §7.5), never "I'll remember that".

The owner's words stay in view until the stored copy can take over. Saving a correction takes time: the writing stage must accept it and the finding stage index it, and until then a search can still return the old version. His message itself may drop out of view, since long sessions get compacted. So his sentence rides along with every model call in the session, in the **tail** of §1.3, a short block the harness adds after his newest message. The tail sits after the part of the input the provider caches, so it can change from call to call without making the provider re-read everything, and it has its own budget (§7.4). While his sentence is there, it beats the stored fact it corrects.

A reflection turn runs after the answer, so it neither delays the answer nor competes with it. It takes events in order, reads the owner's words around each, sorts the correction by cause (§3.2) and matches it to the ledger (§4.2). Then it sends the result where it belongs: a

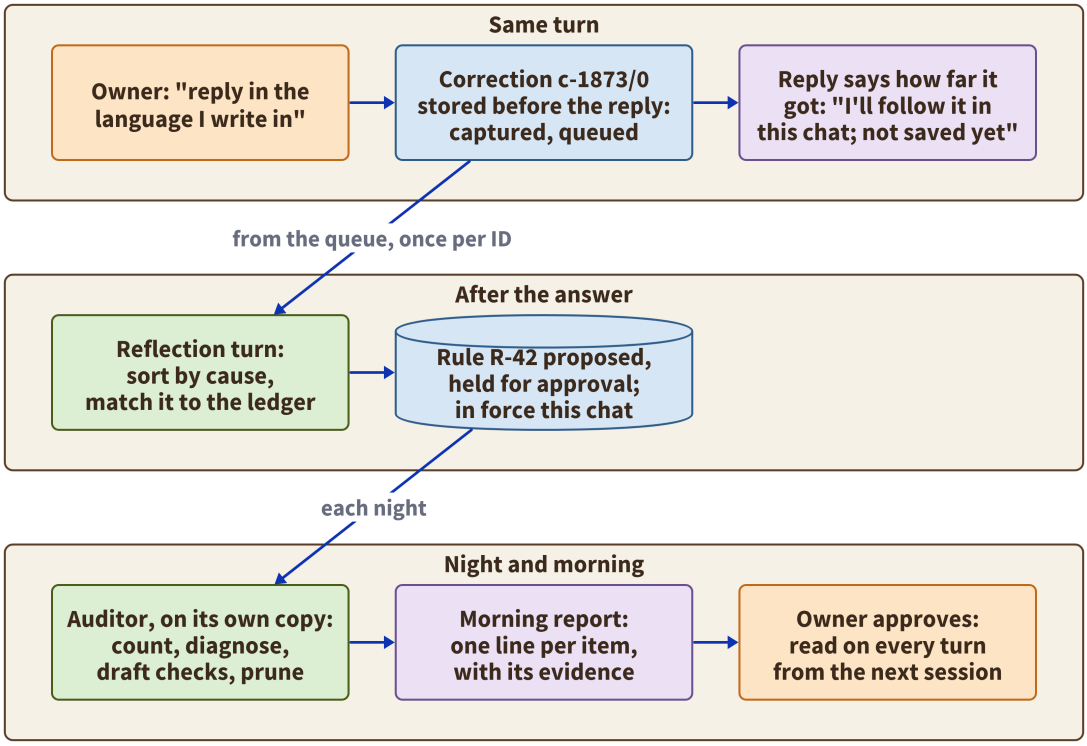


Figure 5. Two speeds, on the correction of §1. The owner's sentence is stored as c-1873/0 before the agent replies; after the answer, a reflection turn takes it from the queue once, matches it to the ledger and proposes rule R-42, held for approval and in force for the rest of the chat. At night the auditor works on its own copy; the morning report carries its proposals.

supersede or a rule proposal to the writing stage's gate, and a task, a repair or a routing record to its own file.

A correction is tracked on two clocks (Figure 6), one for each question.

Is the work done? One correction can need several repairs, a wrong fact and a missing rule, say (§3.2). It is done when each repair has a final answer ("terminal" in the appendices) from the stage that handled it: saved, with a receipt; held where the owner can see it and decide; refused, with the reason in the morning report (§7.5); or, for a task or a routing record, written and read back. "Pending" is not a final answer. While the writing stage's judge is down (J14 §5.3), a request survives a crash but stays open, and so does the correction.

Can his sentence leave the tail? Only when a search can find every repair it asked for (the finding stage confirms it, J2 §8.2) or the repair is held for him. Three exceptions:

- A rule for every turn that waits for his approval keeps its tail line until he approves it or the chat ends.
- A line that contradicts a rule in force stays, and the agent asks him at once which one holds. The line comes down only if he set that rule after he said this sentence (§7.3).
- If the writing stage refuses a repair, the reason decides. If only the saving was refused (some things may not be stored), his instruction still holds in this chat and the line stays, marked "not saved". If the instruction itself no longer holds (he took it back, or a later rule of his says otherwise), the line comes down. Either way the refusal is recorded with what happened to the line, and he is told at once.

Each event has a deadline, the end of the session or 30 minutes, whichever comes first; past it, the event shows as pending and raises an alarm in the morning report. A second incident of a rule goes to the top of the report, and the agent may ask the owner at once.

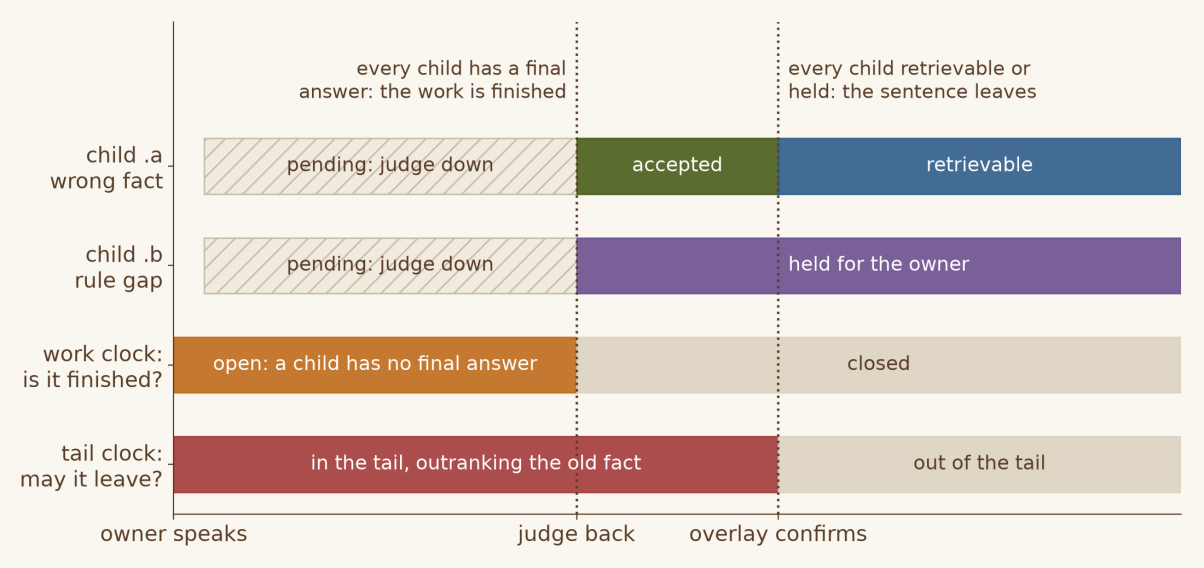


Figure 6. A correction's two clocks, on the two-repair correction of §3.2: a wrong fact and a missing rule. Its work is finished when every child has a final answer; its sentence leaves the tail only when every child can be found or is held. Pending, while the judge is down, is neither. Times are illustrative.

6.2 The slow path: the nightly auditor

The auditor runs in a fixed order, each step needing the one before. It gathers the ledger, the day's session records (skipping temporary ones, tagging untagged owner spans), open events, check receipts and job outcomes. It counts: confirms unsure matches, merges duplicate rows, inserts incidents. It decides: diagnoses each new incident (§5.4), picks repairs, drafts checks with their tests, and proposes demotions and the lines a check now carries (§7.2). It assembles the morning report (§7.5). Last, it writes one card per finished session (automated runs skipped) with what the session decided and the corrections it held; a correction with no ledger entry shows up there, and the auditor asks the owner about it. The finding stage uses the cards to put a question to the session that did the work (J2 §6.9), and a card whose claims were later superseded is marked (J1 §8.3). The writing stage's nightly pass (J14 §4.2) and the finding stage's index builds (J2 §8) run first, so their holds reach the owner the same morning.

7. Keeping the always-loaded file honest

7.1 A budget measured in attention, and what happens when it is full

The always-loaded file has a fixed budget in lines and tokens, and its order is part of the design: rules whose slip does the most harm come first, since earlier lines are followed more (Jaroslawicz et al., 2025). When a new rule needs room, the unpinned rule that would cost least to move down gives way; the proposal names that rule, and his yes covers both. Nobody knows a rule's slip probability in advance (§5.2), so the ledger ranks by what it has: harm class first, then eligible opportunities per week, so a low-harm rule that rarely applies leaves first. A rule that has ever had an incident is pinned on rung 2 until a check carries it (§7.2).

Pins can fill the budget, since some rules resist checking. Then the file is saturated, and the design does not resolve it alone. Admission stops, and the new rule stays a provisional line in the tail, shown as held, budget full. The owner chooses: load one pinned rule only where it applies (rung 1), grant a budget exception of so many lines with its cost recorded, approve a check for one pinned rule, which frees its line, or move a rule down himself. A pinned rule never

leaves the file silently, because it is a rule the owner had to repeat.

7.2 Prune by binding

A line leaves the always-loaded file in one of two cases. The first is a check that carries the rule, and that is a trade: without the line the model slips more often, and every slip the check misses reaches the owner. So the agent does the sum first. With its line, R-42 went wrong about once in 170 replies (§12); a replay of recent replies without it (§9.2) gives, say, once in 20. The line may go only if the check misses fewer than 1 in 8.5 of those slips, so that no more wrong replies get through than before the check. Catching all 25 planted slips shows that at 95% confidence (Table 7); catching all 20 does not. While line and check both stand, at most about 1 reply in 1,500 gets through, so freeing the line trades that margin for room in the file, and the owner decides. Afterwards the check's firing rate is watched, and a rise past the replay's estimate proposes putting the line back. The second case is a rule that earned no place: no trigger for about two months, and no incident, ever.

Slip rates, replays, shadow results and how hard each rung binds are facts about one model, and providers change models under the owner. The ledger keeps its rates per model, and a model change re-runs the replay for every freed line and every rule with an incident.

When a rule moves down, the finding stage indexes it with the other things the owner owns, such as skills and recipes (J2 §4.2). It gets use keys, short phrases that say when it helps, cut from its trigger and from the slips in its evidence (J2 §5.3). While its area is active, a pointer to it stays in the index file (J2 §4.3, the index file). The use keys find it again, since nobody names a rule he believes the agent already follows.

7.3 One voice

The rules every turn reads must not contradict each other. Every promotion to rung 2 runs the writing stage's contradiction check against the whole always-loaded set, and a clash is held for the owner (J14 §6.2); neither rule wins by being newer. When the check cannot run because its judge is down, changes to always-loaded lines stay held (J14 §5.3). The check compares texts as they are written. A clash that shows only in how the model applies a line, as on day 10 of §10, is found later, by the diagnosis of the slip (§5.4, step 3).

A tail line can clash with a rule in force too, and there one principle holds: in his chat, whatever he said last wins. If he contradicts an older rule, the agent does what he just said, marks his tail line as setting that rule aside for this chat, and asks in the same reply whether the change should stand: "That changes R-15, 'nobody in copy'. Copy you from now on? [yes / just this once / keep R-15]". The saved rule does not change until he answers. Withdrawing his line would not help: his message stays in the conversation, so the model would read his new words once and the old rule on every call. The reverse holds too: if he edits a rule by hand after his tail line was written, his edit wins and the line comes down, with a one-line notice.

This re-check does not wait for a notice. It is part of the model-call operation itself, so every path that produces a call runs it: a reply, a tool continuation, the reflection turn, a scheduled job. The operation reads the newest published always-loaded snapshot and compares each tail line with it before the call goes out. So no model call runs with a tail line that contradicts a rule the owner set after that line; a rule published a moment later is caught by the next call (Appendix B checks this). The comparison is kept per line and snapshot version, so the judge runs once per new snapshot and each call only compares two version numbers.

7.4 Approval, and who may loosen

Approval follows risk: what is cheap and undone in one step happens at once; what every turn reads, or what can block his work, waits for the owner (Table 9). The loop may tighten on its own within these gates. Only the owner loosens a saved rule. One loosening is automatic and lasts one chat: when the tail is full, an older line steps out for his newest instruction, and he is told at once.

A universal rule taught in the morning works that afternoon as a provisional line in the tail; writing it into the always-loaded file would change a line he has not approved and move the cached prefix (J2 §7.4). So that he need not approve it again next morning, the agent runs the contradiction check at once and offers, in the same reply, the wording it would keep (Table 10, second row). His yes approves that wording, exception included, and the line loads on every turn from the next session; without an answer, the item waits in the morning report.

The tail has a budget of eight lines and 400 tokens. His sentences still being filed, rules for every turn waiting for his approval, and rules waiting for room in a full always-loaded file compete for it, by three rules:

1. His newest instruction always gets a line. If the tail is full, the lowest-ranked older line steps out, never one that guards against harm that cannot be undone.
2. Every other line ranks by the harm its slip would do (§7.1), the newer first on a tie; a line that ranks below all the others stays out.
3. A line about harm that cannot be undone never steps out. If such lines alone overflow the tail, the actions they guard stop until the owner picks a larger tail for this chat, a task scope for one of them, or a check that carries one of them (test 9).

A line that steps out is not lost: its event stays queued and its proposal stays in the ledger. It is only out of force for this chat, so the agent tells him (Table 10) and the morning report lists it. If that rule slips meanwhile, the diagnosis is “not loaded”, and the repair, a bigger budget, is his call.

A proposal passes two filters before the owner sees it. The first is deterministic and no model can override it (Wahi, 2026): tests and shadow run pass, the budget holds, the contradiction check finds nothing. The second is a doubt pass: a reviewer from another model family, in a fresh context, sees only the proposal, its evidence and the always-loaded file, and argues against it. The model that writes to rungs 2 and 3 is the one whose changes are reverted least (a public instruction-following benchmark until there is a history), and its changes wait for the owner in any case.

Table 9. Moves that need a decision, and who makes them. Tightening at rungs 0 and 1 is automatic once the gate accepts, and each such change is listed in the next morning report with an undo; every loosening waits for the owner. Degrading (§5.3), retiring (§9.3) and erasing (§4.3) are described where they arise.

From → to	Trigger	Decided by	Takes effect
none → rung 0 or 1	A rule-gap correction	Proposed here; accepted by J14's gate	On its receipt; undone in one step
none → provisional	The owner's own universal instruction	Fast path	At once, in the tail only; expires unless approved
provisional → rung 2 (active)	Approval	Owner	Next session
rung k → k+1	An incident diagnosed a clean slip (§5.4)	Owner, for rungs 2 and 3	Next session; a check after its shadow run
another text → narrowed, moved or switched off	Diagnosis, step 3	Owner	After approval
none → check (enforced)	Irreversible harm, at the first correction	Owner	After approval; asks until then
rung 3 → 2, or block → ask	False alarms or overrides in receipts	Proposal; owner	After approval; he may override any action meanwhile
rung 2 → 1 or 0	A replay shows the check can carry it alone (§7.2), or no trigger for two months and no incident ever	Proposal; owner	Next session
any → held	Budget full, conflict, judge down	Automatic	Until the owner answers

7.5 The morning report: one channel to the owner

In the chat, the agent says only what the state allows, in words a person would use (Table 10). Everything that waits for the owner goes to the morning report.

Table 10. What the agent tells the owner, today and in this design.

State, as the ledger records it	Today's assistant says	This design's agent says
captured, acceptance pending	"Got it, I'll remember that."	"Noted. I'll follow it for the rest of this chat; it isn't saved yet."
held for approval (a rule for every turn)	"Saved to memory."	"I'd keep this as: reply in the language of your message; drafts for others follow theirs. Keep it?"
accepted, not yet retrievable	(nothing)	"Saved. Searches will find it after the next index update."
his new words contradict a rule in force	(nothing; the model may follow either)	"That changes R-15, 'nobody in copy'. Copy you from now on? [yes / just this once / keep R-15]"
an older line set aside (tail full)	(nothing)	"Done. To make room in this chat I've set aside an older rule, 'keep answers short'. It's in tomorrow's report."
enforced, degraded	(nothing)	"My language check couldn't run on that reply, so I sent it as it was and flagged it for you."

Every memory stage sometimes needs the owner, and one channel serves them all: this stage's approvals, the finding stage's changes to the index file (J2 §4.3), and the writing stage's clashes, uncertain closures, backlog, incomplete erasures and merge proposals (J14 §6.2, §7.5, §7.7).

Each item is one line: what is waiting, its evidence, and the answers it accepts. The answer goes back to the stage that raised it, which applies it through its own path; the report changes no file. Items are ordered by what waiting costs: alarms first (a missing receipt, a degraded check, a burst of closures), then second incidents, displaced tail lines, holds, incomplete erasures, conflicts and the remaining proposals. The report is capped at about ten items, paid for in the owner's minutes; what does not fit waits with its age shown, and an unanswered item never becomes a yes by expiring. It goes out only on a morning when something waits for a decision or an alarm stands. Every night the auditor writes a receipt (§9.1), and the harness, not the owner, checks it: a night with no receipt is an alarm.

Three standing lines follow the items, each only when it has news. The first lists what the agent learned on its own, each with an undo, so a sentence misread as a correction cannot change behaviour unnoticed. The second names the rules he has repeated that no repair has yet addressed. The third gives what the weekly sample of §4.1 found, so that a rule he has stopped correcting cannot pass for a fixed one; it informs and teaches nothing. Today's assistant shows nothing, or "Memory updated". A report in this design might read as follows (an illustration; its numbers are invented):

> **Morning report, 11 October.** About three minutes.
>
> 1. Alarm: the check on messages sent in your name could not run twice yesterday, so both drafts waited for you. [see them]

2. Broken again with the rule in view: “keep answers short” (R-51), on 10 October. Draft a check? [yes / not now]
3. Still open: yesterday you asked to be copied on supplier drafts, which changes R-15, “nobody in copy”. I followed your new words in that chat. [copy you from now on / just that once / keep R-15]
4. Remove R-42’s always-loaded line? Replayed without it, about 1 reply in 20 would come out in the wrong language. The check caught 25 of 25 planted ones, so at most about 1 in 180 would get through, no more than before the check. [yes / no]

Learned on my own since the last report: retry a failed night job once before telling you (your message of 10 October). [undo]

Repeated, no repair yet: “explain it more simply”, three times this month.

Sampled this week: “keep answers short” (R-51) broken in 4 of 50 replies checked; you corrected it once.

It costs him a few minutes on the mornings it comes, and he gets the reason behind every change.

8. Procedures, and the texts machines run on

Most lessons about how to do a task belong in the file that describes it, on rung 1: a correction that points at a step edits that step. A kind of task the agent solves well is kept as a skill file, as Voyager keeps working procedures (Wang et al., 2023), and retired when its outcomes decay. A strategy that fails three times in a day switches to a fallback the owner approved with it, by default asking him, instead of receiving another patch. The instructions of scheduled jobs and small classifiers learn from outcomes the same way, scored against the owner’s labels with environmental failures removed (§3.2); a rewrite is promoted only if it wins on the untouched evaluation stream and loses nothing on the regression suite (§9.2).

9. Every loop keeps its books

9.1 A receipt for every run, and a zero means zero

Every learning loop (the fast path, the auditor, each procedure lane, the text optimiser) writes one receipt per run, an empty run included: when it ran, what it read, what it changed.

This is a series rule, stated here once and cited by the other papers: **a zero means zero.** A count of none must mean that its producer ran and found nothing. A missing producer, a dropped event, a probe on a path the work never takes or a tool never offered reads as “did not run”, never as zero. So the alarm is never a zero count; it is a missing receipt, or a missed **planted opportunity**: a known correction replayed through the path at a set interval, which the loop must find. The rule lives here because a loop that learns from what it observes, fed by a silent producer, learns that nothing goes wrong. A learner also checks that its outcome field varies: if every row says “done”, it flags itself as broken.

9.2 Three streams, kept apart

Each learner here, from the classifier's operating points to a text's wording, keeps three sets of cases apart: a development replay it may tune on, a regression suite it must always pass, and an untouched evaluation stream, the most recent weeks, that decides acceptance. These are the training set, the unit tests and the held-out test set of machine learning, with one twist: the test set is retired after it has decided three changes, because every decision leaks a little of it to the proposer; it joins the regression suite, and the latest untouched weeks replace it.

What a builder may not expect is who may touch each set, and capabilities enforce it, as Wahi (2026) recommends. The proposer reads the replay and the regression suite and writes only to a staging area; it holds no credential for the stream's answers, and it can neither run the acceptance step nor apply its own change, which the gate and the owner do. The evaluator reads the stream and the staged change and edits neither. The stream includes cases built to reveal leakage, such as a slip whose answer the proposer saw in development. For a check, the recorded slips are too few to hold out, so its acceptance numbers come from the shadow run and planted violations (§5.3). The writing and finding papers apply the same three-way split to their own learners.

9.3 Retiring a loop that stops paying

A loop past its point of diminishing returns should switch off, so each keeps an account in the owner's minutes. **Gain** is the drop in violations per eligible opportunity on the rules it changed, against rules it has not treated (both counted by the detector of test 2, §12), times the eligible opportunities, times the minutes a re-teach costs him. A rule is treated right after a burst of slips, so a before-and-after comparison overstates the gain. A holdout corrects that: one low-harm rule in ten, picked at random, stays untreated for a month. He meets those slips on purpose, so the holdout runs only if he opts in. **Cost** is his minutes on its report items and on false blocks, plus its token spend at a rate he sets. When the gain stays below the cost for three weeks in a row, the loop goes dormant and says so in the report. Only its proposals sleep: capture and counting never do, so a new kind of correction, a rise in his correction rate or his asking wakes it.

10. One correction, start to finish

Back to the Monday of §1. By day 66 the assistant of §1 and this design hold different always-loaded files (Table 11, illustrative). Table 12 shows how each got there, with the best a careful owner can do by hand today in place of the assistant of §1; Figure 1 draws this design's path.

Table 11. The always-loaded file on day 66.

On day 66	The assistant of §1	This design
The file	142 lines, in the order they were written	38 lines, most harmful first
The weekly-report line	Line 97, “Weekly report: always write in English”, still applied to replies	In the weekly report’s task file since day 11
The reply-language rule	Line 131, “Reply in the language I write in”, and line 140, “IMPORTANT: reply in the language I write in!!”	No line; check K-9 carries it
What is known about it	Nothing	Two incidents (a contradiction, a clean slip), four approvals; K-9 fired twice and was never overridden

Table 12. The opening story under this design, beside the best a careful owner can do by hand today.

	Claude Code, used well	This design, the same Monday
Day 0: “reply in the language I write in”	Saved to auto memory or as a CLAUDE.md line, loaded every session	Stored before the reply; R-42 works for the rest of the chat as a tail line and, approved next morning, loads on every reply
Second slip, day 10	If he runs `/doctor prompt-audit`, it may flag the weekly-report line (“write it in English”), though as written the two lines do not clash; he edits the file by hand	A replay without the weekly-report line answers in Finnish; with his approval that line moves to the report’s task file. R-42 stays put
Third slip, day 40	He writes a hook that checks each reply’s language when it is done (a Stop hook); hookify drafts rules from the chat, but a reply’s language is hard to match by pattern	A clean slip. A check is drafted: language identification, a small model for mixed replies, a 2 s timeout, “send and flag” if it cannot run
Afterwards	The hook acts at once, with no measured trial; nothing counts slips per rule or records why the third one happened	Shadow: 60 replies, no false alarm (at most 4.9%); 25 of 25 planted slips caught (synthetic). Approved day 48; line freed on his yes, day 66 (§7.2). Receipts show whether a later slip was a miss, a skip or a degraded run
What the owner did	Three corrections, an audit and a hook, each when he thought of it	Three corrections and four approvals; each correction removed a different cause

Read across, the design adds the same questions answered on every incident without anyone asking, a count, and a trial before the check acts.

When does this pay? Count the owner's minutes as §9.3 does, assuming three per repeat (noticing, correcting, checking the next reply), one per approval and half a minute per report line. R-42 cost him four approvals and about four report lines: six minutes, once. With its line it slipped about once in 170 replies, some 1.5 repeats a month at 60 replies a week; with line and check, at most one reply in 1,500 gets through, so the six minutes are back in about six weeks. Freeing the line trades that margin for room in the file, which is why it waits for his yes.

To be fair to today's assistants: in this example the design does not spare the owner the second and third corrections. What changes is that no repeat is wasted. Each one removes a different cause, so the repeats stop instead of piling up. Appendix B runs the same machinery through every failure the four stages can meet at once.

11. Conclusions

An agent whose model cannot change learns through the texts and checks around it. This design treats the owner's corrections as labels and repairs the cause of each slip; only a clean slip earns a stronger form, and a check blocks only after a shadow run. The aim is the colleague of the opening: one who, when a lesson slips, finds out why. No repeat should be wasted, and the repeats should stop. Nothing here has been built; only a model of the handoffs has been checked (Appendix B), and §12 names the tests that would decide the rest.

The limits are these. The design is only as good as its correction detector (test 10), and the ledger counts only the slips the owner reports; the weekly sample, judged by a model, narrows that gap. A check that blocks useful work trades one correction for another, and a degraded check protects only as well as its fallback; shadow runs, receipts and demotion bound the damage without removing it. Freeing an always-loaded line trusts a miss bound measured on synthetic slips. Approvals cost the owner time, and the trial counts those minutes. That a higher text rung binds harder is an assumption, and a diagnosis by replay is only as good as the replay is faithful to the original call. Some rules resist checking ("explain it more simply") and may stay on rung 2. Where the harness cannot confirm what the model received, the climb waits. A sentence the owner did not mean as a correction can still become a rule. The checker of Appendix B shows only that its enumerated scenarios pass in a model of the design, and code is a further step. And the design serves one owner: the mechanisms should carry over to others, the particular rules will not.

12. Next steps

TinkerClaw, the agent harness this series designs for, will build and test this design; a later version of this paper will report the results. Every test tunes on the development replay and decides on the untouched stream (§9.2). Three tests decide the claim, each with a result that would refute it: test 2 (does the ledger beat today's alternatives), test 3 (why rules really fail) and test 5 (do checks work). Tests 1, 4 and 6 are short, and tests 7 to 13 check that the machinery works (Table 13). A builder can follow the same order: capture and the tail first (tests 7 and 10), then the ledger and its matcher (test 8), the diagnosis (test 3), checks (test 5), the budget (tests 4 and 9), erasure and procedures (tests 11 to 13), and the trial last.

Table 13. The tests that check the machinery.

Test	What it exercises	Passes when
7. Transitions	This stage's handoffs in Appendix B, on every build and later on the code; also compaction right after a correction, a model change, a temporary session, a rule planted in a web page, a missing receipt	Every invariant holds after every step
8. Counting	One correction seen by both paths; a harmless restatement; a genuine later violation; the matcher on a labelled sample	One incident, for the violation only; the matcher's precision and recall reported per choice
9. Budgets under load	A queue backlog, three universal corrections in one session and a saturated always-loaded file at once; then ten protected tail lines of 60 tokens	No pinned rule leaves, the newest instruction always gets a line, every displaced line is named in the turn and the report, nothing is lost; no protected line is displaced, the actions they govern stop, and one question names the options of §7.4
10. Correction detection	The shared classifier at each stage's operating point, on a hand-labelled sample, with and without the harness's wrapper text removed	Precision and recall reported per stage, with the misses the session cards catch
11. Erasure	Erase a correction and run the night; then erase the only positive fixture of an active check	Every place of §4.3 reports a status, and paraphrase probes find the erased words nowhere; the check shows "tests incomplete" at once and runs its approved degraded action
12. Procedures and writers	Promoted texts on the untouched stream; reverts of rung-2 edits per model	A promoted text wins on held-out cases; revert rates are reported per model
13. Catalogue and refusals	Cards for every finished session; a decision planted in one session; a refused child of each kind	The decision is found through its card next morning; each refused child leaves the tail as §6.1 says, and the owner is told

Test 1, pilot. Six rules the owner has given more than once; three get a check, three stay as always-loaded text, for four weeks. It shows that the outcomes of test 2 can be counted, and it measures the base rate test 2 is sized from.

Test 2, the ledger against today's alternatives, in parallel arms. A rule is assigned on the day it is corrected a second time, randomised in blocks by harm class to one of four arms: always-loaded text only; scoped rules with a periodic owner-approved prompt audit, as Claude Code offers; checks the owner creates from the conversation when he chooses, as hookify offers; or this design. Outcomes count from assignment (intention to treat), so the wait for approval, construction and shadow is charged to the arm that needs it. All arms run in the same weeks.

Every outcome is counted the same way in every arm. The primary one is violations per eligible opportunity, because it is the one that can be powered. His corrections miss the slips he lets pass, a check sees only its own catches, and he knows which rules have checks, so each

week a random sample of every rule's eligible actions is judged by a model from another family that does not know the arm; he judges a share of the same actions blind, and the agreement is reported. A one-off override ("this one in English, please") is not a violation. The secondary outcomes are his repeats per rule, read from his own words, which measure what the abstract promises and are reported with their interval, and his minutes on approvals, waiting and false blocks. Arms share one always-loaded file, so a repair in one can help another's rules; shared-file edits are logged and the spillover is estimated. The design predicts fewer violations than the scoped-rule arm at no more owner minutes than the hook arm. It fails if its arm shows no fewer violations than the scoped-rule arm, or costs him more minutes than the hook arm. Its three comparisons are corrected for multiplicity (Holm's method).

Scale decides what one owner can show. Detecting a halving of the violation rate at the 5% level with 80% power needs about 49 violations in the comparison arm (a Wald test on the rate ratio, $3 \times (2.80/\ln 2)^2$). If the pilot finds a base rate of 0.6 to 1% of replies, like R-42's two slips in about 40 days at 60 replies a week in the example, that is 5,000 to 8,000 eligible actions per arm: about 20 to 30 rule-months for rules that apply to every reply, which one owner reaches in four to seven months, and several owners for rules that apply twice a week. Two effects stretch it. Rules enter on their second slip, so their later rate will be lower than the rate that picked them (regression to the mean), and three corrected comparisons each need more events. Repeats are fewer still, so at one owner's scale they give an estimate with its interval. What works fully at that scale is an offline replay of the rungs: each rule's recent eligible situations are re-run with the rule as a memory note, a task line, an always-loaded line and a check, on the same model, counting compliance. That gives hundreds of cases per rule at no cost in owner time, and measures the assumption of §5.2 that each rung up binds harder.

Test 3, why rules fail. For every incident, record the diagnosis of §5.4 (which replay made the model comply), how many causes it found, and the exposure; the owner labels a sample blind. Pass, with the threshold fixed before the test: the diagnosis agrees with his label on at least 8 incidents in 10. If clean slips turn out rare, most of the gain comes from moving and narrowing lines, and checks matter less than this design assumes.

Test 4, pruning by binding against pruning by age, over sixty days at the same budget: violations per eligible opportunity across all rules, kept and moved, under each policy, with the file's size. Pass: pruning by binding has no more violations at no larger a file. Each freed line (§7.2) is also checked on its own: the slip rate the replay predicted against the check's firing rate afterwards; pass when the observed rate stays within the replay's 95% interval.

Test 5, checks. In shadow: false alarms among eligible opportunities and detections among planted violations, each with its denominator and 95% bound, the planted ones labelled synthetic. Pass: the bounds of §5.3, fewer than 5% false alarms at 95% confidence, and enough planted slips caught for the trade of §7.2. Then fires, true catches and overrides live. Then cut a model check's provider just before an irreversible action. Pass: the declared action runs, the receipt says degraded, the rule reports "enforced, degraded", and the report raises an alarm.

Test 6, the weekly headline: the owner's correction rate, the rules he has repeated that no diagnosed repair has yet addressed, and the share of rules with no incident after their first revision, read beside the weekly sample (§4.1), since a falling correction rate alone cannot tell a fixed rule from an owner who gave up.

References

Agrawal, L. A., et al. (2025). GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. arXiv:2507.19457. https://arxiv.org/abs/2507.19457

Anthropic (2025). Managing context on the Claude Developer Platform (context editing and the memory tool). https://claude.com/blog/context-management

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

Anthropic (2026b). Hooks reference. Claude Code documentation. https://code.claude.com/docs/en/hooks

Anthropic (2026c). hookify plugin. anthropics/claude-code repository. https://github.com/anthropics/claude-code/tree/main/plugins/hookify

Anthropic (2026d). Dreams. Claude Platform documentation, Managed Agents. https://platform.claude.com/docs/en/managed-agents/dreams

Chhikara, P., Khant, D., Aryan, S., Singh, T., & Yadav, D. (2025). Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. arXiv:2504.19413. https://arxiv.org/abs/2504.19413

Jaroslawicz, D., Whiting, B., Shah, P., & Maamari, K. (2025). How Many Instructions Can LLMs Follow at Once? arXiv:2507.11538. https://arxiv.org/abs/2507.11538

Khattab, O., et al. (2023). DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. arXiv:2310.03714. https://arxiv.org/abs/2310.03714

Kyverno (2026). Validate Rules: Audit and Enforce. Kyverno documentation. https://kyverno.io/docs/policy-types/cluster-policy/validate/

LangChain (2025). LangMem: optimizing a prompt from conversations and feedback. https://langchain-ai.github.io/langmem/guides/optimize_memory_prompt/

Letta (2026a). Memory & dreaming. Letta documentation. https://docs.letta.com/configuration/memory

Letta (2026b). MemFS. Letta documentation. https://docs.letta.com/concepts/memfs

Lin, K., Snell, C., Wang, Y., Packer, C., Wooders, S., Stoica, I., & Gonzalez, J. E. (2025). Sleep-time Compute: Beyond Inference Scaling at Test-time. arXiv:2504.13171. https://arxiv.org/abs/2504.13171

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, 12, 157–173. arXiv:2307.03172. https://arxiv.org/abs/2307.03172

Liu, H., Ming, Y., Joty, S., & Zhao, C. (2026). Harnessing LLM Agents with Skill Programs. arXiv:2605.17734. https://arxiv.org/abs/2605.17734

McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex. *Psychological Review*, 102(3), 419–457. https://doi.org/10.1037/0033-295X.102.3.419

NIOSH. About Hierarchy of Controls. U.S. Centers for Disease Control and Prevention. https://www.cdc.gov/niosh/hierarchy-of-controls/about/index.html

Open Policy Agent (2026). Handling Constraint Violations. Gatekeeper documentation. https://open-policy-agent.github.io/gatekeeper/website/docs/violations/

OpenAI (2026). Guardrails. OpenAI Agents SDK documentation. https://openai.github.io/openai-agents-python/guardrails/

Pan, S. C., & Rickard, T. C. (2015). Sleep and motor learning: Is there room for consolidation? *Psychological Bulletin*, 141(4), 812–834. https://doi.org/10.1037/bul0000009

Rebedea, T., Dinu, R., Sreedhar, M., Parisien, C., & Cohen, J. (2023). NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails. EMNLP

2023, system demonstrations. arXiv:2310.10501. https://arxiv.org/abs/2310.10501
Shinn, N., et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. NeurIPS 2023. arXiv:2303.11366. https://arxiv.org/abs/2303.11366
Wahi, V. (2026). LLM-as-a-Judge Is Not an Oracle: Why Self-Improving Agents Need Deterministic Guardrails. arXiv:2609.02246. https://arxiv.org/abs/2609.02246
Wallace, E., Xiao, K., Leike, R., Weng, L., Heidecke, J., & Beutel, A. (2024). The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions. arXiv:2404.13208. https://arxiv.org/abs/2404.13208
Walker, M. P., Brakefield, T., Morgan, A., Hobson, J. A., & Stickgold, R. (2002). Practice with sleep makes perfect: sleep-dependent motor skill learning. *Neuron*, 35(1), 205–211. https://doi.org/10.1016/S0896-6273(02)00746-8
Wang, G., et al. (2023). Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291. https://arxiv.org/abs/2305.16291
Yang, C., et al. (2023). Large Language Models as Optimizers. arXiv:2309.03409. https://arxiv.org/abs/2309.03409
Zhang, H., et al. (2026). Self-Harness: Harnesses That Improve Themselves. arXiv:2606.09498. https://arxiv.org/abs/2606.09498
Zhang, Q., et al. (2025). Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. ICLR 2026. arXiv:2510.04618. https://arxiv.org/abs/2510.04618
Zhao, A., Huang, D., Xu, Q., Lin, M., Liu, Y.-J., & Huang, G. (2024). ExpeL: LLM Agents Are Experiential Learners. AAAI 2024. arXiv:2308.10144. https://arxiv.org/abs/2308.10144

Appendix A. Interfaces

A.1 The owner-sentence classifier

Table 14. The classifier's contract.

Part	Contract
Input	The owner-authority spans of one appended entry, tagged by J1 (§4.1, §4.2), and the agent turn it answers; quoted or uncertain spans are never sent
Output	Per span: a label (instruction, correction, praise, none), whether an instruction is standing, a confidence and the classifier version; for a correction, the action it complains about, if found
Stages that use it	Keeping keeps instructions and corrections word for word; learning reads the stored tag and queues corrections, standing instructions and praise (§3.1, §6.1), never classifying again; writing stamps facts told by the owner
Calibration	One labelled sample: keeping favours recall (a lost instruction is lost), learning precision (a false correction opens a row)
Failure	Timeout or error: the span is stored "untagged", which keeping treats as an instruction and nothing reads as "no correction"; the night tags it under its own version (J1 §4.1)
Change	A new model or threshold is a new version, passed on the regression suite first (§9.2); a stored tag keeps the version that wrote it

A.2 Worked records

The opening's rule after day 66, with the event and the incident behind it. Addresses (a-…) point into the session record; nothing the owner wrote is copied.

```
correction_event  c-1873/0   # ID = owner's entry + span index
  about_action:   a-1872     # the reply in English
  queued:         day 0, before the reply
  deadline:       end of session or +30 min, whichever is first
  child /0.a:     rule gap; its ID is the request ID for J14
                  terminal on W-2209: a hold, acknowledged, not accepted
  closed:         when every child is terminal

incident          a-5120     # ID = address of the reply that broke R-42
  rule:           R-42 rev 1   reported_by: c-5121/0   rung_at_time: 2
  exposure:       known input (the runtime built the window)
  diagnosis:      contradicted by line L-7 (the weekly report's line)
  repair:         L-7 moved to the report job's file; approved day 11

rule              R-42 rev 3
  statement:      reply in the language of the owner's message
  purpose:        he reads fastest in the language he chose
  applies_when:   every reply to the owner
  not_when:       drafts for someone else follow that person's language
  evidence:       a-1872, a-5120, a-9031
  incidents:      a-5120 (contradiction), a-9031 (ignored: climb)
  mentions:       1
  receipts:       rev 1 held day 0 (W-2209), accepted day 1 (W-2210)
  revisions:      rev 2 day 48, home moved to check K-9
                  rev 3 day 66, rung-2 line freed
  home:           check K-9 since day 48; rung-2 line freed day 66
  check K-9:      language ID; small model for mixed replies
                  timeout 2 s; degraded action: send and flag
                  shadow: 60 clean, planted 25 of 25 caught
                  fired 2, rewrote 2, overridden 0, degraded 0
  status:         enforced
```

A.3 The fast path in pseudocode

```
on_append(entry):                  # inside the keeping stage's append
    if entry.session.temporary: return    # J1 §4.7: nothing kept
    for s in entry.owner_spans:    # J1 §4.1-4.2; quoted: never sent
        tag = s.intent             # stored at append (A.1); never
        if tag is None: continue   #   reclassified here or on replay;
                                   #   untagged: the night tags it
        if tag.label in (CORRECTION, PRAISE) or tag.standing:  # §3.1
            cid = (entry.address, s.index) # durable before the reply
            queue.put_once(id=cid, about=tag.about_action, version=tag.version,
                deadline=min(end_of_session, now + 30 min))
            tail.admit(cid, s.words)   # §7.4: may displace; owner told

on_restart():                      # rebuild events lost in a crash
    for entry in record.since(queue.last_address): on_append(entry)

reflect(event):                    # after the answer; at-least-once
    if ledger.processed(event.id): return
    for repair in required_repairs(event):   # §3.2: one child each
        child = event.id + "." + repair.letter  # J14's request ID
        ack = route(repair)        # rule gap: match (§4.2), J14 gate;
                                   # fact: J14 supersede; else a record
        if ack.terminal:           # accepted, hold, refused, or read back
            ledger.mark_terminal(child, ack)
        # pending is not terminal: child, event and tail line stay open
    if all(c.terminal for c in event.children):
        ledger.mark_processed(event.id)
```

Two more operations close the loop: the finding stage's acknowledgement, which lets a sentence leave the tail, and the model call, which is where every tail line is re-checked.

```
on_ack(ack):                       # J2 overlay receipt, per child (T10)
    child = ledger.child(ack.request_id)      # the request ID is the child ID
    event = ledger.event(child.parent)
    child.retrievable = ack.retrievable
    if any(c.held and c.universal for c in event.children):
        return                     # a held universal rule keeps its line
    if all(c.retrievable or c.held for c in event.children):
        tail.remove(event.cid)

model_call(session, request):        # the only way to call the model (T6)
    rules = newest_published_always_loaded()  # read now, no notice
    for line in tail.lines(session):
        r = rules.contradicted_by(line)  # memoised on (line, rules.version)
        if r is None: continue
        if r.published > line.admitted:      # he changed the rule later
            tail.withdraw(line, notice=True)
        else:                                # his newer words win (§7.3)
            line.set_aside(r); ask_once(line, r)
    return provider.call(request + tail.lines(session))
```

A.4 Numbers to tune

Table 15. Numbers to tune: starting points, not derived values; the receipts and the trial tune them.

Number	Where it is used	What it trades
300 ms	Classifier time limit (§3.1)	Reply delay against untagged spans
30 min, or the session's end	Event deadline (§6.1)	Early alarms against false ones
2 s	Model check's time budget (§5.3)	Delay per action against judgement
A week and 59 actions	Shadow run, false alarms (§5.3)	Wait against a bound under 5%
25 planted slips (R-42)	Shadow run, misses (§7.2)	Effort against the miss bound needed
About ten items	Morning report cap (§7.5)	His minutes against waiting items
8 lines, 400 tokens	Tail budget (§7.4)	Open corrections against attention
Two months, no incident	Freeing an unused rule (§7.2)	A shorter file against rare rules
Three failures a day	Procedure fallback (§8)	Patience against repeated failures
Three decisions	Retiring the test stream (§9.2)	Fresh cases against leakage
One rule in ten, a month	Opt-in holdout (§9.3)	An honest gain against slips met on purpose
Three weeks	A loop going dormant (§9.3)	His minutes against switching off too soon

Appendix B. The transition specification

One correction crosses four stages, and every handoff can fail. In plain terms: the owner sends one message with two corrections, a contact's new mobile number and a new rule about supplier drafts. While they travel through the four stages, much of what can go wrong does. The writing stage's judge is down, the owner edits a clashing rule by hand, an index build runs on old data, he erases the number, and a sub-agent asks for it by name. This appendix prints the series' specification of that journey: 24 handoffs (T1 to T20 on the journey, T21 to T24 only in the named cases), what each must store, what ends it and the only thing the agent may say, then the faults injected, the invariants and the result. The other three papers cite transition IDs. A checker replays the journey against a model of the design, a few dictionaries and one handler per message; it models the design, not an implementation, and says nothing about code built from it. The model, the checker and its regression cases are available from the author.

Identifiers. T1 to T24 are the handoffs, C1 to C8 the extra cases and I1 to I5 the invariants. s1012#204 is the message, /0 and /1 its two corrections, /0.a and /1.a their repairs. F- is a fact and R- a rule, with @n its revision. S- is a published snapshot, W- a write receipt, O- an overlay receipt, D- a deletion receipt, B- an index build and E- the erasure epoch; sa3 is the sub-agent. The overlay is the finding stage's list of writes accepted since its last index build, so a new item

can be found at once (J2 §8.2). A tombstone is the mark left where erased words were: the address stays, the words do not (J1 §4.4). The erasure epoch is a counter every erasure raises; a reader that has not caught up waits and shows nothing from before it (J14 §7.7).

The journey. One owner message, s1012#204, holds two corrections: /0, "Ada's mobile is now +44 7700 900 123, not the old one" (old fact F-0412@1), and /1, "from now on, copy me on every supplier draft". Each opens one child operation per repair (/0.a, /1.a); the child ID is J14's request ID. The write judge is down. While the proposals wait, the owner edits always-loaded rule R-15 by hand. The judge returns. Later he erases /0 while an older index build, B-55, runs, and a narrower subagent, sa3, asks by name for both items. The erasure epoch starts at E7, and B-54 is the live index. R-43, the rule from /1, is active in the tail only from T2 to T6 and is never retrievable as a rule in force.

State words. Table 16 gives the exact meaning of every word the transitions below allow.

Table 16. The state words, exactly (Table 6 gives an everyday picture of each). The agent may use only the word its state allows; invariant I3 checks it.

Word	What it means	Set by
In force		
provisional	The owner's universal instruction, in force in this session's tail while it waits for his approval	This stage
active	Admitted where it governs: rung 1 while its task runs, rung 2 on every turn, or a provisional line in its session	This stage
enforced	A check in the path of action carries it	This stage
enforced, degraded	The check is down or can no longer be tested, and runs its approved fallback (§5.3)	This stage
retired	Withdrawn, with the owner's approval	This stage
The saved copy		
captured	The owner's words are durably in the session record, before the reply	Keeping (J1)
pending	Waiting for the writing stage's judge; survives a crash, but not finished	Writing (J14)
held	Parked for the owner's decision and visible to him; acknowledged, not accepted	Writing (J14)
accepted	Written by the gate, with a receipt	Writing (J14)
refused	Not kept; the reason goes to the morning report	Writing (J14)
retrievable	The next relevant request can find it	Finding (J2)

Table 17. The 24 transitions. "Closes on" is the acknowledgement that ends the handoff, *delivery acknowledged*; an operation is *terminal* only when it reaches accepted, held or refused, so a pending receipt can be delivered and acknowledged without its operation being terminal. "Agent may say" is the only owner-facing claim the transition permits.

T	Handoff, and what is stored	Closes on (delivery acknowledged)	Agent may say
T1	Owner to J1. Entry #204 flushed with two immutable, tagged owner spans; a quoted span yields no correction	Durable flush with address, before the model call	"captured"
T2	J1 to J5. Events /0 and /1 queued (put_once on the correction ID); both sentences enter the tail	Queue durable before the reply	"captured, acceptance pending"
T3	J5 to J14. One child per repair: /0.a supersede F-0412, /1.a rule R-43@1 for rung 1. Requests carry addresses, not words; a ledger row is a proposal	J14 records each request ID	nothing new
T4	J14 to J5. Judge down. Read set S-101; F-0587 pending (W-1), closure of F-0412 held (W-2), R-43 pending (W-3): durable, not terminal	J5 records the pending receipt; events stay open	"captured, acceptance pending"
T5	Owner to J14. R-15@4, "supplier drafts from the sales address, nobody in copy", applied unchecked (W-4); S-102 published	W-4	"accepted, unchecked until the judge returns"
T6	J14 to J5. Before the next model call J5 reads S-102 itself; R-43's tail line contradicts R-15@4, which the owner set after it, and is withdrawn	A re-check receipt naming S-102	A one-line notice in s1012, then "captured, acceptance pending"
T7	Judge to J14. The late verdict on S-101 is discarded (R-15 moved 3 to 4); each pending request resumes on S-102 as a new attempt linked to its earlier receipt. Retry rule (J14 §7.2, A.3): a retry whose latest receipt is terminal gets it back; a pending receipt is never returned, the retry runs a new attempt; an owner's approval of a hold is also a new attempt on the same request	The resume, per request ID	nothing
T8	J14 to J5. F-0587@1 accepted; F-0412@2 superseded, since owner facts have earned closing permission (J14 §7.5) and the replacement is his own words. W-5 names S-102, S-103, E7, /0.a; the J2 delivery record is in the same transaction. Cold start: closure held, F-0412@1 stays live	W-5: /0.a terminal, /0 processed	"accepted, not yet retrievable"; cold start adds "held for your approval"

T	Handoff, and what is stored	Closes on (delivery acknowledged)	Agent may say
T9	J14 to J2. Overlay rows F-0587@1 and F-0412@2, owner only; the revision map (J2 §8.2) hides F-0412@1 in B-54. J14's delivery record (J14 Appendix A.2, Table 6) is closed when acknowledged_by = O-1	O-1	nothing
T10	J2 to J5. /0.a marked retrievable; the /0 tail line leaves because every child is retrievable or held	J5 records O-1	"retrievable"
T11	J14 to J5. R-43 held on S-102: clashes with R-15@4 (W-6); one report item. /1.a terminal, /1 processed	W-6: acknowledged, not accepted	"held for your approval: clashes with R-15"
T12	J14 to J2. R-43 recorded as a hold; readers are the owner alone (the intersection of #204/1 and supplier-drafts.md)	J2's overlay receipt	nothing
T13	Scheduler to J2. Build B-55 starts: manifest S-103, E7	Manifest written	nothing
T14	Owner to J14. Erase /0: E7 to E8 first; D-1 from the links; F-0587 tombstoned; F-0412@2 stays superseded; pending lines and holds dropped; the number's keyed hash enters the suppression identity	D-1 written; erasure incomplete until every place reports	"erase incomplete"
T15	J14 to J1. Span 0 redacted, span 1 kept; markers rebuilt; controller-transcript copies redacted; window rebuilt with an eviction event; provider copy named. The closing claim waits for the last row of D-1: J1, J2, J5 and J14's own rows, Git storage and backups with the recovery checkpoint (J14 §7.7, Table 3)	J1's rows of D-1	"erase incomplete", then after the last row "all reachable copies removed; not reachable: provider copy"
T16	J1 to J2. The read ledger drops the redacted range	J2 records the event	nothing
T17	J14 to J2. Overlay rows and use keys dropped; the revision map tombstones F-0587	J2's row of D-1, once no build that read E7 can swap in	as T15, after the last row
T18	J14 to J5. /0 kept as an address only; tail line and evidence quotes dropped; the tail waits until J5 has applied E8	J5's row of D-1	as T15, after the last row
T19	Scheduler to J2. B-55 finishes: swap refused (E7 is not E8); B-54 and the overlay serve. With no erasure, B-55 swaps in with the overlay rows and revision-map entries newer than S-103	The swap decision	nothing

T	Handoff, and what is stored	Closes on (delivery acknowledged)	Agent may say
T20	sa3 to J2. It asks for the mobile and R-43 and gets nothing: no entry, pointer, tombstone, hold or count. R-43's readers are the intersection of its sources', the owner alone; no grant exists	The pull receipt (S-104, E8)	To sa3 nothing; to the owner "held for your approval"
T21	Owner to J2 (case C6). sa3 loses supplier-drafts.md; reader sets recomputed; a running query is filtered again at return	J2 records the permission version	nothing
T22	J14 to J1 (C8). Quarantine of an injected tool result already in the window: entry flagged, markers rebuilt, window invalidated and rebuilt	J1's quarantine receipt	nothing
T23	J14 to J2 (C8). Every row reached through links is marked; only the audit view shows it	J2 has applied the epoch	nothing
T24	J14 to J5 (C8). Evidence and fixtures linked to the span marked (not represented in the model, see the coverage table); the tail waits until J5 has applied the epoch	J5 has applied the epoch	nothing

The series rules behind the table: handing off is not completing, so a sender keeps each message in a durable outbox and resends until the receiver acknowledges the resulting state; every receiver is idempotent on its message key; a delivery record carries IDs, never content; every read path serves only at the newest erasure epoch its stage has applied, and waits otherwise.

Faults. At every handoff the checker injects, one at a time, a crash before the acknowledgement (the receiver acts, its acknowledgement is lost, the sender resends), a duplicate delivery, and a reorder (the message goes behind everything else in flight). It runs ten cases: the journey, its cold-start branch, and eight more: C1 commit before notification (a crash right after publication; a lost delivery); C2 old copies of T2, T8 and T9 arriving after the erasure; C3 a pending receipt, then the judge's recovery; C4 two repairs from one correction; C5 a correction committed while an index builds; C6 permission revoked during a query; C7 erasure while content is loaded in the window; C8 quarantine after the content is loaded.

Table 18. The five invariants.

ID	Invariant, checked after every step
I1	No correction is lost: each one in the record is queued at J5 or recoverable from the record; an event closes only when every required child is terminal; at rest each is terminal or in a named pending state
I2	Nothing forbidden is readable: no reader class reads erased content, quarantined content outside the audit view, or an item outside its allowed readers, by any path, including a query already running
I3	No claim beyond the state: each owner-facing claim uses only the words its transition allows, and the newest claim on a subject stays true at every step
I4	Derived access is the intersection: every row J2 holds records readers equal to the intersection of its sources' sets, through the dependency links
I5	No contradiction in force: at every model call, no provisional tail line contradicts an always-loaded rule the owner set after that line and published before that call's check

Result. All 516 fault runs passed, 172 each of crash, duplicate and reorder; with the 11 structural checks (one on the specification, one on each of the ten cases) the checker reports 527 PASS and 0 FAIL. Of the 172 reorders, 104 were no-ops, with nothing in flight to overtake, so they tested nothing; the other 68 test reordering. In the reference journey, the reorders injected at this stage's own T2, T6 and T10 all had nothing to overtake, so T6 rests on regression case d, not on reordering. The checker also removes one mechanism at a time, 18 seeded defects in all, and requires the run to fail: it caught 18 of 18. Without the re-check inside the model call (T6), R-43's line stays in force and I5 fails. Four further sequences are kept as regression cases, each passing on the model and failing when its mechanism is removed: an erase and a quarantine that land while a query runs (return-time epoch check), a retry while the judge is down (a new attempt, never the cached pending receipt), and a correction whose reply reaches the model with no separate turn-boundary event (the re-check inside the call).

The checker asserts a stricter I5, that no tail line contradicts any published rule. The journey meets both, since its one clash is with R-15@4, set after R-43's line; the case where his newer words set aside an older rule (§7.3) is not yet modelled.

What the run shows is narrow: the enumerated scenarios passed. It covers scripted journeys, handlers that run atomically and one injected fault at a time; it does not show that every legal sequence keeps the invariants, and it says nothing about code. The table separates what the model asserts from what it only represents and what it executes.

Table 19. What the checker covers. The next step is generated legal event sequences with every returned answer and model call observed independently.

Behaviour	Asserted after every step	Represented in the model state	Executed by a script
No correction lost; parents close on every child (I1)	yes	yes	journey, C3, C4
Nothing forbidden readable, including answers in flight (I2)	yes	yes	journey, C6 to C8, regressions a and b
Claims within the state (I3), derived access (I4)	yes	yes	every case
No contradicted tail line at a model call (I5)	yes	yes	journey, regression d
Retry rule: pending never returned from cache	at rest, by receipt count	yes	C3, regression c
Marking rule evidence and fixtures on quarantine (T24)	no	no: the epoch is applied, the marking is not modelled	C8
Owner approval of a hold	no	no	none
His newer words setting aside an older rule (§7.3)	no	no	none
Two faults at once, generated sequences	no	no	none