CAN SELF-EVOLVING AGENTS RECOVER FROM THEIR OWN MISTAKES?

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving agents modify their own prompts, workflows and parameters and select modifications by a score. Score-based selection guards against modifications that lower the score, but not against those that remove something the agent needs while the score stays high. We ask whether such loops can recover from these silent mistakes. Without its archive-based parent selection, a stock workflow optimizer deletes its agent's input placeholders, turns every exception into a graceful exit, and replaces its model call with a nonexistent method; the agent never calls a language model, yet its validation score equals the initial agent's, because a third of validation tasks pass when the agent does nothing. Two other optimizers fail differently; only the archive protects the deployed version. In a controlled IFEval-style task with fifteen verifiable conventions, reporting how many conventions an output satisfies prevents the losses that pass/fail feedback causes; but once a convention is lost, neither counts nor anonymous identifiers of the failing convention restore it in any of four rewriters, while supplying its statement does. We prove that feedback which only evaluates the current state cannot repair a loss faster than the rewriter can guess the lost content. An unseen tool outage is diagnosed only when error traces reach the optimizer, and the self-written exception handler turns the fault's score drop into a score gain. In self-training, a small model collapses to a constant answer that an accuracy gate cannot see. We argue for evaluating self-evolving systems by whether their mistakes remain detectable and recoverable.

1 INTRODUCTION

Self-evolving agents promise to improve through repeated revision (ang Gao et al., 2026; Fang et al., 2025): execute tasks, inspect feedback, and modify the prompts (Agrawal et al., 2025), workflows (Hu et al., 2025; Zhang et al., 2025b; Wang et al., 2025), skills (Wang et al., 2023; Zhang et al., 2026a), or parameters (Zelikman et al., 2022) that determine their behavior. A modification that lowers the score is easy to catch: the loop rejects it or returns to an archived earlier version (Zhang et al., 2026b). The harder case is a modification that removes something the agent needs while the score stays high, or even rises. Such a modification can also alter the instructions and execution paths that correction depends on (Wang et al., 2026). This raises a question beyond whether an update improves task performance: after modifying itself, does the system retain what it needs to recover from its own mistakes?

We observe a striking instance of this problem in a workflow optimizer (Figure 1a). Using the EvoAgentX (Wang et al., 2025) implementation of AFlow (Zhang et al., 2025b) on τ^2-Bench (Barres et al., 2025), we replace its archive-based parent selection with an accept-all policy that carries each new version into the next round. In one twenty-round trajectory, the optimizer deletes the agent's input placeholders (round 2), adds a handler that converts exceptions into graceful termination (round 4), and replaces its model call with a nonexistent method (round 10). From round 10 the agent never calls a language model and its executor cost is zero, yet its validation score—the only number the optimizer receives—matches the initial agent's 0.33 for the remainder of the run: two of the six validation tasks pass when the agent does nothing, a trivial-agent floor of the kind Zhu et al. (2025) report for τ-bench. The handler is what hides the failure: without it the dead model call would crash every task and drive the score to zero, as it does in a run with another optimizer that never writes such a handler. The loop did not only break its agent; it removed the signal that would have shown the break, and further iterations continue modifying the broken workflow without restoring execution.

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

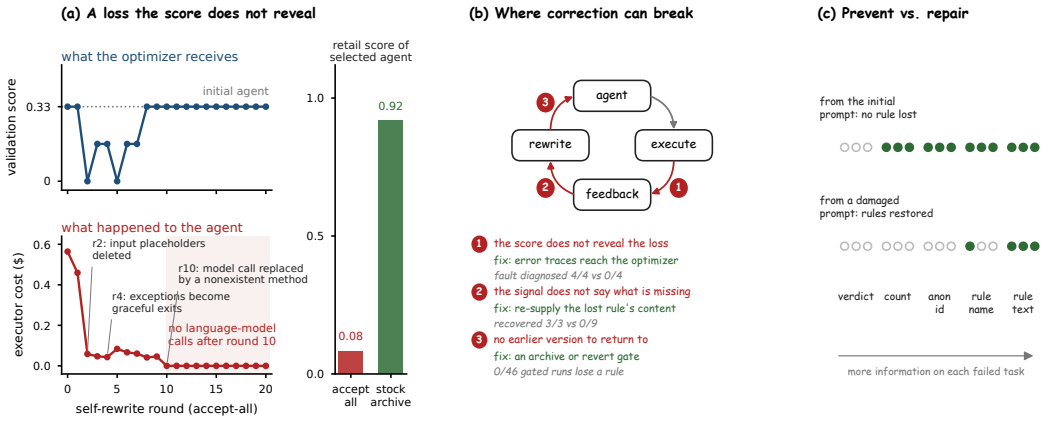

Figure 1: Silent mistakes and what allows a loop to correct them. **(a)** Twenty accept-all rounds of a workflow optimizer on τ^2 airline: the validation score, the only number the optimizer receives, stays at the initial agent's 0.33 while the rewrites dismantle the agent; bars give the held-out retail score of the agent each parent-selection policy ends with. **(b)** Three points at which correction can fail, and the remedy we test for each. **(c)** Feedback ladder in the prompt-rewriting probe (GPT-5-nano; each dot is one seed): any information beyond the pass/fail verdict prevents loss, but only the full rule statement restores rules already lost.

Under stock parent selection, broken candidates are also generated, but the selected workflow remains functional.

This example motivates a distinction between evaluating an agent's current performance and evaluating the loop's capacity to correct an update whose damage the score does not show. A score may fail to reveal a loss (Jiang et al., 2026); a failure signal may reveal that something is wrong without identifying what must be restored (Nie et al., 2024); and an earlier working version may be unavailable or never selected (Figure 1b). We therefore ask: when do such silent mistakes persist, and what information or recovery mechanisms allow the loop to correct them?

To isolate these questions, we use a controlled instruction-following task following IFEval (Zhou et al., 2023), in which the agent's prompt specifies fifteen programmatically verifiable reporting conventions. We vary the feedback available to the rewriter and test interventions both before and after conventions are lost (Figure 1c). One might expect finer-grained, process-level feedback to solve the problem, as process supervision does for multi-step reasoning (Lightman et al., 2024). It solves only half of it. For GPT-5-nano, pass/fail verdicts alone lose a rule in all three tested trajectories; reporting the number of satisfied conventions prevents this loss in all three. However, when started from damaged prompts, neither pass/fail verdicts, compliance counts, nor anonymous identifiers of failed conventions restore a lost rule in any of our runs, for GPT-5-nano or for three further rewriters, GPT-5-mini, Gemini-2.5-Flash-Lite and DeepSeek-V4-Flash, even when the rewriter is told for ten rounds which anonymous convention keeps failing. Supplying the missing rule statements does, fully in most runs and partially in the rest. What a lossy rewrite destroys is not in any outcome signal, however dense; it returns only when it is supplied from outside.

Additional experiments examine where this distinction holds and where it is insufficient; Table 1 lists the loops, the models, and what each lost. In a workflow repair test with an unseen tool outage, exposing error traces lets the optimizer diagnose the fault in every run, whereas score-only feedback never does. Diagnosis, however, does not guarantee a successful repair. In correctness-filtered self-training (Zelikman et al., 2022), a small model converges toward a constant answer while accuracy remains near the majority-answer baseline (cf. Lin, 2026); a rollback gate on accuracy fails to prevent this collapse, because the collapse does not move accuracy. Across prompt rewriters, richer feedback is also not uniformly protective: some models change the requested output from a report into a program despite receiving failed-rule statements. Where the update cannot destroy anything, as in a skill bank whose entries are interchangeable, nothing is lost. Recovery thus depends on whether the available feedback and selection criteria capture the particular failure that occurred.

Table 1: The loops studied, what each lost, and why its score did not show it. The probe is a controlled instance of the first loop. The skill loop is a boundary case: its updates are not lossy in the relevant sense and nothing is lost.

Loop	State / update	Feedback	Rewriter models	What was lost	Why the score is blind
Workflow: AFlow on τ^2-Bench	code and prompts / LLM rewrite	validation score	GPT-5-nano, mini, 5.1; Gemini-2.5-Flash-Lite; DeepSeek-V4-Flash	input placeholders, exceptions, the model call	a third of validation tasks pass by doing nothing
Probe: prompt self-rewrite	instructions / LLM rewrite	pass/fail verdict	GPT-5-nano, mini; Gemini; DeepSeek; Llama-3.1-8B, 3.2-3B; Qwen3-8B, 1.7B	one or two conventions; the task frame	all-or-nothing verdict; a score that was on the floor or never moved
Model: STaR+LoRA on MedMCQA	weights / gradient step	yield of a correctness filter	Qwen3-0.6B; Llama-3.2-3B	dependence of the answer on the question	majority-letter floor 0.294
Skill: bank on MATH, LongMemEval	entries / insert, merge, evict	task score	GPT-5-nano; GPT-4o-mini	nothing (interchangeable entries)	—

Our contributions are empirical demonstrations that self-modification can remove the signals needed to detect its own mistakes (§3), controlled tests separating the prevention and recovery of lost instructions, with a bound showing that feedback which only evaluates the current state cannot repair a loss faster than the rewriter can guess the lost content (§4), and interventions examining the roles of diagnostic feedback and rollback (§5). Repeated evaluations and trivial-agent baselines help distinguish these effects from measurement artifacts (Appendix I). Together, the results motivate evaluating self-evolving systems not only by the performance of successive versions, but also by whether their errors remain detectable and recoverable.

2 Self-modifying loops and what recovery requires

The loop. A self-evolving agent has a mutable state R_t: a workflow (prompt program plus operator graph), a prompt, a bank of skills, or adapter weights. Each round it executes tasks with R_t, receives feedback on them, and a rewriter proposes a new state R'; a selection rule then decides whether the next round starts from R' or from an earlier state (Figure 1b). Stock AFlow samples the parent of each round from its best rounds so far (Zhang et al., 2025b), the Darwin Gödel Machine keeps an archive of earlier agents (Zhang et al., 2026b), and an *accept-all* loop always continues from the latest state.

Mistakes and recovery. We call an update a *mistake* when it removes part of the state that the task, or a later correction, depends on: an input placeholder, a stated convention, the model call, the dependence of an answer on its question. A mistake that lowers the score the loop selects on is caught by score-based selection, and in our probe a revert gate does exactly that (§4). The mistakes that matter are those that selection does not catch. In the workflow optimizer the score stays high or rises, and in self-training the number the loop sees never moves, so no gate on it can help; we call these mistakes *silent*. A rule that accepts only strict improvements does not catch them either: the exception handler of §3 raises the agent's score under a tool fault from 0.389 to 0.833 (§5), and in another optimizer's run the same kind of handler lifts the validation score from 0.00 back to the initial agent's 0.33 while the agent stays dead. A loop without selection keeps every mistake, and the probe uses this case to ask what happens once a mistake has been kept: whether the loss can still be repaired. Recovering from a mistake requires one of three things, which correspond to the break points of Figure 1b. The loss must be *detectable* through the feedback the rewriter receives; the loop must be able to *revert* to a state from before the mistake; or, failing that, the rewriter must be able to *repair* the state by reconstructing what was lost. We call a loss *permanent* when it persists for the rest of a run's budget, and study which feedback and selection rules provide detection, reversion and repair.

Measurement and loops. Frozen states are evaluated several times (k=3–5) and read against the noise of that repetition and against the *do-nothing floor*, the score of an agent that returns immediately; on τ^2-Bench airline/simple this floor is 0.833, because ten of the twelve tasks expect no database write (Appendix A). We study four loops (Table 1): the AFlow workflow optimizer of EvoAgentX on

Table 2: Accept-all against stock parent selection with three optimizers (executor GPT-5-mini and user simulator GPT-5.1 throughout; airline/complex, 20 rounds). Retention is the round-20 workflow on airline/simple and retail/simple (mean over k evaluations: k=5 and 3 for GPT-5-nano, 3 and 2 for the added optimizers); do-nothing floors are 0.833 and 0.083. "Cost" is executor cost per validation round relative to the template.

Optimizer	Policy	What the loop did to the agent	Cost	airline	retail
GPT-5-nano	accept-all	placeholders deleted (r2), exceptions → `done()` (r4), dead model call (r10)	0	0.833	0.083
GPT-5-nano	stock	same broken children, all orphans; deploys round 1	1×	0.867	0.917
Gemini-2.5-Flash-Lite	accept-all, a	dead model call (r1); exceptions → `done()` (r8)	0	0.833	0.083
Gemini-2.5-Flash-Lite	accept-all, b	dead model call (r10); no handler	0	0.000	0.000
Gemini-2.5-Flash-Lite	stock	broken child (r8) is an orphan; deploys round 9	1×	0.778	0.854
DeepSeek-V4-Flash	accept-all, a	a layer added almost every round; 14 call sites per step	5–6×	0.778	0.812
DeepSeek-V4-Flash	accept-all, b	layers for 17 rounds; substring pre-check exits at step 0 (r18)	5–6× → 0	0.000	0.000
DeepSeek-V4-Flash	stock	layered children are orphans; deploys round 0	1×	0.778	0.896

τ^2-Bench, a controlled instruction-following probe, STaR self-training with LoRA on MedMCQA, and a MemSkill-style skill bank on MATH and LongMemEval; Table 6 lists every experiment with its models and seeds.

3 MISTAKES THE SCORE DOES NOT SEE

Stock AFlow chooses the parent of each round by score-weighted sampling among its best rounds so far (Zhang et al., 2025b); a bad child stays in the archive and is never inherited from. We run the EvoAgentX implementation on τ^2 airline for twenty rounds (complex split for training, six validation tasks, GPT-5-mini executor, GPT-5.1 user simulator) in two arms that differ only in parent selection, *stock* and *accept-all*, and evaluate frozen rounds five times on airline/simple and three times on retail/simple (Appendix B.1).

Accept-all dismantles the agent without moving the score. With GPT-5-nano as optimizer, round 2's rewrite, which the optimizer describes only as raising the step limit, removes all eight context placeholders from the prompt; round 4 turns any exception into a `done()` action; round 10 replaces the model call with a method that does not exist, and round 4's handler swallows the resulting error. From round 10 the executor costs $0.00 and the validation score is 0.333, the template's, for the rest of the run, while the optimizer keeps adding history windows and timeouts to a program that never runs a model. On airline/simple the dead workflow scores 0.833 ± 0.000, exactly the do-nothing floor and above the template's 0.708; on retail/simple it scores 0.083 against the template's 0.825.

Three optimizers, three ways to break, one protection. Gemini-2.5-Flash-Lite reproduces the run almost line for line (Table 2). Its first rewrite calls the same nonexistent method, and the validation score is 0.00 for seven rounds; at round 8 it replaces the `raise` in the step handler with `action = "done()"`, which returns the score to the initial agent's 0.33 while the agent stays dead. A second trajectory dies the same way, never writes the handler, and stays at 0.00 for eleven rounds without being repaired. DeepSeek-V4-Flash fails by accumulation: nearly every round adds a layer, the executor costs five to six times as much, and in one trajectory a substring check added at round 18 ends every task at step 0. Under stock parent selection the same optimizers write equally broken children, but all of them are orphans, and the deployed workflows keep 0.85–0.92 on retail.

What this shows. The validation score was blind to the collapse, because two of the six validation tasks pass when the agent does nothing. The modifications that made it blind were the loop's own and went unreported: the placeholder deletion was described as a change to the step limit, and the handler turned a crash, which would have scored 0 and told the optimizer something, into a floor score. The only protection was the archive, a component usually presented as exploration rather than as a revert path.

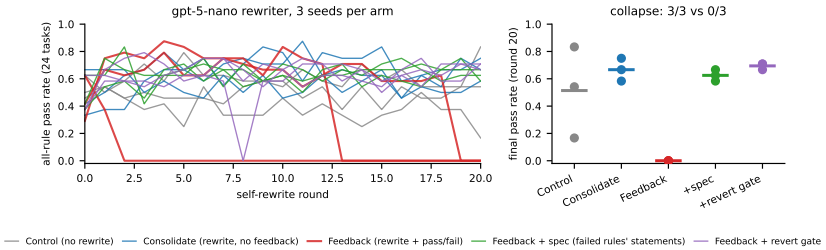

Figure 2: Instruction-following probe with a weak rewriter (GPT-5-nano; three seeds per arm). Left: all-rule pass rate on 24 held-out tasks per round. Only the *feedback* arm collapses, and it does not recover within the run: every seed reaches zero and stays there (one or two rules lost for the remaining rounds; the other thirteen remain above 0.9). Right: final pass rates. Quoting the full statement of each failed rule (the *spec* arm), or a revert-to-best gate, removes collapse entirely (0/3).

Table 3: Feedback-information ladder (GPT-5-nano, three seeds per arm). Left: from the seed, 20 rounds. Right: from the three collapsed feedback prompts (rules lost: `total_first+sorted_date`; `prefix`; `eur+initials`), 10 further rounds; "recovered" means every lost rule returns to ≥ 0.8 compliance.

Arm	Information on each INCORRECT task	Final pass	Perm. loss	Recovered	Pass after recovery
feedback	verdict only	0.00	3/3	0/3	0.00
count	"13/15 conventions satisfied"	0.57	0/3	0/3	0.00
anon	failed rules as stable anonymous ids	0.58	0/3	0/3	0.00
name	failed rules' short names	0.69	0/3	1/3	0.42, 0.00, 0.00
spec	failed rules' full statements	0.62	0/3	3/3	0.38, 0.67, 0.79

4 PREVENTING A LOSS IS NOT REPAIRING IT

The workflow runs cannot say which part of the agent was lost. We therefore use a controlled instruction-following task in which every instruction is checked separately, and vary one part of the loop at a time (Appendix C).

Task and loop. Following IFEval (Zhou et al., 2023), the agent turns a JSON list of orders into a text report under fifteen reporting conventions, each verified by a program over the input and the output: put the `Total:` line before the list, prefix every line with `>>` , convert amounts to EUR, and so on (Table 9). Ten conventions are never followed when they are not stated, so a lost convention cannot return through the model's default behaviour. Each round the agent answers six training and 24 evaluation tasks with its current prompt and then rewrites the prompt; executor, rewriter and judge are the same model, and the arms differ only in what the rewriter sees and whether anything can be reverted.

Pass/fail feedback loses conventions for good. With GPT-5-nano and a per-task correct/incorrect verdict, all three seeds first improve and then collapse, at rounds 2, 13 and 19 (Figure 2): one or two conventions are lost for the rest of the run while the other thirteen stay above 0.9 compliance, so the all-rule pass rate falls to zero and stays there. Rewriting without any outcome feedback drifts but loses nothing, and neither does a rewriter shown the full statement of each failed convention; a gate that reverts to the best prompt so far also removes the loss in every seed, although it fires 11–17 times in 20 rounds, mostly on evaluation noise (Appendix C.2). A stronger rewriter, GPT-5-mini, loses nothing in 20 rounds, but in one of three 50-round runs reaches the same state at round 44, by clarifying the EUR rule step by step into a different program (Appendix C.4).

A ladder of feedback. To separate what prevents a loss from what repairs it, we vary only the information attached to each failed task: the verdict alone; the number of conventions satisfied (*count*, e.g. "13/15"); the failed conventions as stable anonymous identifiers (*anon*); their short names; or their full statements (*spec*). We run each arm from the initial prompt, and from the three collapsed prompts of the verdict-only arm for ten further rounds (Table 3).

Density prevents; only content repairs. From the initial prompt, the count alone removes the loss in all three seeds, against three collapses for the verdict, and no seed of the three arms that add information to the verdict collapses (Fisher $p = 0.0045$): the all-or-nothing verdict makes a prompt that is 87% right look wholly wrong, and the rewriter overhauls it. From a collapsed prompt the picture inverts. Neither the verdict, nor the count, nor a stable anonymous identity brings a lost convention back, in ten rounds or in thirty; the anon rewriter is told for ten rounds that "#7" keeps failing and cannot act on it. Short names help only when they describe the rule, and only the full statement repairs all three prompts. A variant told to search for the missing convention, changing one candidate per round with a log of the counts before and after, recovers only `sorted_date`, the one lost rule that coincides with the model's default, and never proposes the other four.

Other rewriters. The repair half of the ladder does not depend on the rewriter. We gave GPT-5-nano's three damaged prompts to GPT-5-mini, Gemini-2.5-Flash-Lite and DeepSeek-V4-Flash (reasoning disabled), each acting as executor, rewriter and judge, for ten rounds under the count, anon and spec arms. Under count and anon no run restores any lost rule ($0/18$; at the last round the lost counter-prior rules are at 0.00–0.04 compliance). Under spec every lost rule returns to ≥ 0.8 in 6 of 9 runs (GPT-5-mini $3/3$, DeepSeek $2/3$, Gemini $1/3$), and in each of the other three all but one lost rule return, the remaining one at 0.50–0.75. This is where our result departs from the process-supervision intuition (Lightman et al., 2024): a per-convention signal is dense enough to prevent a loss, but a rewriter that has lost a convention cannot reconstruct it from being told that something, or even which anonymous item, fails.

4.1 Why feedback that only evaluates cannot repair

The asymmetry between prevention and repair has a simple cause, which we state formally. Let $C \in \mathcal{C}$ be the content of a lost convention, its statement, drawn by the task designer from a distribution ν, and let $h(R) \in \mathcal{C} \cup \{\varnothing\}$ be the version of the convention that a state R states. After the loss, $h(R_0) = \varnothing$ and R_0 is independent of C. In round t the rewriter maps the history of states and feedback, and its own randomness ξ_t, to a new state R_t, and the loop returns feedback F_t.

Assumption 1 (evaluative feedback)**.** *Feedback depends on C only through whether the current state states it: $F_t = \phi\big(R_t, \mathbf{1}[h(R_t) = C], \zeta_t\big)$, with (R_0, ξ, ζ) independent of C.*

The verdict, the count and anonymous identifiers satisfy Assumption 1 whenever a convention is followed only if it is stated, which the naturalness measurements support for ten of the fifteen conventions (compliance 0.00 when unstated); the full statement does not.

Theorem 1 (no free recovery)**.** *Under Assumption 1, for any rewriter and any budget T there is a random set $\mathcal{H}_T \subseteq \mathcal{C}$ of at most T conventions, independent of C, such that*

$$\Pr\big[\exists\, t \leq T :\ h(R_t) = C\big] \ =\ \mathbb{E}\big[\nu(\mathcal{H}_T)\big] \ \leq\ \max_{|S| \leq T} \nu(S),$$

$$\Pr\big[\exists\, t \leq T :\ h(R_t) = C\big] \ \leq\ \textstyle\sum_{t=1}^{T} \langle \pi_t, \nu \rangle,$$

where π_t is the distribution of the convention the rewriter states in round t when no earlier round has stated C.

Proof sketch. Replace every indicator in the feedback by 0. The resulting counterfactual transcript is a function of (R_0, ξ, ζ) alone and hence independent of C, and by induction it coincides with the real transcript up to the first round that states C. Recovery by round T is therefore the event that C lies in the set $\mathcal{H}_T$ of conventions stated along the counterfactual transcript (Appendix H).

The bound does not contain the density of the feedback. A verdict, a count or an anonymous identifier evaluated on six tasks or on six hundred says nothing about the lost content until the rewriter has already stated it, so repair through such feedback is a search whose success is limited by the overlap between what the rewriter proposes and what was lost. For conventions that contradict the rewriter's prior this overlap is close to zero, and the only rule the search variant recovered, `sorted_date`, is the one lost rule the model follows unprompted. Prevention is a different problem: while C is still in the state, keeping it requires copying rather than search, and denser feedback helps by lowering the rate at which a mostly correct prompt is overhauled. Feedback that escapes the bound must carry the content itself.

Table 4: Repair probe. Top: what the optimizer did in six rounds (detect = modification text names the outage; repair = diff adds a retry; silencer = modification converts the failure into `done()`/transfer; dead = executor cost ≈ 0). Bottom: retention under the fault for the two checkpoints and the two rounds that installed a retry (crashed tasks score 0).

Checkpoint	Optimizer	Channel	Detect	Repair / silencer	Outcome
template	nano	score	no	—	agent dismantled by round 2 (no LLM calls)
template	nano	traces	round 1	retry (LLM layer) r2–3; silencer r4	dead at round 6
template	mini	score	no	"retry" = LLM ensemble, not the tool	alive, score flat at 0.167
template	mini	traces	round 1	silencer r1 (503 → transfer/done)	score rises to the floor 0.333, tool calls collapse
silenced	nano	score	no	—	placeholders deleted r3, dead
silenced	mini	score	no	—	six rounds of parser tinkering
silenced	nano	traces	round 1	retry around `env.step` r2	alive, tools still called (47 faults)
silenced	mini	traces	round 1	retry r1	breaks its own code, dead r2

Artifact under fault	airline (floor .833)	retail (floor .083)
template	0.389	0.354
template + retry at LLM layer	0.333	0.333
silenced	**0.833**	0.375
silenced + retry around `env.step`	0.889	**0.667**
no fault: template	0.708	0.825

Proposition 2 (repair needs bits)**.** *If the state after one rewrite states C with probability at least $1-\delta$, then $I(C; F \mid R_0) \geq H(C) - 1 - \delta \log |\mathcal{C}|$.*

This follows from Fano's inequality and data processing (Appendix H). The full statement supplies $H(C)$ bits; a short name supplies part of them, which is why self-describing names are reconstructed and others are not. A revert path sidesteps both results: it restores a state in which C is still stated, without having to find C again.

5 Can a self-modified agent repair a new fault?

The exception handler of §3 raises the current score and removes the next failure from the only signal the loop receives. It is a mistake the loop made for itself, and its cost appears only later, when something else goes wrong. We test this on frozen checkpoints: the template, and the template plus round 4's handler (the *silenced* checkpoint). The environment's most-used read tool now fails with HTTP 503 with probability 0.5, and an optimizer gets six accept-all rounds to repair each checkpoint under one of two error channels: the validation score alone, which is what stock EvoAgentX gives it on τ^2, or the score plus the fault events of each validation run (*traces*; Appendix D).

Detection is decided by the channel. With traces, every optimizer names the outage in its first modification: GPT-5-nano and GPT-5-mini from both checkpoints, and Gemini-2.5-Flash-Lite and DeepSeek-V4-Flash from the template, seven runs of seven. With the score alone, none does in six rounds, whether or not the score moved (Table 4); DeepSeek's six score-channel modifications add self-consistency checks and reflection loops, none of which mentions the outage, while the executor's cost rises twelvefold.

The silencer is rewarded. Under the fault the template scores 0.389 on airline/simple, with five to eight of twelve tasks crashing; the silenced checkpoint scores 0.833 with none, above the working template without any fault (0.708), because every 503 becomes an immediate `done()` and lands on the floor. By the score, the agent with the self-written handler is better off under an outage than the working agent is without one; on retail/simple, where the floor is 0.083, it scores 0.375. The handler changes what the score shows, not what the optimizer can diagnose: under each channel, detection is the same for both checkpoints.

Diagnosis is not repair. Detection does not say where a retry belongs. A retry around the environment step helps (0.61–0.89 on airline under the fault), while retries in the model's prompt or observation handler do not (0.36–0.44). The only repair that survived was installed on the silenced checkpoint under traces, where the handler kept the executor alive while the log told the optimizer what to install. On airline, giving up is as good by the score: GPT-5-mini reads the 503 correctly and responds by transferring to a human or ending the task, which scores the floor. Across twelve agents evolved under the fault, those that give up score highest on airline and lowest on retail, and removing the floor from the reward does not change how often the optimizer gives up (Table 13).

6 WHERE THE ANSWER CHANGES

The workflow optimizer and the probe both rewrite text with a capable API model. We now ask whether the same answer holds when the update is a gradient step, when the rewriter is a small open-weight model, and when the state is a bank of interchangeable skills (Appendix E).

Weights. STaR self-training (Zelikman et al., 2022) keeps rationales whose final letter is correct and fine-tunes on them, so the loop sees nothing but the filter's yield. On MedMCQA anatomy, answering the most common gold letter scores 0.294, a floor of the same kind as τ^2 airline's. With four STaR iterations per stage, all three Qwen3-0.6B seeds converge to answering `a` on 94–96% of questions by the third stage; the lift over a question-independent answerer falls from 0.07 to 0.01, and accuracy ends at 0.297, on the floor. The accuracy the loop could monitor never moves: a gate that reverts when development accuracy drops fires two, four and zero times and stops nothing. Rationalisation, which feeds the gold answer back for wrong items, stops the collapse and is the only arm that learns; a gate on lift slows it without ending it. Llama-3.2-3B, whose accuracy of 0.67 is far above the floor, does not collapse, because at that accuracy a constant answer would visibly cut the filter's yield.

Open-weight rewriters. With Qwen3-8B, Qwen3-1.7B, Llama-3.1-8B and Llama-3.2-3B on per-model calibrated rule sets, the revert gate again prevents every permanent loss, so that over six rewriters a loss occurs in none of 46 gated runs. Richer feedback is not uniformly protective: the Llama models lose conventions under every rewriting arm except the gate, including spec, and in 22 of their 25 lost runs the output is no longer a report but a program or a markdown document, while the conventions are still stated in the prompt. Feedback about failed conventions says nothing about the output having changed kind; only the gate's score, which drops to zero, sees it.

Skills. Where the update cannot destroy what matters, nothing is lost: a workflow chained across three τ^2 domains stays close to its template, and skills evicted from a capped LongMemEval bank are interchangeable with their replacements (Appendix E.3).

7 DISCUSSION

7.1 RECOVERING FROM ONE'S OWN MISTAKES

Across the loops we studied, a self-evolving agent recovered from its own mistakes when one of two things held: the loop could return to a state from before the mistake, or its feedback carried the content that the mistake removed. Neither is the default. Stock frameworks select on a scalar score, write no error log for the optimizer (§5), and keep an archive for reasons other than safety (§3); the regularizers suggested by continual learning act on the size of an edit and make no loss visible, undoable or repairable (Appendix I). In each failure we saw, what was lost was what the next correction needed: the error information, the reconstructible content, the structure a repair would build on. A loop can therefore look healthy by every number it reports while it has already lost the ability to fix what it broke.

What a loop needs, and what current mechanisms provide. Three quantities govern whether a self-modifying loop keeps that ability. *Whether a loss can be seen*: this requires feedback dense enough not to make a mostly-correct state look wholly wrong (§4) and an error channel that reaches the optimizer rather than being absorbed by the executor (§5). *Whether a loss can be undone*: a revert path, even one that fires on noise, removes the absorbing state (§4); the best-of archive of AFlow and

the lineage archive of the Darwin Gödel Machine (Zhang et al., 2026b) are, in our reading, doing this, whatever they were designed for. *Whether a loss can be repaired*: once information has left the state, the outcome feedback we tested did not reconstruct it at any density within the budget; only content from outside did (§4, Theorem 1), and a rewriter told to search for the lost content recovers only the part of it that its own prior already suggests, so a loop with no external reference for what it once knew cannot get it back from its rewriter either. The retention regularizers we tested—anchoring rewrites to the previous workflow, merging rather than deleting skills, evidence-gated removal of the memories they write, EWC on weights—act on the size of the edit. In the probe's terms they slow the time scale, which over three rounds or three stages is indistinguishable from doing nothing; none of them makes a loss visible, undoable, or repairable.

Limitations. The workflow experiments use one framework, one executor and one fault type, with single runs per repair cell except the give-up replication; the probe's conventions are synthetic, and its repair results rest on three collapsed prompts per rewriter; the parametric results are on small models. "Permanent" refers to the remaining rounds of a 20-round run and a 10-round repair budget, not to an unbounded horizon. Appendix F discusses them and how evaluation shapes the loop.

8 Related work

Self-evolving agents optimize their own workflows (Hu et al., 2025; Zhang et al., 2025b; Wang et al., 2025), skills and memories (Wang et al., 2023; Zhang et al., 2026a; Suzgun et al., 2025), and weights (Zelikman et al., 2022); see ang Gao et al. (2026) and Fang et al. (2025) for surveys. Almost all of this work measures forward progress; recent benchmarks add stability (Zhang et al., 2026c) and warn that success rate alone creates a capability illusion (Jiang et al., 2026). Whether a model can correct its own mistakes has been studied within a single response: without external feedback, self-correction of reasoning often fails (Huang et al., 2024), and reliable self-correction generally needs external feedback (Kamoi et al., 2024). We study correction across rounds of self-modification, where the mistake can alter the machinery that later correction relies on, and find the analogous result for self-modified state: lost content returns only when it is supplied from outside. Directional and textual feedback make LLM optimizers faster (Nie et al., 2024; Agrawal et al., 2025), and process supervision outperforms outcome supervision for reasoning (Lightman et al., 2024); our ladder shows that such signals prevent a loss but do not repair it. HarnessFix repairs agent harnesses with an external repairer that always sees the traces (Chen et al., 2026a); in our loops, the optimizer's own modification decides whether a failure reaches it. Trivial-agent baselines (Zhu et al., 2025) and anytime-valid acceptance tests (Shawn, 2026) address evaluation validity; we show what a do-nothing floor does inside a self-modifying loop. Safety studies document misevolution under self-improvement (Shao et al., 2026; Jiang & Tang, 2026), and Wang et al. (2026) argue that safety is a matter of whether a system stays correctable; our results are measured instances of that separation on the capability side. Continual learning studies forgetting in weights and its regularization remedies (Kirkpatrick et al., 2017; Wang et al., 2024), loss of plasticity (Dohare et al., 2024), and collapse in self-training (Lin, 2026); Appendix I shows that the erosion and regularizer effects a single evaluation suggests in our channels lie within noise, and Appendix G gives an extended discussion.

9 Conclusion

Can self-evolving agents recover from their own mistakes? In the loops we studied, only under conditions that current frameworks do not guarantee. Without its archive, a workflow optimizer dismantled its agent while its validation score stayed at the initial agent's. Denser feedback prevented instructions from being lost, but once lost they returned only when their content was supplied from outside, as Theorem 1 predicts. A fault was detected only when its error text reached the optimizer, and the loop's own handler turned the fault's score drop into a gain; in self-training, a gate on accuracy missed a collapse to a constant answer. Self-evolving systems should be evaluated not only by how well successive versions perform, but by whether their mistakes remain detectable, revertible and repairable.

10 AI USE STATEMENT

Generative AI tools were used for coding assistance (writing and debugging evaluation and analysis scripts), for literature search, and for drafting the text of this paper. All AI-generated code was reviewed and tested by the authors; all numbers reported were produced by the authors' runs and checked against the raw evaluation files. We take responsibility for the final content of this work.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning, 2025. URL https://arxiv.org/abs/2507.19457.

Huan ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence, 2026. URL https://arxiv.org/abs/2507.21046.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable LLM agents: Diagnosing and repairing harness flaws, 2026a. URL https://arxiv.org/abs/2606.06324.

Shiqi Chen, Jingze Gai, Ruochen Zhou, Jinghan Zhang, Tongyao Zhu, Junlong Li, Kangrui Wang, Zihan Wang, Zhengyu Chen, Klara Kaleb, Ning Miao, Siyang Gao, Cong Lu, Manling Li, Junxian He, and Yee Whye Teh. Skillcraft: Can llm agents learn to use tools skillfully?, 2026b. URL https://arxiv.org/abs/2603.00718.

Zihan Chen, Songwei Dong, Chengshuai Shi, Peng Wang, Song Wang, Cong Shen, and Jundong Li. The past is prologue: A plug-in controller for selective updates in sequentially evolving llm memory, 2026c. URL https://arxiv.org/abs/2606.31121.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632: 768–774, 2024.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems, 2025. URL https://arxiv.org/abs/2508.07407.

Qisheng Hu, Quanyu Long, and Wenya Wang. When continual learning moves to memory: A study of experience reuse in llm agents, 2026. URL https://arxiv.org/abs/2604.27003.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2025. URL https://arxiv.org/abs/2408.08435.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024.

Hengle Jiang and Ke Tang. Why agents compromise safety under pressure, 2026. URL https://arxiv.org/abs/2603.14975.

Sihang Jiang, Lipeng Ma, Zhonghua Hong, Keyi Wang, Zhiyu Lu, Tengfei Wang, Shisong Chen, Jinghao Zhang, Tianjun Pan, Weijia Li, Jiaqing Liang, and Yanghua Xiao. Sea-eval: A benchmark for evaluating self-evolving agents beyond episodic assessment, 2026. URL https://arxiv.org/abs/2604.08988.

Thomas Joshi, Shayan Chowdhury, and Fatih Uysal. Swe-bench-cl: Continual learning for coding agents, 2025. URL https://arxiv.org/abs/2507.00014.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? A critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017. ISSN 1091-6490. doi: 10.1073/pnas.1611835114. URL http://dx.doi.org/10.1073/pnas.1611835114.

Chingkwun Lam, Jiaxin Li, Lingfei Zhang, and Kuo Zhao. Governing evolving memory in llm agents: Risks, mechanisms, and the stability and safety governed memory (ssgm) framework, 2026. URL https://arxiv.org/abs/2603.11768.

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

Jianzhe Lin. Self-improvement can self-regress: The rise-and-collapse failure mode of LLM self-training, 2026. URL https://arxiv.org/abs/2606.21090.

Siwei Liu, Jinyuan Fang, Han Zhou, Yingxu Wang, and Zaiqiao Meng. Sew: Self-evolving agentic workflows for automated code generation, 2026. URL https://arxiv.org/abs/2505.18646.

Gustav Olaf Yunus Laitinen-Fredriksson Lundstrom-Imanov. Mechanistic analysis of catastrophic forgetting in large language models during continual fine-tuning, 2026. URL https://arxiv.org/abs/2601.18699.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning, 2025. URL https://arxiv.org/abs/2308.08747.

Dang Nguyen, Viet Dac Lai, Seunghyun Yoon, Ryan A. Rossi, Handong Zhao, Ruiyi Zhang, Puneet Mathur, Nedim Lipka, Yu Wang, Trung Bui, Franck Dernoncourt, and Tianyi Zhou. Dynasaur: Large language agents beyond predefined actions, 2025. URL https://arxiv.org/abs/2411.01747.

Allen Nie, Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. The importance of directional feedback for LLM-based optimizers, 2024. URL https://arxiv.org/abs/2405.16434.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory, 2026. URL https://arxiv.org/abs/2509.25140.

Weijieying Ren, Xinlong Li, Lei Wang, Tianxiang Zhao, and Wei Qin. Analyzing and reducing catastrophic forgetting in parameter efficient tuning, 2024. URL https://arxiv.org/abs/2402.18865.

Shuai Shao, Qihan Ren, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, Dongrui Liu, and Jing Shao. Your agent may misevolve: Emergent risks in self-evolving llm agents, 2026. URL https://arxiv.org/abs/2509.26354.

Zayx Shawn. Pace: Anytime-valid acceptance tests for self-evolving agents, 2026. URL https://arxiv.org/abs/2606.08106.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025. URL https://arxiv.org/abs/2504.07952.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. Toward self-improvement of llms via imagination, searching, and criticizing, 2024. URL https://arxiv.org/abs/2404.12253.

Charles L. Wang, Keir Dorchen, and Peter Jin. Agentic safety is an epistemic property, not a behavioral one, 2026. URL https://arxiv.org/abs/2606.28347.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application, 2024. URL https://arxiv.org/abs/2302.00487.

Yingxu Wang, Siwei Liu, Jinyuan Fang, and Zaiqiao Meng. Evoagentx: An automated framework for evolving agentic workflows, 2025. URL https://arxiv.org/abs/2507.03616.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory, 2025. URL https://arxiv.org/abs/2511.20857.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022. URL https://arxiv.org/abs/2203.14465.

Weihao Zeng, Yuzhen Huang, Lulu Zhao, Yijun Wang, Zifei Shan, and Junxian He. B-star: Monitoring and balancing exploration and exploitation in self-taught reasoners, 2025. URL https://arxiv.org/abs/2412.17256.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. Evoflow: Evolving diverse agentic workflows on the fly, 2025a. URL https://arxiv.org/abs/2502.07373.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents, 2026a. URL https://arxiv.org/abs/2602.02474.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents, 2026b. URL https://arxiv.org/abs/2505.22954.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation, 2025b. URL https://arxiv.org/abs/2410.10762.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. Skillflow:benchmarking lifelong skill discovery and evolution for autonomous agents, 2026c. URL https://arxiv.org/abs/2604.17308.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners, 2024. URL https://arxiv.org/abs/2308.10144.

Yaoyue Zheng, Yin Zhang, Joost van de Weijer, Gido M van de Ven, Shaoyi Du, Xuetao Zhang, and Zhiqiang Tian. Revisiting weight regularization for low-rank continual learning, 2026. URL https://arxiv.org/abs/2602.17559.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

Yujun Zhou, Yue Huang, Han Bao, Kehan Guo, Zhenwen Liang, Pin-Yu Chen, Tian Gao, Werner Geyer, Nuno Moniz, Nitesh V Chawla, and Xiangliang Zhang. Alignment risks from capability-seeking rl training, 2026. URL https://arxiv.org/abs/2602.12124.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Jasjeet Sekhon, Jacob Steinhardt, Antony Kellermann, Sarah Schwettmann, Matei Zaharia, Ion Stoica, Percy Liang, and Daniel Kang. Establishing best practices for building rigorous agentic benchmarks, 2025. URL https://arxiv.org/abs/2507.02825.

A SETTING DETAILS

The *do-nothing floor* is the score of an agent that returns immediately, the trivial-agent baseline that Zhu et al. (2025) recommend reporting and found to be 38% on τ-bench airline; it is non-zero whenever a benchmark rewards final state rather than actions. On τ^2-Bench (Barres et al., 2025) airline and retail score a task by comparing the final database with a target, and a task whose expected actions contain no write is passed by an agent that does nothing. Ten of the twelve airline/simple evaluation tasks are of this kind (Table 5); we confirm the analytic floors with a workflow that terminates on its first step (0.833 ± 0.000 on airline/simple over five runs, 0.083 on retail/simple, 0.000 on telecom/simple), and use them throughout. Every airline/simple number we measured—the single-shot $57.1/64.3$ of the vanilla and regularized workflows, the repeated seed 0.733 and evolved 0.817, the template 0.708—is at or below 0.833.

Table 5: Do-nothing floor on τ^2-Bench evaluation slices, analytic from task criteria; **bold** entries were also measured with a workflow that returns immediately and match exactly.

	airline		retail		telecom	
	simple	complex	simple	complex	simple	complex
floor	**0.833** (10/12)	0.250 (6/24)	**0.083** (2/24)	0.031 (2/64)	**0.000** (0/20)	0.018 (1/57)

Loops and models. Workflow experiments use EvoAgentX's AFlow optimizer (Wang et al., 2025; Zhang et al., 2025b) on τ^2-Bench with GPT-5-mini as executor, GPT-5.1 as user simulator, and GPT-5-nano, GPT-5-mini or GPT-5.1 as optimizer; skill and memory experiments use a MemSkill-style bank (Zhang et al., 2026a) on MATH and LongMemEval (Wu et al., 2025) with GPT-5-nano and GPT-4o-mini; the model channel uses STaR self-training with LoRA (Zelikman et al., 2022) on MedMCQA with Qwen3-0.6B and Llama-3.2-3B. Table 6 lists every experiment with its loop and models; Appendix K–N give configurations and full tables.

Table 6: Experiments, loops and models. "Rewriter" is the component that modifies the state; executor and user simulator in the workflow experiments are GPT-5-mini and GPT-5.1 throughout. Open-weight models are served locally; Gemini and DeepSeek are accessed through an API.

§	Experiment	Loop / benchmark	Rewriters (seeds or trajectories)
B.1	accept-all vs. stock parent selection, 20 rounds	AFlow / τ^2 airline	GPT-5-nano (1+1); Gemini-2.5-Flash-Lite, DeepSeek-V4-Flash (2+1 each)
E.1	self-training at five intensities	STaR+LoRA / MedMCQA	Qwen3-0.6B (5; 3 per added intensity; 3 per arm), Llama-3.2-3B (5; 3)
5	repair under an unseen tool outage	AFlow / τ^2 airline, retail	GPT-5-nano, GPT-5-mini (8 cells; 12 reward trajectories); Gemini, DeepSeek (2 cells each)
C.1–C.3	rewrite arms and feedback ladder	prompt self-rewrite / probe	GPT-5-nano, GPT-5-mini (3); Gemini, DeepSeek (5); Qwen3-8B, Llama-3.1-8B (3; 5 with calibrated rules); Qwen3-1.7B, Llama-3.2-3B (5)
E.3	cross-domain chain; eviction under capacity	AFlow / τ^2; skill bank / LongMemEval	GPT-5.1; GPT-5-nano
App. I	audit of single-shot erosion	all three channels	GPT-5.1, GPT-5-mini; GPT-5-nano, GPT-4o-mini; Qwen3-0.6B, Llama-3.2-3B

B WORKFLOW OPTIMIZER: FULL RESULTS

B.1 WITHOUT ITS ARCHIVE, A WORKFLOW OPTIMIZER DISMANTLES ITS AGENT

Stock AFlow chooses the parent of each optimisation round by score-weighted sampling among the top-k rounds so far (Zhang et al., 2025b). This is usually described as exploration or as selection of better versions. It is also a revert path: a bad child stays in the archive and is never inherited from. We run the EvoAgentX implementation on τ^2 airline (complex split for training, six validation tasks, GPT-5-mini executor, GPT-5.1 user simulator, convergence check disabled) for 20 rounds with GPT-5-nano as the optimizer, in two arms that differ only in parent selection: *gated* (stock) and *accept-all* (parent = latest round). Frozen rounds are evaluated k=5 times on airline/simple and three

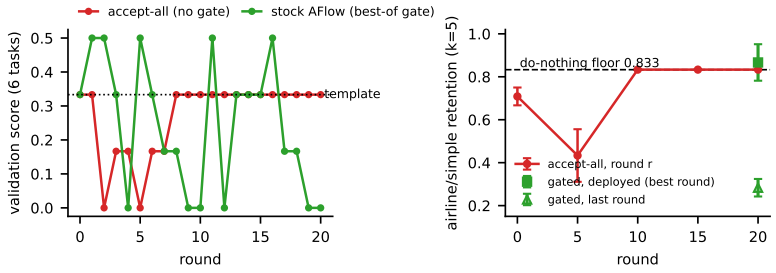

Figure 3: AFlow on τ^2 airline for 20 rounds with a weak optimizer (GPT-5-nano), stock best-of parent sampling versus accept-all (the accept-all arm's executor cost is in Figure 1a). Left: validation score on the six-task split; from round 8 the accept-all workflow's score equals the template's, and from round 10 it never calls an LLM. Right: airline/simple retention (k=5); the dead workflow sits exactly on the do-nothing floor, the blind round-5 workflow below it, the gated arm's deployed round above it, and the gated arm's *last* round collapses as well.

Table 7: Accept-all arm, round by round: validation score, executor cost of the validation run, and the optimizer's own description of its modification (abridged).

Round	Score / cost	Modification
0	.333 / $0.56	template
1	.333 / $0.46	fallback to `done()` if action extraction returns nothing
2	.000 / $0.06	"increase default maximum steps from 40 to 60" — and, undescribed, the prompt rewritten from 1846 to 1082 characters with all 8 placeholders removed
3	.167 / $0.05	"escape braces in the prompt for `str.format`" — two characters changed
4	.167 / $0.04	"graceful degradation: convert step exceptions into a final `done()` and break instead of raising"
5–9	.000–.333 / $0.04–0.08	action-extraction robustness, tool-availability check, max steps 60 → 50, tool ids in signatures
10	.333 / $0.00	"substitute LLM-driven prompt generation with a Custom operator" — calls `self.custom.custom(...)`, which does not exist; the exception is swallowed by round 4's handler
11–18	.333 / $0.00	max steps 50 → 60, history summary added then removed, history window shortened, None guards, `asyncio.wait_for` timeout, observation sanitisation
19–20	.333 / $0.00	optimizer output had a syntax error; original graph kept

times on retail/simple; for the gated arm we evaluate what AFlow would deploy (the best validation round) and, as a contrast, its last round.

Accept-all reaches an absorbing state in ten rounds. Table 7 lists what the optimizer did. Round 2's rewrite, which the optimizer describes only as raising the maximum number of steps from 40 to 60, replaces the prompt (1846 to 1082 characters) with a stub that keeps the output schema and a few rules and drops the task, observation, tool, policy and history sections: all eight context placeholders are gone, and the optimizer's own description of the round never mentions the prompt. Round 3, described as escaping braces for `str.format`, changes two characters. From round 2 on the agent sees no observation, tools, policy or history. Round 4 adds "graceful degradation": any exception in a step becomes a `done()` action and the loop breaks instead of raising. Round 10 replaces the model call with `self.custom.custom(...)`, a method that does not exist; the resulting `AttributeError` is swallowed by round 4's handler, every task terminates on its first step, and executor cost is $0.00 for rounds 10–20. The validation score is 0.333 throughout, identical to the template's, because two of the six validation tasks are solved by leaving the database alone. The optimizer receives "0.333 → 0.333" for ten rounds and keeps adding history windows, payload sanitisation and timeouts to a program that never runs a model (Figure 1a).

Retention reads the collapse as an improvement. The airline/simple trajectory is 0.708 (template) → 0.433 ± 0.12 (round 5: blind but active, *below* the floor because it acts on nothing) → 0.833 ± 0.00 (rounds 10–20: exactly the floor, with zero variance across five runs, the only reproducible agent on

Table 8: W2 evaluations (mean ± sd; zero infrastructure failures).

Artifact		airline/simple (k=5)	retail/simple (k=3)
template		0.708	0.825
accept-all, round 5	blind prompt, LLM still called	0.433 ± 0.123	—
accept-all, rounds 10 / 15 / 20	dead workflow	0.833 ± 0.000 each	0.083 ± 0.000
gated, deployed (round 1)	template + one fallback line	0.867 ± 0.085	0.917 ± 0.059
gated, last round (20)	placeholders replaced by literals	0.283 ± 0.041	—
do-nothing floor	analytic	0.833	0.083

this benchmark). On retail/simple the same dead workflow scores 0.083: a collapse from 0.825 that the airline split reads as a gain of 12 pp (Table 8).

The gate is the difference. The gated arm's optimizer produced equally broken children: rounds 4 and 10 cost $0 and score 0, and round 20 replaced the placeholders with literal strings and scores 0.283 ± 0.04 on airline. But its lineage tree has depth two—every parent is round 0, 1 or 2—and the broken children are orphans. Its deployed workflow, round 1, differs from the template by one fallback line and scores 0.867 ± 0.09 on airline and 0.917 ± 0.06 on retail (Figure 3, right). Same optimizer, executor, tasks and budget; the only variable is whether a rewrite can be undone.

Other optimizers. We repeated the comparison with two optimizers from other model families, Gemini-2.5-Flash-Lite and DeepSeek-V4-Flash (reasoning disabled; with reasoning enabled the optimizer prompt exhausts the output budget and returns no content), leaving the executor and the user simulator unchanged: two accept-all trajectories and one stock trajectory per optimizer, all reported (Table 2). Gemini reproduces the first run almost line for line. In trajectory a its first rewrite calls `self.custom.custom`, the same non-existent method GPT-5-nano wrote at round 10; every task crashes at its first step, the executor's cost falls to zero, and the validation score is 0.00 for seven rounds while the optimizer describes each rewrite as an improvement to JSON parsing. At round 8 it replaces the `raise` in the step handler with `action = "done()"`, commenting that this is "a fallback". Nothing is repaired—the agent still never calls a model—but the validation score returns to 0.33, the template's, and stays there for thirteen rounds, and retention on airline/simple goes from 0.000 at round 5 to 0.833 at rounds 10 and 20: exactly the do-nothing floor, above the working template (0.708), with retail at its floor of 0.083. A death that the score could see was converted, by the loop's own edit, into one it cannot. Trajectory b dies the same way at round 10, never writes the handler, and scores 0.00 on validation and on both retention splits for the remaining eleven rounds without being repaired. DeepSeek, a stronger optimizer, fails by accumulation rather than deletion. Under accept-all nearly every round adds a layer—self-consistency checks, self-ask reflection, ensembles, task classification, result verification—and after twenty rounds the workflow has grown from 7.8k to 20–30k characters and from zero to 14–20 model call sites per step, the executor costs five to six times as much, and the validation score has not moved outside its noise. One trajectory is still alive at round 20 with retention intact (0.778 airline, 0.812 retail); in the other, round 18 adds a pre-check that returns a reward of zero whenever the model's reply contains the string "complete", which "incomplete" also does, and the agent exits at step 0 on every task from then on. With stock parent selection neither optimizer's lineage contains any of this: Gemini's broken child of round 8 and DeepSeek's layered children of rounds 2, 5, 14 and 16 are orphans, cost stays at the template's, and retention on retail is 0.854 and 0.896 against 0.917 for GPT-5-nano. Three optimizers from three families destroy the agent in three different ways under accept-all, and the archive stops all three.

Three things this run shows. First, the optimisation signal was blind to the collapse: on a six-task validation set with two zero-write tasks, a dead workflow and the template are the same number, and the score never moved while the agent was dismantled. Second, the modifications that made it blind were the loop's own, and the loop did not report them: round 2's rewrite deleted the agent's entire context while describing itself as a change to the step limit, and round 4's handler turned an

Table 9: The fifteen conventions of the probe, as stated in the seed prompt, with their naturalness (spontaneous compliance of GPT-5-nano on 24 tasks when the rule is not stated).

Rule	Statement in the seed prompt	Naturalness
header	The first line must be exactly: ORDER REPORT	0.00
one_line	Each order appears on exactly one line.	1.00
sorted_date	List orders sorted by date, ascending.	0.96
title_case	Write customer names in Title Case (e.g. John Smith).	0.00
two_decimals	Write every amount with exactly two decimal places.	1.00
total_exists	Include a line that starts with 'Total:' giving the sum of listed amounts.	0.00
date_dmy	Write dates as DD.MM.YYYY (e.g. 14.03.2026), never ISO format.	0.00
prefix	Start every order line with the marker '¿¿ '.	0.00
rfd	Write the status 'refund' as 'RFD'; the word 'refund' must never appear.	0.21
total_first	Put the 'Total:' line immediately after the header, BEFORE the order lines.	0.00
pending_excl	Omit orders with status 'pending' from the list entirely; instead end the report with a line 'Pending: ¡count¿'.	0.00
separator	Separate the fields of each order line with ' — '.	0.92
eur	Convert every amount from USD to EUR at 0.90 and write it as '¡amount¿ EUR' (e.g. 12.34 EUR).	0.00
id_nohyphen	Write order ids without the hyphen (A-4821 becomes A4821).	0.00
initials	After each customer name, add their initials in parentheses, e.g. John Smith (JS).	0.00

interpretable failure (a crash, which would have scored 0 and told the optimizer something) into a constant, and the evaluation harnesses audited in Appendix I contain the same silencer, hand-written, in the form of infrastructure failures scored as zero. Third, the only thing that protected the stock run was the archive, which is never presented as a safety mechanism.

C Instruction-following probe: full results

C.1 A controlled instruction-following probe

Task and conventions. Following IFEval (Zhou et al., 2023), we specify the agent's behaviour through instructions whose compliance a program can verify. The agent turns a JSON list of orders into a text report, and its prompt states fifteen reporting conventions (Table 9): one order per line, omit pending orders and append a count, put the `Total:` line before the list, prefix every line with `>>` , convert amounts to EUR at a fixed rate, write names in title case with initials, sort by date, and so on. Each convention has its own checker over the input and the output, so for every report we know not only whether it is correct but which conventions it violates. The experiments below depend on this record: it identifies which instruction was lost and when, determines what the rewriter can be told about a failure, and supplies the exact text to re-supply after a loss. Some conventions describe what the model does unprompted; most contradict it. We measure each convention's *naturalness* as the executor's compliance when the convention is *not* stated: four are natural (0.92–1.00), one is partly natural (0.21), and ten are never followed unprompted (0.00). A lost counter-prior convention therefore cannot return through the model's default behaviour. The dependent variable is behaviour: per-rule compliance on 24 held-out reports, and the fraction that satisfy every convention. An LLM judge of whether each rule is still *present* in the prompt is recorded as an auxiliary signal only, because a rule can remain present while its meaning drifts.

Loop and arms. Each round the agent answers 6 training tasks and 24 evaluation tasks with its current prompt and then rewrites the prompt. The arms differ only in what the rewriter sees and whether anything can be undone: *control* (no rewrite; evaluation noise over rounds); *consolidate* (rewrite "to be clearer, better organized and more concise while preserving its intent"; no outcome information); *feedback* (rewrite given the six training inputs, the agent's outputs, and a per-task CORRECT/INCORRECT label, "so that future outputs are judged correct more often"); *feedback+spec* (as feedback, plus the full statement of each rule that failed on each task; §C.3 separates localisation from content); *feedback+gate* (as feedback, but the prompt reverts to the best-scoring prompt so far whenever the evaluation score drops). Executor and rewriter are the same model; GPT-5-nano is the weak rewriter and GPT-5-mini the strong one, three seeds per arm, 20 rounds (50 for the strong rewriter). An additional arm corrupts a fraction $p \in \{0.1, 0.3, 0.5\}$ of the training outputs (empty text labelled INCORRECT) to simulate infrastructure failures fed back as feedback. All runs use

Table 10: The probe with three rewriters (executor = rewriter = judge; all fifteen conventions; 20 rounds; temperature 0.7 for the two added models). Entries are seeds with a permanent loss (a convention the control arm follows whose mean compliance over the last five rounds is ≤ 0.10) and, in parentheses, mean compliance over all conventions at round $0 \rightarrow$ last five rounds. The gate for the added models decides on a separate six-task validation set; GPT-5-nano's gate row gives both variants (evaluation-set gate / validation-set gate). Seed all-rule pass rates are 0.45 (Gemini) and 0.38 (DeepSeek). Order lines are located by order id, hyphen-less id or its digits, so that a mutated id is counted once.

Arm	GPT-5-nano	Gemini-2.5-Flash-Lite	DeepSeek-V4-Flash
control (no rewrite)	0/3	0/5 (0.94 → 0.94)	0/5 (0.86 → 0.88)
consolidate (no feedback)	0/3	1/5 (0.93 → 0.71)	2/5 (0.87 → 0.63)
feedback (verdict only)	3/3	1/5 (0.95 → 0.87)	1/5 (0.87 → 0.87)
feedback + count	0/3	0/5 (0.93 → 0.91)	1/5 (0.87 → 0.90)
feedback + spec	0/3	0/5 (0.94 → 0.94)	0/5 (0.88 → 0.95)
feedback + gate	0/3 / 0/3	0/5 (0.94 → 0.94)	0/5 (0.86 → 0.95)

Table 11: Probe outcomes. "Collapse" means the all-rule pass rate reaches 0 and stays there for the remaining rounds; "lost" lists rules that disappear from behaviour and do not return within the run. Strong-rewriter numbers are at 20 rounds unless noted.

Rewriter	Arm	Peak	Final	Collapse	Lost rules
nano	control	0.65	0.51	0/3	—
nano	consolidate	0.79	0.67	0/3	—
nano	feedback	0.76	**0.00**	**3/3** (r2, r13, r19)	sorted_date, total_first; prefix; eur, initials
nano	feedback + spec	0.76	0.62	0/3	—
nano	feedback + gate	0.75	0.69	0/3 (11–17 reverts/run)	—
mini	control / consolidate / feedback	—	≥ 0.91 per rule	0/3 each	— (presence 15/15)
mini	feedback, 50 rounds	—	0.92, 0.79, **0.00**	1/3 (r44)	eur (over-specified)
mini	feedback, corrupted $p \leq 0.5$	—	per rule 0.75–0.99	0/9	— (prompts 8–12k chars)

explicit failure flags and large completion budgets; an early version of the probe collapsed from silent truncation alone and is archived, not reported (Appendix N).

C.2 A WEAK REWRITER WITH OUTCOME FEEDBACK LOSES RULES FOR GOOD

Table 10 reports the same arms for two further rewriters accessed through an API, Gemini-2.5-Flash-Lite and DeepSeek-V4-Flash (reasoning disabled), with five seeds per arm.

Improve, then lose. Under *feedback*, GPT-5-nano first improves itself: the all-rule pass rate rises from 0.3–0.6 at the seed to 0.8–0.9 within four rounds as the rewriter states the rules more clearly (Figure 2, Table 11). Then, in some round, a rewrite drops one or two rules, or restates a format as a different semantic field, and 7–18 further rounds of INCORRECT labels never restore them. All three seeds share this fate, at rounds 2, 13 and 19.

What is lost, per rule. The all-rule pass rate is an all-or-nothing aggregate, and it overstates the damage. In every collapsed run, 12–13 of the fifteen rules stay above 0.9 compliance after the collapse; the zero is produced by one or two rules that are lost and never recovered (seed 0: `total_first` then `sorted_date`; seed 1: `prefix`; seed 2: `eur` and `initials`). The absorbing state is not a loss of the task but a *permanent loss of a small part of the state that the loop cannot see*: the rewriter receives INCORRECT on every task, cannot tell that thirteen rules are fine, and cannot tell which two are not. We report per-rule loss as the primary outcome from here on.

Deletion is blind to naturalness; damage is not. The rewriter drops natural rules too—one-per-line vanishes from a prompt with no effect on behaviour, because the executor formats that way anyway—but only losing a rule against the prior changes the output: the executor then reverts to its default (total at the end, no prefix, amounts in USD, names as given). The prediction we pre-registered, that

rules survive in proportion to their naturalness, is wrong in its mechanism and right in its outcome: deletion rate does not depend on naturalness; the cost of deletion is borne entirely by the rules the model would not have followed on its own. This is what regression to the prior means operationally, and it is what a benchmark of generic competence cannot see: the idiosyncratic conventions go, the base model fills in the generic behaviour, and the average stays up.

Feedback is more dangerous than no feedback. *Consolidate*, which rewrites every round without any outcome information, never collapses; its cost is a mild drift on two under-specified rules (§C.4). Our second pre-registered prediction, that rewriting without feedback would erode as much as rewriting with it, is therefore also wrong, in the opposite direction. A rewriter that receives INCORRECT without knowing which rule failed edits whatever it can think of; each edit is lossy; the label stays INCORRECT; the loop is a positive feedback with an absorbing state. The two arms also differ in the rewrite instruction (consolidate is asked to preserve intent, feedback to raise the correctness rate), so we ran the two crossed controls. A consolidate arm given the feedback arm's objective, with no runs shown, never collapses ($0/3$; final pass 0.62); a feedback arm given the consolidate arm's objective, with the runs and verdicts shown, collapses in one seed of three (`separator` lost at round 1). The outcome information is what makes the rewrite lossy; the objective changes how often ($1/3$ against $3/3$).

Two interventions remove the absorbing state. Quoting the failing rule (*spec*) or allowing reversion (*gate*) both give $0/3$ collapses with peaks equal to the feedback arm's, and final pass rates of 0.62 and 0.69 against 0.00. The gate fires 11–17 times in 20 rounds, almost always on evaluation noise rather than on a real loss; a revert path does not need to be well calibrated to remove the absorbing state, it only needs to exist. This gate decides on the 24 evaluation reports it is scored on; a gate decided on a separate six-task validation set, with the evaluation set untouched, also gives $0/3$ permanent loss, with a final pass rate of 0.47 rather than 0.69 (§7).

C.3 Preventing a loss versus repairing it

The *spec* arm conflates two things: it tells the rewriter *which* rule failed, and it re-supplies the rule's content. We run a ladder of feedback arms that differ only in the information attached to each INCORRECT training task, holding the rewrite prompt fixed: verdict only (*feedback*); the number of conventions satisfied, e.g. "13/15" (*count*: denser signal, no identity); the failed conventions as stable anonymous identifiers, e.g. "#7, #12", with a seed-specific mapping the rewriter never sees (*anon*: identity, no content); the failed conventions' short names, e.g. `eur`, `total_first` (*name*: identity plus a hint); and the full statements (*spec*: identity plus content). We ask the question twice: from the seed (does the information *prevent* loss?) and from the three collapsed prompts of the feedback arm, continued for ten rounds under each arm (does it *repair* loss?).

Preventing loss needs only density; repairing it needs content (Table 3). From the seed, the lowest rung is enough: telling the rewriter that a task satisfied 13 of 15 conventions, with no identity at all, removes the loss in all three seeds ($0/3$ against $3/3$ for the verdict-only arm). Pooling the three arms that add information to the verdict (count, anon, name) gives $0/9$ against $3/3$ (Fisher two-sided $p = 0.0045$); the count arm on its own, at three seeds per arm, gives $p = 0.10$, so the density claim rests on the same pattern holding in every arm rather than on any single arm. The absorbing state of §C.2 is therefore manufactured, in part, by the all-or-nothing verdict: every training task looks completely wrong, and the rewriter treats a prompt that is 87% right as a prompt to overhaul. The mechanism is not smaller edits—consecutive-prompt similarity is 0.86–0.90 in every arm, and the *count* arm still makes wholesale rewrites—but what happens to a rule after a rewrite drops it. From a collapsed prompt the picture inverts. Neither the verdict, nor the count, nor a stable anonymous identity of the failing rule brings a lost rule back in ten rounds ($0/9$ pooling the three arms; $0/3$ each); the anonymous-id arm knows that "#7 keeps failing" for ten rounds and cannot act on it. Short names work only when they are self-describing (`total_first` and `sorted_date` are reconstructed; `prefix`, `eur` and `initials` are not), and only the full statement repairs all three ($3/3$, lost rules back to 0.88–1.00; $p = 0.0045$ against the pooled $0/9$). Once a rule has left the state, none of the outcome-feedback arms reconstructs it within ten rounds, nor within thirty ($0/5$ lost rules for count and for anon at round 30). The checker still holds the rule, so a rewriter that searched could in principle identify one from the count alone; we tested this with a search variant of the count and anon

Table 12: Strong rewriter (GPT-5-mini, 20 rounds, three seeds pooled): failure rate on the title-case and initials rules, split by whether the task contains a name with an apostrophe or hyphen.

Arm	ordinary names (14/24 tasks)	edge names (10/24 tasks)
control	0.006	0.131
feedback	0.004	0.282
consolidate	0.003	0.525

arms, which is told to change exactly one candidate convention per round and is shown a log of every earlier hypothesis with the mean number of satisfied conventions before and after it. In thirty rounds it recovers one of the five lost rules: `sorted_date`, found by the count-search arm at round 3 and stable thereafter (0.88–1.00); the anon-search arm reaches the same rule at round 15 and loses it again. The other four (`total_first`, `prefix`, `eur`, `initials`) are never proposed. Of 23–29 hypotheses per run, 15–25 are tie-breakers and blank lines; on a six-task batch the count moves by 0.5–1.1 conventions per hypothesis, so as many hypotheses "improve" it as "worsen" it, and the search circles. The one rule search recovers is the one that coincides with the executor's default (ascending order); none of the rules that cut against the prior is ever tried. The lost information is identifiable in principle, but a rewriter's proposals are drawn from its own prior, and what a lossy rewrite destroys is precisely what that prior does not contain. The anonymous-id arm also shows a third regime between drift and collapse: with identity but no content, several rules decay slowly to 0.5–0.8 over twenty rounds as the rewriter keeps "fixing" things it cannot understand.

C.4 A STRONG REWRITER CHANGES THE TIME SCALE AND THE FAILURE MODE, NOT THE OUTCOME

GPT-5-mini deletes nothing in 20 rounds: presence is 15/15 in every arm and per-rule compliance stays above 0.91. Its drift is subtler. The seed states two rules by example ("Title Case, e.g. John Smith"; "initials, e.g. John Smith (JS)"); the rewriter restates them as explicit algorithms whose handling of apostrophes and hyphens is its own convention, not the checker's. Failures therefore concentrate on the 10/24 tasks with names such as *o'brien* and *al-sayed*: the edge-name failure rate on these two rules is 0.13 under control, 0.28 under feedback and 0.53 under consolidate, while ordinary names fail at ≤ 0.006 in all arms (Table 12). Rules that are fully specified do not drift at all; rules with implicit edge semantics drift toward the rewriter's prior, and consolidate is worse than feedback because pure "tidying" has no error signal at all. At 50 rounds one of three seeds reaches the absorbing state: over forty rounds the EUR rule is "clarified" step by step into a program of integer arithmetic, at round 41 the presence judge no longer recognises it as the EUR rule, and at round 44 its output parts company with the original quantity and never returns. Two modes, one cause: a weak rewriter loses rules by deletion in rounds 2–19; a strong rewriter loses them by over-specification at round 44. Requests to be concise are ignored in the aggregate (prompts grow from 1.3k to 2.5–8k characters), and corrupting up to half of the training feedback inflates them further without losing any rule in 20 rounds.

C.5 WHEN IS A MISTAKE UNRECOVERABLE?

A mistake becomes permanent when a lossy rewrite is driven by feedback that does not carry what was lost, and nothing can be reverted. Removing any one factor removed the collapse in every seed we ran (Table 11; three seeds per arm, 20 rounds): no rewrite (control) or rewrite without outcome feedback (consolidate) does not collapse; feedback that carries the lost content (spec) or a revert path (gate) removes the collapse and reaches the feedback arm's peak. What the gate buys is the absence of permanent loss, not a higher final score: with a clean validation split its final pass rate (0.47) is no higher than that of the no-rewrite control (0.51). We state this as a condition observed under these budgets, with one ablation per factor, not as a proof that each factor is necessary. Density of feedback prevents loss but, under our rewriter, does not undo it (Table 3). The rewriter's strength sets the time scale and the failure mode, not the outcome.

Read as an answer to the question in the title, the condition says when a loop recovers from its own mistakes: when it can revert, or when its feedback carries the content that the mistake removed. Feedback that reports only that something failed, however densely, prevents new losses but does not undo old ones.

D REPAIR TEST: FULL RESULTS

The exception handler of round 4 is a modification that *raises* the current score and removes the next failure from the only signal the loop receives. It is a mistake the loop made for itself, and its cost appears only later, when something else goes wrong. Two agents with the same score today can differ entirely in what happens when the environment changes tomorrow: one can notice and fix the problem, the other has lost the error information, the history, or the structure that a fix would need. We test this directly: take frozen checkpoints, expose them to a fault they have never seen, and let the optimizer try to repair them under a fixed budget.

Design. The fault is an intermittent outage of the most-used read tool (`get_reservation_details` returns HTTP 503 with probability 0.5, raised as an exception), injected in the environment so that checkpoint code is untouched. Two checkpoints differ in exactly one thing: the template (exceptions propagate and the task scores 0) and the template plus round 4's handler (exceptions become `done()`), which we call the *silenced* checkpoint. Two optimizers (GPT-5-nano, GPT-5-mini) run six accept-all rounds from each checkpoint, under two error channels: *score* (the optimizer sees only validation scores, which is what stock EvoAgentX gives it, since it writes no failed-case log for τ^2) and *traces* (the fault events of each validation run, with the 503 message, are written to the log the optimizer samples). Detection is whether the optimizer's own description of a modification names the outage; repair is whether the diff installs a retry; we then evaluate the checkpoints and the repaired rounds under the fault on airline/simple (k=3) and retail/simple (k=2, fault on `get_order_details`). The blind checkpoints of §B.1 (rounds 2–4) never call a tool, so the fault cannot reach them; they are omitted.

Detection is decided by the channel, not by the optimizer. With the fault events visible, all four runs name the outage in their first modification; with the score alone, none of the four does in six rounds, whether or not the score moved (Table 4). The weak optimizer's response to an unexplained drop is to dismantle the agent (escape the prompt's braces, then replace the call loop with a stub); the strong one's is to tinker with action parsing. Stock EvoAgentX on τ^2 is in the score column by construction.

The same holds for optimizers from other families. We ran the template cell with Gemini-2.5-Flash-Lite and DeepSeek-V4-Flash as the optimizer, under both channels. With the fault events visible, both name the 503 in their first modification and spend every later round installing or rewriting a retry, which brings detection under the traces channel to seven runs of seven across four optimizers. With the score alone, DeepSeek produces six valid modifications—a self-consistency check, a reflection loop, a tool-use guard, a final validation, a self-ask cross-check, a task decomposition—none of which mentions the outage or adds a retry, while the executor's cost rises twelvefold and the validation score falls from 0.17 to 0.00: five valid score-channel runs, no detection. Gemini's score-channel run produced no loadable child in six rounds (it calls an operator that does not exist), and DeepSeek's first pass, with reasoning enabled, returned no content; we count neither as a failure to detect. Detection does not tell us whether the repair works. The validation score is 0.17 in all seven rounds of DeepSeek's traces run, yet retention under the fault separates its repairs by the layer they act on: a retry wrapped around the environment step scores 0.611 on airline and 0.625 on retail, against 0.389 and 0.354 for the crashing template, while retries placed in the model's prompt or the observation handler, which is where Gemini and DeepSeek's second pass put them, score 0.36–0.44 and 0.29–0.33 (k=3 and 2 evaluations per cell).

The silencer cuts the score channel and is rewarded for it. Under the fault, the template scores 0.389 on airline/simple with 5–8 of 12 tasks crashing; the silenced checkpoint scores 0.833 with zero crashes, above the no-fault template (0.708), because every 503 becomes an immediate `done()` and lands on the floor. From the score, the silenced agent looks *better* under an outage than the working agent does without one. On retail/simple, where the floor is 0.083, the same silencer scores 0.375, no

Table 13: Does the floor select giving up? GPT-5-mini optimizer, traces channel, six accept-all rounds under the outage, three trajectories per cell. *state* is the stock τ^2 reward; *no floor* scores every task in which the agent transferred or never called a write tool as 0. Round-6 agents are evaluated under the fault on airline/simple (k=3; floor 0.833) and retail/simple (k=2; floor 0.083), one entry per trajectory in a fixed order; "gave up" is the share of retail episodes with a transfer or no write tool call. A *pure give-up* agent never writes and never transfers; a dead run never reaches the environment.

Checkpoint	Reward	pure give-up / dead	airline	retail	gave up (retail)
template	state	1 / 0	0.61, 0.19, 0.11	0.08, 0.46, 0.31	100, 37, 20%
template	no floor	1 / 0	0.11, 0.03, 0.42	0.46, 0.21, 0.08	29, 63, 100%
silenced	state	1 / 1	dead, 0.64, 0.25	dead, 0.06, 0.29	—, 100, 38%
silenced	no floor	0 / 0	0.36, 0.17, 0.31	0.60, 0.27, 0.27	38, 69, 50%

better than the crashing template. This is the decoupling in its purest form: the modification raised the current score and erased the fault from it. Even before the erasure, the score alone was not enough: with the drop to 0.389 visible, neither optimizer found the fault in six rounds (Table 4, score rows), so what the handler removes is a signal that was necessary but not sufficient.

Under a high floor, giving up is a score-optimal repair; the floor rewards it without selecting it. GPT-5-mini reads the 503 correctly and, on airline, responds by giving up (transfer to a human, otherwise `done()`); its validation score rises to 0.333, the floor, and tool calls collapse. This is the score-optimal response on that split: the one correct repair we observed—a retry wrapped around `env.step`, installed by GPT-5-nano at round 2 on the silenced checkpoint—scores 0.889 on airline, within noise of the silencer's 0.833, but 0.667 on retail against the silencer's 0.375, recovering 80% of the no-fault level. A benchmark's floor is therefore not only a measurement problem: it makes the repair that abandons the task at least as good, by the score, as the one that restores it. Whether the score *causes* the choice is a separate question, and the run above is one trajectory. We repeated the strong optimizer's cell three times from each checkpoint under the stock reward and three times under a reward in which giving up earns nothing: any task in which the agent transfers, or terminates without ever calling a write tool, scores 0 (Table 13). From the template under the stock reward, one trajectory in three converges to a *pure give-up* agent that never writes and never transfers, and two install a retry around the environment step and keep acting; under the no-floor reward the split is the same, one pure give-up agent and two that act, although the give-up agent's validation score was 0 in every round on which it was chosen. From the silenced checkpoint, the stock reward gives one dead run, one pure give-up agent and one that acts; the no-floor reward gives three that act. Pooled over both checkpoints, pure give-up agents are $2/6$ under the stock reward and $1/6$ without the floor; with three trajectories per cell this is no difference, and the choice looks like the optimizer's own rather than the score's. What the replication does establish is the pattern behind the single run, on twelve agents and two domains: under the fault, the agents that give up score highest on airline (0.42–0.64, against 0.03–0.36 for the agents that act) and lowest on retail (0.06–0.08, the floor, against 0.21–0.60). A benchmark with a high do-nothing floor rewards the give-up repair at evaluation time whether or not it induced it, and the agents that act recover 0.21–0.60 on retail against 0.354 for the crashing template.

What repairs is a silencer plus a channel. The only checkpoint that was repaired and survived is the silenced one under the traces channel: the handler keeps the executor alive while the retry is being installed, and the log tells the optimizer what to install. The unsilenced template under traces put its retry in the wrong layer and died two rounds later. The error channel therefore has two parts that current frameworks conflate: whether a failure is fatal to the *executor*, and whether it is visible to the *optimizer*. Graceful degradation is right for the first and wrong for the second; the accept-all run of §B.1 did the first, and its score never noticed. The second was never present in the stock framework, which writes no error text for its optimizer; supplying only that—the error text in the optimizer's log, nothing else—yields detection in every run and correct repair in one. What these runs do not show is a channel that worked before a self-modification and was broken by it: detection is identical for the template and the silenced checkpoint under each channel, so the handler's effect is on the score, not on a diagnostic path the optimizer ever had. Showing the causal version requires a checkpoint that is still alive after the handler was installed; the accept-all trajectory does not contain one, because its placeholders were deleted a round before the handler was written. We looked for one in three

Table 14: Answer collapse in the parametric channel (MedMCQA, anatomy evaluation after each stage of anatomy → biochemistry → dental). "Top share" is the fraction of the 2000 questions that receive the model's most frequent letter; "lift" is accuracy minus that of a question-independent answerer with the same letter marginal; accuracy under the lenient extractor. The best constant answerer scores 0.294 (a); d has base rate 0.197. Qwen settings D1–D4 are cumulative: four STaR iterations per stage; learning rate ×4; LoRA rank ×4; no carry-over. "Base" is the training intensity of Appendix I (one iteration per stage). Mean ± sd over 5 seeds (base) or 3 (D1–D4).

Setting	iters/stage	top share, after stage 1 / 2 / 3	lift, 1 / 2 / 3	acc., 1 → 3
Qwen3-0.6B				
no fine-tuning	—	d 0.65	0.02	0.24
base, vanilla	1	0.45 ± 0.05 / 0.37 ± 0.05 / 0.57 ± 0.03	0.07 / 0.07 / 0.05	0.31 → 0.32
base, L2	1	0.44 ± 0.05 / 0.44 ± 0.05 / 0.45 ± 0.06	0.07 / 0.07 / 0.07	0.31 → 0.31
D1	4	0.51 ± 0.05 / 0.90 ± 0.03 / **0.95 ± 0.01**	0.07 / 0.02 / **0.01**	0.33 → 0.30
D2: + lr ×4	4	0.55 ± 0.15 / 0.73 ± 0.16 / 0.81 ± 0.12	0.05 / 0.05 / 0.03	0.31 → 0.32
D3: + rank ×4	4	0.54 ± 0.10 / 0.56 ± 0.10 / 0.68 ± 0.14	0.04 / 0.04 / 0.04	0.28 → 0.30
D4: + no carry-over	4	0.75 ± 0.20 / 0.97 ± 0.03 / **0.99 ± 0.02**	0.03 / 0.00 / **0.00**	0.27 → 0.26
Llama-3.2-3B				
no fine-tuning	—	a 0.31	0.45	0.70
base, vanilla	1	0.31 ± 0.01 / 0.31 ± 0.01 / 0.38 ± 0.06	0.43 / 0.42 / 0.40	0.69 → 0.66
base, EWC	1	0.31 ± 0.01 / 0.31 ± 0.01 / 0.41 ± 0.03	0.44 / 0.43 / 0.40	0.70 → 0.66
base, L2	1	0.31 ± 0.01 / 0.31 ± 0.01 / 0.31 ± 0.01	0.44 / 0.44 / 0.44	0.69 → 0.69

further stock accept-all trajectories from the template, two with GPT-5-nano for eight rounds and one with GPT-5-mini for sixteen, scanning every round for a handler around the environment step that does not re-raise. The weak optimizer wrote such a handler only after it had already deleted the prompt's placeholders (round 6 of one trajectory, as at round 4 in §B.1); the strong optimizer never wrote one, and spent sixteen rounds on action parsing while its agent stayed alive. The state the paired test needs—an agent silenced but otherwise intact—did not arise on its own in these runs, which is why the silenced checkpoint above is the template with round 4's handler transplanted, and why the paired comparison—before the modification, after it, and after restoring only the channel, under one log mechanism—is left open here.

E Other loops: full results

E.1 Weights: self-training converges to a constant answer at the score's floor

The mistake of Section 3, a loss the score cannot see, also appears where the rewrite is a gradient step and the feedback a correctness filter. STaR self-training (Appendix M) generates rationales for the stage's training questions, keeps those whose final letter is correct, and fine-tunes on them; the loop sees nothing but the filter's yield. On MedMCQA the most common gold letter is a, at 29.4% of anatomy questions, so an agent that answers a to everything scores 0.294: a do-nothing floor of the same kind as τ^2 airline's. We ran the pipeline on Qwen3-0.6B at four training intensities beyond the base one of Appendix I (four STaR iterations per stage instead of one, then a higher learning rate, a larger LoRA rank, and no carry-over of earlier iterations' data; three seeds each), looking for the intensity at which anatomy accuracy erodes. It never does by more than noise (Table 14). What happens instead is that the answer stops depending on the question. We measure this as the share of evaluation questions that receive the model's most frequent letter, and as the *lift*: accuracy minus that of a question-independent answerer with the same letter marginal. At the base intensity the untrained model already answers d on 65% of anatomy questions with a lift of 0.02; the round-one adapters have a lift of 0.07 at every stage, and their top-letter share drifts from 0.45 to 0.57 over the chain. With four iterations per stage, all three seeds converge to a on 94–96% of questions by the third stage, the lift falls from 0.07 to 0.01, and anatomy accuracy ends at 0.297, the floor. The most aggressive setting collapses in the first stage: one seed answers d on 98–100% of questions from its first evaluation on and scores 0.195, the base rate of d.

The mechanism is the filter. A model that answers a to everything is "correct" on 29% of its rationales, all of which end in a, and the next update makes a more likely; in the self-generated data the top letter's share among the kept rows is at or above its share among the generated rows

Table 15: What stops the collapse. Qwen3-0.6B at four STaR iterations per stage, three seeds per arm; anatomy test set (n=2000) after each stage, and the trained domain's test set after its own stage. "Kept share" is the share of rationales in the last stage's training sets that end in the majority letter. Gates compare a 500-item development probe with the last accepted checkpoint. Llama-3.2-3B is shown under the outcome filter at the same intensity.

Arm	Feedback / gate	Reverts	Top share stage 1 / 2 / 3	Lift	Anatomy acc.	Biochem., dental acc.	Kept share
outcome	correctness filter	—	0.51 / 0.90 / **0.95**	0.01	29.7	33.7, 31.2	0.90
rationalisation	filter + gold-hinted rationales for wrong items	—	0.55 / 0.49 / **0.41**	**0.06**	33.0	**42.9**, **36.9**	**0.33**
gate on accuracy	revert if dev. accuracy drops > 1.5 pp	2, 4, 0	0.57 / 0.75 / 0.91	0.00	29.4	34.4, 30.7	0.86
gate on lift	revert if dev. lift drops > 0.02	4, 7, 8	0.45 / 0.59 / 0.65 $\pm$ 0.21	0.06	32.9	38.6, 31.6	0.66
Llama-3.2-3B, outcome	correctness filter	—	0.39 / 0.40 / 0.43	0.41	67.6	77.4, 55.1	—

in nearly every iteration, and the filter's yield, the only number the loop sees, stays at 0.29–0.34 throughout (Appendix M). Two attractors appear, the base model's own letter (d, which 63% of its first-iteration rationales end in) and the gold majority (a); every collapsed seed lands on one of them. Nothing in the loop distinguishes 29% correct by reasoning from 29% correct by constancy, and the pre-registered erosion criterion cannot see it either, because a 0.6B model's accuracy is at the floor to begin with. Llama-3.2-3B, at the base intensity, shows the beginning of the same drift: its top-letter share on anatomy moves from 0.31 to 0.38 $\pm$ 0.06 (vanilla) and 0.41 $\pm$ 0.03 (EWC) over the chain while its lift falls from 0.44 to 0.40 and its accuracy by 2–3.6 pp, with nothing learned on the new domains; the L2 penalty, which prevents learning, also holds the share at 0.31. This is the loss of §B.1 in the parametric channel: a self-modification that improves nothing and destroys the dependence of the output on the input, held invisible by a score that the floor keeps up.

What stops it. We ran three further arms at the four-iteration intensity, three seeds each, and evaluated every arm on the 2000-item anatomy test set after each stage (Table 15). *Rationalisation*, STaR's own remedy for wrong answers—the model is shown the gold letter, asked to write a rationale for it, and the result joins the training set—is feedback that carries the content. It stops the collapse in every seed (top-letter share 0.41 after the third stage against 0.95; lift 0.06 against 0.01), and it is the only arm that learns the new domains (42.9% on biochemistry and 36.9% on dental against 33.7% and 31.2% for the outcome filter). The mechanism is visible in the training data: under the outcome filter the share of kept rationales that end in the majority letter rises from 0.53 to 0.90 over the chain, while with rationalisation it stays at 0.29–0.40, because every question contributes a rationale whatever letter the model currently favours. Rationalisation also quadruples the training set, and the two effects are not separated here. A revert gate on development accuracy—discard the iteration when accuracy on a 500-item anatomy development set falls by more than 1.5 pp—fires two, four and zero times in the three seeds and stops nothing (top-letter share 0.91, lift 0.00): the collapse never moves the number it watches. The same gate on development lift fires four to eight times and slows the collapse without ending it (top-letter share 0.85, 0.66 and 0.43 in the three seeds; lift 0.023, 0.068 and 0.080). Llama-3.2-3B at the same intensity does not collapse: over twelve iterations its top-letter share moves from 0.39 to 0.43, its lift stays at 0.41 and its anatomy accuracy at 67%, and our pre-registered prediction that it would collapse failed. The difference is the floor. At 67% accuracy a constant answer would cut the filter's yield to 29%, so the outcome filter does carry the information a collapse would destroy; at Qwen's 31%, one point above the floor, it carries nothing, and the loop cannot tell a model that reasons from one that answers a.

E.2 Open-weight rewriters: richer feedback is not uniformly protective

The arms identify the feedback and gate factors only if the verdict and the gate's score can move: the seed prompt's all-rule pass rate must be off the floor, and the executor's output must be a report. We check both for every rewriter before reading its arms (seed pass rates: GPT-5-nano 0.3–0.6, Gemini-2.5-Flash-Lite 0.45, DeepSeek-V4-Flash 0.38). Open-weight models at 8B and below fail the first check with the full rule set, so we ran them twice: once with all fifteen conventions, reported at the

Table 16: The probe with four open-weight rewriters on per-model calibrated rule sets (executor = rewriter = judge; five seeds per arm; 20 rounds; temperature 0.7). The rule set keeps the conventions with control-arm compliance ≥ 0.90 in a five-round calibration run; its size and the seed all-rule pass rate are given under each model. Entries are seeds with a permanent loss (a kept convention whose mean compliance over the last five rounds is ≤ 0.10) and, in parentheses, the all-rule pass rate at round $0 \rightarrow$ last five rounds. The gate decides on a separate six-task validation set; for the Llama models a second gate compares mean per-rule compliance instead of the all-rule pass rate. The last row counts, among the runs with a loss, those whose round-20 output is no longer a report (a program, or a markdown document). Order lines are located as in Table 10.

Arm	Qwen3-8B 8 rules, seed 0.89	Llama-3.1-8B 7 rules, seed 0.78	Qwen3-1.7B 4 rules, seed 0.97	Llama-3.2-3B 6 rules, seed 0.93
control (no rewrite)	$0/5$ ($1.00 \rightarrow 1.00$)	$0/5$ ($0.97 \rightarrow 0.97$)	$0/5$ ($0.95 \rightarrow 0.95$)	$0/5$ ($0.94 \rightarrow 0.91$)
consolidate (no feedback)	$1/5$ ($1.00 \rightarrow 0.78$)	$5/5$ ($0.95 \rightarrow 0.00$)	$0/5$ ($0.94 \rightarrow 0.86$)	$5/5$ ($0.97 \rightarrow 0.00$)
feedback (verdict only)	$0/5$ ($1.00 \rightarrow 1.00$)	$2/5$ ($0.99 \rightarrow 0.58$)	$0/5$ ($0.96 \rightarrow 0.78$)	$2/5$ ($0.93 \rightarrow 0.34$)
feedback + count	$0/5$ ($1.00 \rightarrow 0.87$)	$4/5$ ($0.98 \rightarrow 0.20$)	$1/5$ ($0.94 \rightarrow 0.75$)	$3/5$ ($0.93 \rightarrow 0.15$)
feedback + spec	$0/5$ ($1.00 \rightarrow 0.99$)	$3/5$ ($0.97 \rightarrow 0.32$)	$0/5$ ($0.96 \rightarrow 0.82$)	$1/5$ ($0.88 \rightarrow 0.71$)
feedback + gate (all-pass)	$0/5$ ($1.00 \rightarrow 0.97$)	$0/5$ ($0.95 \rightarrow 0.98$)	$0/5$ ($0.96 \rightarrow 0.93$)	$0/5$ ($0.90 \rightarrow 0.93$)
feedback + gate (mean rule)	—	$0/5$ ($0.98 \rightarrow 0.99$)	—	$0/5$ ($0.91 \rightarrow 0.82$)
lost runs, output not a report	$0/1$	$11/14$	$1/1$	$11/11$

end of this section as a description, and once with a per-model *calibrated* subset of the conventions the model already follows, which puts the seed pass rate at 0.78–0.97 and makes the arms readable (Table 16; Appendix N).

With calibrated rule sets. Three things hold across the four models. The control arm loses nothing, so the calibrated conventions are held; and the gate arm loses nothing—$0/5$ in every model, with either gate score, at 0–13 reverts per run—so that over the six rewriters of Tables 10 and 16 a permanent loss has occurred in 0 of 46 gated runs. The two Qwen models behave like the strong rewriter of §C.4: Qwen3-8B loses one convention in one consolidate seed (the placement of the `Total:` line, deleted from the prompt at round 1) and nothing in any feedback arm; Qwen3-1.7B, the weakest rewriter we ran, loses nothing in the feedback arm in 20 rounds, against the prediction we registered before the runs that it would lose in at least three seeds of five, and loses conventions in one count seed, when its output turns into a markdown document; one of its feedback seeds settles at an all-rule pass rate of 0.2–0.3 from round 1 with no single convention below 0.10, a diffuse degradation the loss criterion does not count. The two Llama models lose in every rewriting arm except the gate, and what they lose is again mostly the frame: in 22 of their 25 lost runs the round-20 output is a Python or JavaScript program, or a markdown document with headings and tables, in place of the report, and in the feedback arms the lost conventions are still stated in the prompt. The ordering of the arms is not the one under GPT-5-nano: rewriting *without* feedback is the most destructive arm on both Llama models ($5/5$ each; the prompt shrinks to 142–568 characters and most of the conventions are deleted from it), and the spec arm does not prevent the loss on Llama-3.1-8B ($3/5$)—the statement of a failed convention says nothing about the output having become a program, and in the two of those runs where the report survives, a separator convention deleted from the prompt and a date format that breaks the decimals check are not restored by it either. The frame loss is visible to the gate, whose score drops to 0 when the output becomes a program, and to nothing else in the loop: the verdict is INCORRECT either way, and the count and the spec report failed conventions, not a changed kind of output.

At the floor: the full rule set. We ran the five original arms with Qwen3-8B and with Llama-3.1-8B-Instruct as executor, rewriter and judge (three seeds, 20 rounds, temperature 0.7; Appendix N). At this scale the seed prompt's all-rule pass rate is 0.00–0.08, so the verdict is INCORRECT on every task from round 0 and the all-pass gate never fires for Llama (0 reverts, since it compares 0 with 0): the feedback and gate factors are not identified by this design, and we report these runs as a description, not as a test of the condition. Qwen3-8B loses none of the 13 rules it follows in any arm, like the strong rewriter above; its consolidate arm reaches a fixed point after one rewrite. Llama loses something else. In 8 of 12 rewriting runs (consolidate $3/3$, spec $2/3$, feedback and the inert gate $3/6$; control $0/3$) a rewrite changes the frame of the prompt—in seven of the eight the role sentence "You generate a daily orders report" becomes the imperative "Generate a daily orders report from a JSON

Table 17: Top: W1 cross-domain chain, retention on the three simple splits after each stage (k=5) and similarity of the evolved artifact to the template. Bottom: S4, LongMemEval skill channel at bank capacity 10 under four eviction policies (churn = fraction of facts questions that flip between the facts-stage and updates-stage banks; rebuild floor 0.24).

	template	after airline	after retail	after telecom
airline/simple	0.708	0.817	0.800	0.817
retail/simple	0.825	0.850	0.892	0.850
telecom/simple	0.430	0.450	0.430	0.390
trained complex	—	0.375	0.542	0.140
prompt.py sim. to template	1.00	1.00	1.00	0.73

Eval @ bank	cap 30	LFU 10	FIFO 10	rand. 10	LRU 10
facts @ facts	0.381	0.460	0.492	0.381	0.397
facts @ updates	0.429	0.492	0.429	0.413	0.492
updates @ updates	0.469	0.500	0.500	0.500	0.500
facts skills surviving	19/21	8/10	**0/10**	1/10	5/10
churn facts→updates	0.30	0.32	0.29	0.35	0.29

list", sometimes inside a code fence—and the executor answers with a Python program that would produce the report instead of the report, on 96–100% of tasks, from that round to the end of the run (14–19 rounds). Between 7 and 14 of the fifteen rules are still stated in seven of those prompts (the eighth ends as a 321-character description of a Python function), and every "lost" rule but one in these runs is lost this way. No arm's feedback says that the output is a program: the verdict was already INCORRECT, the spec arm lists the rule statements that the program's text fails, and the gate sees a score that was on the floor before and after. The structure is the one above—a lossy rewrite, feedback that does not carry what was lost, a gate the floor has blinded—with the task frame in place of a rule.

E.3 SKILLS: WHEN NOTHING CAN BE LOST

Two experiments on the testbeds of Appendix I were designed to create the distribution-shift and capacity-pressure conditions under which continual-learning intuition predicts erosion; both show none, and in both the reason is that the rewrite is not lossy. *Cross-domain workflow chain.* One workflow evolves through airline → retail → telecom on τ^2 (four rounds per stage, GPT-5.1 optimizer, chained by the last round so that best-of cannot intervene) and retains airline/simple at $0.817 \rightarrow 0.800 \rightarrow 0.817$ (k=5); the prompt after the first two stages is byte-identical to the template, and the trained complex splits reach only $0.375/0.542/0.140$ (Table 17, Appendix K.5). A strong optimizer's rewrites under this framework are near-identity: nothing is lost, and nothing is learned. *Eviction under capacity.* LongMemEval skill banks capped at ten entries under LFU, FIFO, random and LRU eviction all keep facts accuracy within the rebuild floor; FIFO replaces all ten facts-stage skills and facts accuracy moves by four questions out of 63 (Table 17). The no-memory baseline is 0, so the memory mechanism is the capability, but *which* skills populate the bank does not matter: what is rewritten carries nothing.

F EXTENDED DISCUSSION

F.1 EVALUATION SHAPES THE LOOP

The do-nothing floor is not only the measurement artifact that benchmark audits report it as (Zhu et al., 2025). A benchmark that rewards final state rewards inaction on every task whose target is the initial state, and under such a benchmark giving up is, by the score, a repair: GPT-5-mini read a 503 correctly and chose to transfer to a human or terminate, which is score-optimal on airline (floor 0.833) and catastrophic on retail (floor 0.083). Repeating the cell with the floor removed from the reward shows that the score did not drive the choice: one trajectory in three gives up either way (§5); the floor's effect is at evaluation, where it ranks the give-up agent above every agent that acts. The same holds for silent failure scoring: a harness that scores crashes as zero, and an executor that converts crashes into graceful exits, both remove the one signal that would have told the optimizer what to fix, and the second is something the optimizer will write for itself if the first does not. Reporting practice follows. Every retention number needs its repeat, rebuild or seed floor and the dependence

and do-nothing floors of its split; a single evaluation of twelve tasks has an MDE of $19\,\mathrm{pp}$ and should not be reported as a comparison; failures must be counted separately; the deployed artifact and its diff from the seed must be shown, since five of six "evolved" workflows behind our single-shot tables were the seed; and per-task outcomes must be recorded, because net accuracy hid churn of 24–55% in every channel we measured.

F.2 Limitations

The workflow experiments use GPT-5-mini as executor throughout, with the optimizer varied over three families; the probe's conventions are synthetic; and the repair test uses one fault type and one framework, with three seeds per arm in the probe and single runs per cell in the repair test, except the strong optimizer's give-up cell, which has three trajectories per reward condition (Table 13). For open-weight rewriters the probe is identified only on a per-model calibrated set of four to eight conventions (§E.2), so their arms are not on the same rule set as the API models', and the loss they show is mostly a change in the kind of output rather than of one convention; in the parametric channel the arms of Table 15 test the content and revert factors on one small model, with rationalisation's larger training set left unseparated from its content, and the loss factor was not varied. The memory-retrieval variant of the skill channel was audited through a dependence check on the same benchmark family rather than a re-run of the exact pipeline. The do-nothing floor is a property of state-based rewards; other benchmarks may have analogous floors we have not computed. Finally, our condition is stated as three factors with one ablation each at three seeds per arm; it is not a sufficiency result, the necessity of each factor rests on those three seeds, and the strong-rewriter collapse at round 44 shows that "eventually" can be long. Two design choices in the probe deserve note. The consolidate and feedback arms differ in their rewrite objective as well as in the information they receive; the crossed controls of §C.2 show that the information drives the loss and the objective its rate. The gate arm of Table 11 reverts on the same 24 evaluation reports on which its outcome is reported; a gate decided on a separate six-task validation set gives the same $0/3$ permanent loss with a final pass rate of 0.47 against 0.69 (9–18 reverts in 20 rounds), which is the part of the original number that selection on evaluation noise had inflated. "Permanent" and "never recovers" throughout refer to the remaining rounds of a 20-round run and a 10-round repair budget, not to an unbounded horizon.

G Extended related work

Self-evolving agents. Agents that optimize their own workflows (Hu et al., 2025; Zhang et al., 2025b; Wang et al., 2025; Zhang et al., 2025a; Liu et al., 2026), skills and tools (Wang et al., 2023; Zhao et al., 2024; Nguyen et al., 2025; Zhang et al., 2026a; Chen et al., 2026b), memories (Suzgun et al., 2025; Wei et al., 2025; Ouyang et al., 2026), and weights (Zelikman et al., 2022; Zeng et al., 2025; Tian et al., 2024) are surveyed by ang Gao et al. (2026) and Fang et al. (2025). Almost all of this work measures forward progress on new tasks. Retention has entered the literature recently: SkillFlow (Zhang et al., 2026c) benchmarks lifelong skill evolution and finds that most models cannot evolve stably; SEA-Eval (Jiang et al., 2026) adds a stability dimension to evolutionary gain and warns that success rate alone creates a capability illusion; Hu et al. (2026) argue that external memory relocates rather than removes the stability–plasticity dilemma; and Chen et al. (2026c) propose selective-update controllers for evolving memories. We are not aware of work that asks whether such loops recover from their own mistakes, or that separates preventing a loss from repairing it.

Self-correction. Whether a language model can correct its own mistakes has been studied at the level of a single response: without external feedback, self-correction of reasoning often fails to improve answers and can degrade them (Huang et al., 2024), and a survey of the area concludes that reliable self-correction generally requires external feedback or verification (Kamoi et al., 2024). Our setting differs in two ways: correction happens across rounds of self-modification rather than within one response, and the mistake can alter the machinery—the prompt, the feedback channel, the error handling—that later correction relies on. Our finding that lost content returns only when it is supplied from outside is the analogue, for self-modified state, of their conclusion for reasoning.

Continual learning. Catastrophic forgetting is defined for weights, and its remedies fall into regularization (EWC and its relatives, Kirkpatrick et al., 2017), replay, and parameter isolation (Wang et al., 2024). Empirical and mechanistic studies of forgetting during LLM fine-tuning (Luo et al.,

2025; Ren et al., 2024; Lundstrom-Imanov, 2026; Zheng et al., 2026) likewise assume a parametric state, and continual-learning benchmarks for code agents (Joshi et al., 2025) measure weight-level forgetting. A second line studies *loss of plasticity*, in which a network trained continually loses the ability to learn at all (Dohare et al., 2024); it is the closest relative of our reframing, but it locates the loss in the optimisation properties of the weights, whereas we locate it in what the loop's feedback can carry and what its update can undo. Lin (2026) report a rise-and-collapse pattern in reinforcement-learning self-training on a fixed distribution that KL- and EWC-style penalties do not prevent; our STaR collapse (§E.1) agrees that a penalty is the wrong remedy, and adds that the quantity the loop monitors does not move while it happens. Our model-channel audit finds that at LoRA scale on MedMCQA the single-shot effect is within seed noise and that a plain L2 penalty matches EWC, consistent with the observation that LoRA with small learning rates is itself a strong implicit regularizer. Our non-parametric results suggest that the relevant failure there is not interference along a loss surface but an undetected discrete loss, which is why acceptance and diagnosis rather than penalties remove it; whether the same holds for weights is the open question we leave.

Feedback for LLM optimizers, and repair. That an LLM optimizer needs more than a scalar is established: Nie et al. (2024) separate directional from non-directional feedback and show that the former makes LLM-based optimisation more stable and efficient, and reflective prompt optimizers such as GEPA (Agrawal et al., 2025) are built on the premise that textual feedback is more informative than a reward. Process supervision, which scores intermediate steps rather than outcomes, likewise outperforms outcome supervision for multi-step reasoning (Lightman et al., 2024); our count and anon arms are process-level signals of this kind. These results concern how fast an optimizer improves. Our feedback ladder (§C.3) asks a different question, what feedback must contain for state that a rewrite has destroyed to come back, and finds an asymmetry the speed question does not expose: a denser signal without any direction already prevents the loss, while identifying the failing rule without its content does not repair it, because the rewriter's proposals come from its own prior. HarnessFix (Chen et al., 2026a) repairs an agent's harness from failed trajectories by attributing failures to the responsible component, with an external repairer that always sees the traces; our repair test (§5) concerns the case where the loop's own modification decides whether the failure reaches the optimizer at all.

Evaluation validity for agents. τ-bench introduced pass^k to expose episode variance (Yao et al., 2024), and its successor τ^2 (Barres et al., 2025) scores by final state. Zhu et al. (2025) observed that τ-bench passes an agent that returns nothing on its intentionally unsolvable tasks (38% of airline, 6% of retail) and recommend reporting trivial-agent baselines. We compute the same floor for the τ^2 evaluation slices, where it reaches 0.833 and exceeds every agent we measured, and show what it does inside a self-modifying loop, which a static audit does not see: the floor is where a dismantled agent comes to rest with its optimisation signal unchanged, and it makes giving up a score-optimal repair. Per-cell variance on these benchmarks additionally makes single-shot comparisons uninformative. Anytime-valid acceptance tests (Shawn, 2026) address whether a single noisy evaluation should be trusted and report that greedy commit rules accept 30–40% false improvements; we address what the acceptance rule protects against, and find that even a noise-driven revert suffices. The Darwin Gödel Machine's archive (Zhang et al., 2026b) and AFlow's best-of sampling are, under our reading, implicit instances of the same gate. Safety-oriented studies of self-modification (Shao et al., 2026; Zhou et al., 2026; Jiang & Tang, 2026; Lam et al., 2026) document alignment drift under autonomous self-improvement; the absorbing-state dynamics we describe are a capability-side analogue and share the blind-feedback mechanism. Wang et al. (2026) argue, as a position, that safety is a matter of whether a system stays correctable as it modifies itself, and that visible competence can persist while the conditions for future correction erode; §B.1 and §5 are measured instances of that separation on the capability side, in a stock framework.

H PROOFS

Proof of Theorem 1. Write the rewriter as $R_t = g_t(R_{0:t-1}, F_{1:t-1}, \xi_t)$ and the feedback as $F_t = \phi(R_t, B_t, \zeta_t)$ with $B_t = \mathbf{1}[h(R_t) = C]$. Define the counterfactual process $\tilde{R}_0 = R_0$, $\tilde{F}_t = \phi(\tilde{R}_t, 0, \zeta_t)$, $\tilde{R}_t = g_t(\tilde{R}_{0:t-1}, \tilde{F}_{1:t-1}, \xi_t)$. It is a measurable function of (R_0, ξ, ζ), which is independent of C, so the sequence $\tilde{H}_t = h(\tilde{R}_t)$ is independent of C. Let $\tau = \min\{t : h(R_t) = C\}$.

By induction on t, $R_s = \tilde{R}_s$ for all $s \le \min(t, \tau)$: if the processes agree up to $t-1 < \tau$, then $B_s = 0$ for $s \le t-1$, hence $F_s = \tilde{F}_s$ for $s \le t-1$ and $R_t = \tilde{R}_t$. Consequently $\{\tau \le T\} = \{\exists\, t \le T : \tilde{H}_t = C\} = \{C \in \mathcal{H}_T\}$ with $\mathcal{H}_T = \{\tilde{H}_1, \dots, \tilde{H}_T\} \setminus \{\varnothing\}$, a set of at most T elements independent of C. Conditioning on $\mathcal{H}_T$ gives $\Pr[\tau \le T] = \mathbb{E}[\nu(\mathcal{H}_T)] \le \max_{|S| \le T} \nu(S)$, and the union bound gives $\Pr[\tau \le T] \le \sum_{t \le T} \Pr[\tilde{H}_t = C] = \sum_{t \le T} \sum_c \pi_t(c)\nu(c)$, where π_t is the law of $\tilde{H}_t$, which is the law of the convention stated in round t on the event that no earlier round stated C.

Proof of Proposition 2. Let $Y = h(R_1)$ with $R_1 = g_1(R_0, F, \xi_1)$ and (R_0, ξ_1) independent of C. Treating Y as an estimate of C with error probability $P_e \le \delta$, Fano's inequality gives $H(C \mid Y) \le 1 + \delta \log |\mathcal{C}|$, so $I(C;Y) \ge H(C) - 1 - \delta \log |\mathcal{C}|$. Since Y is a function of (F, R_0, ξ_1), data processing and the chain rule give $I(C;Y) \le I(C; F, R_0, \xi_1) = I(C; R_0, \xi_1) + I(C; F \mid R_0, \xi_1) = I(C; F \mid R_0)$, because (R_0, ξ_1) is independent of C and ξ_1 is independent of (C, F, R_0).

Scope. Assumption 1 is a modelling assumption. It fails for conventions that the executor follows unprompted, where compliance leaks information about C even when C is not stated, and for feedback whose wording describes the convention, such as self-describing short names; both are the cases in which we observe recovery without the full statement.

I Measurement audit: single-shot erosion evidence under noise floors

This appendix collects the measurement groundwork behind the main text: the retention framing that motivated the project, the noise floors, the audit protocol, and what the protocol does to the erosion and mitigation effects that a single evaluation per cell shows.

Mutable state and retention. An agent at stage t has a mutable state R_t: a workflow (prompt program plus operator graph), a bounded skill bank together with the memories it writes, or adapter weights. It evolves through task distributions $D_1, \dots, D_T$, adapting $R_{t-1} \to R_t$ on D_t alone. The *retention matrix* $P_{i,t}$ is accuracy on the held-out split of D_i using the state after stage $t \ge i$; erosion on D_i is $P_{i,i} - P_{i,t}$. Net accuracy hides churn (ten tasks forgotten and ten gained leave it unchanged), so we also record per-task *flips*: forgotten (pass then fail) and gained (fail then pass).

Noise floors. A retention difference is interpretable only above the noise of the design that produced it. We use three floors. The *repeat floor* is the same frozen state evaluated k times; it is needed whenever the executor, a user simulator or a judge samples. The *rebuild floor* is the same state re-instantiated twice before evaluation (a memory rebuilt from the same skill bank); it measures the pipeline that turns state into behaviour. The *seed floor* is the same pipeline re-trained from different seeds; it is needed when the state is itself the outcome of a stochastic optimisation (weights). From a measured per-cell standard deviation σ, the minimum detectable effect of a two-cell comparison with k repeats each is $\mathrm{MDE} \approx (z_{1-\alpha/2} + z_{1-\beta})\, \sigma \sqrt{2/k}$; we report the MDE of the conventional single-evaluation design next to every comparison.

The audit protocol. For any claim that a mutable state erodes, or that a method preserves it, we apply five checks in order (Appendix J gives them as a checklist): **A1** dependence (evaluate with the state removed); **A2** failure separation (count API errors, truncations and parse failures apart from wrong answers, and report the correlation between accuracy and truncation rate across cells); **A3** flip decomposition with calibration (re-answer each forgotten task k times under both states; flips that reverse are noise); **A4** rebuild control (the unchanged early state against itself is the null for every transition); **A5** power (the MDE of the design that produced the claim).

The conventional protocol and the remedy under test. Self-evolution results are typically reported from one evaluation per cell on a dozen to a few dozen tasks, with sampled decoding and the framework's default answer extraction, and with the deployed artifact chosen by best validation round; we call this the *single-shot* protocol and use it as the baseline measurement in this appendix. The remedy continual learning suggests is a retention regularizer: at stage t solve $\min_R \mathcal{L}_t(R) + \lambda\, \Omega(R, R_{t-1})$ instead of $\min_R \mathcal{L}_t(R)$, where Ω penalises departure from the structure that supported

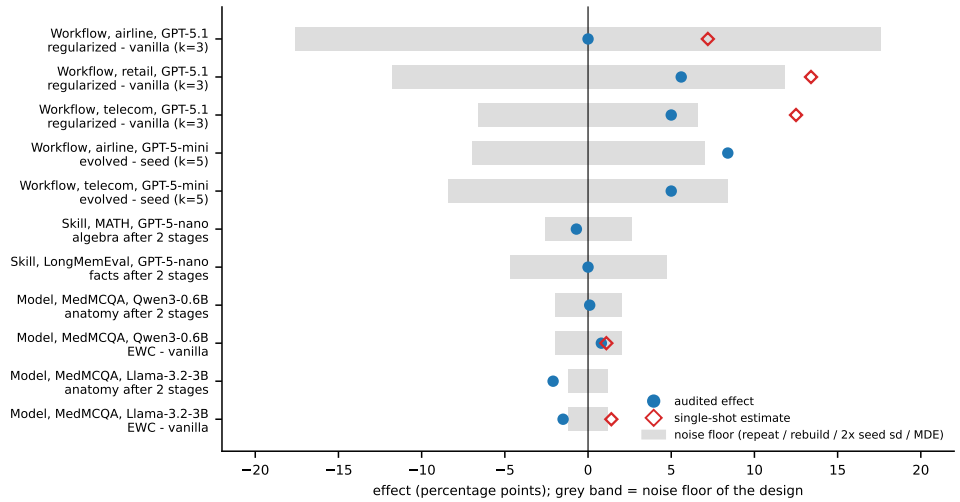

Figure 4: Audited effects against their noise floors. Each row is one erosion or mitigation comparison; the grey band is the floor of the design (repeat sd, rebuild churn, twice the seed sd, or the MDE at $k{=}1$), the dot is our measurement, and the hollow diamond is the single-shot estimate of the same cell where one exists. Only the Llama cells leave their band, in the direction of erosion under EWC as well as vanilla, on a model that learns nothing from the new stages (§E.1).

Table 18: Single-shot effects versus what survives the protocol. "Single-shot" is the regularized-minus-vanilla retention gap from one evaluation per cell; "audited" is the same comparison with the noise floor of the design; "floor" is what the effect must exceed.

Channel / testbed	Single-shot	Audited	Floor	Artifact class
Workflow: τ^2 simple, GPT-5.1	$+7$ to $+13$ pp	0.0; $+5.6 \pm 11.8$; $+5.0 \pm 6.6$ pp ($k{=}3$)	MDE 18.7 pp at $k{=}1$	episode variance; infra scored 0; alias drift
Workflow: τ^2 simple, GPT-5-mini	—	evolved $\geq$ seed in 3/3 domains ($k{=}5$)	repeat sd 0.03–0.10	best-of returns the seed (5/6)
Skill: MATH, GPT-5-nano	$+1.6$ pp	net -1.2 / -0.7 pp; 83% of flips are noise	rebuild 26 vs 35 flips/1000	truncation; zero dependence
Skill: LongMemEval, GPT-5-nano	—	$42.9 \to 38.1 \to 47.6 \to 42.9$	rebuild churn 24%, ± 4.7 pp	rebuild variance
Skill (memory retrieval): MATH, nano / 4o-mini	$2.3 \to 0.7$ pp	dependence gap ≤ 0	—	zero dependence
Model: MedMCQA, Qwen3-0.6B	$+1.1$ pp	anatomy $+0.1 \pm 1.5$ pp; EWC $+0.8$ pp vs. vanilla; L2 learns nothing	$2\times$ seed sd $= 2.0$ pp	sampled evaluation; answer marginal (§E.1)
Model: MedMCQA, Llama-3.2-3B	$+1.4$ pp	anatomy -2.1 ± 2.0 (vanilla), -3.6 ± 1.5 pp (EWC); nothing learned (= freeze)	$2\times$ seed sd $= 1.2$ pp	per-batch truncation flag; answer marginal

earlier stages. We implemented one instance per channel and refer to them collectively as *regularized* evolution: for workflows, the optimizer is shown the previous stage's workflow and asked to preserve its successful execution behaviour; for skills, merging is preferred to deletion when the bank is full, and a memory the skills wrote is removed only with evidence that it is superseded; for weights, EWC (Kirkpatrick et al., 2017), $\Omega = \frac{1}{2}\sum_i F_i(\theta_i - \theta_{t-1,i})^2$ with Fisher weights re-estimated after each stage, plus a plain L2 penalty ($F_i \equiv 1$) and replay as controls. Vanilla evolution is $\lambda = 0$.

The failure mode of the main text is not the one the literature on self-evolving agents has looked for. The inherited worry is capability erosion: adapting to new tasks degrades old ones, in every evolution channel, with regularization as the remedy. We first ran all three channels, vanilla and regularized, under the single-shot protocol described above; every channel showed erosion and every regularizer appeared to help. We then applied the audit protocol to the same artifacts, frameworks, splits and models, and added one testbed (LongMemEval) where the first check fails on the original. Table 18 and Figure 4 summarise; Appendix K–M give every matrix.

Workflow. The single-shot protocol evaluates each evolved workflow once on 12–24 tasks. Evaluating the same frozen artifacts five times gives per-cell standard deviations of 0.017–0.105 (up to 0.171 with GPT-5.1) and per-task flip rates of 25–55% between repeats of the *same* artifact (Table 19); the MDE of a single-shot single-domain comparison is 18.7 pp, above every single-shot gap (Table 20). With k=5, evolved workflows are at or above the seed in every domain and the regularized workflow is never above vanilla; the largest single-shot gap (telecom, +12.5 pp) has a repeat mean of $+5.0 \pm 6.6$, and the single-shot regularized value lies three standard deviations above the repeat mean of that cell. The evaluator scored infrastructure failures as wrong answers (A2), and the same artifacts evaluated four months apart with unpinned model aliases differ by 10–30 pp. Finally, in five of six GPT-5.1 configurations the deployment rule (best validation round on 6–16 tasks) returned round 0, so the "evolved" workflow is the seed and the vanilla-versus-mitigation cells are two single evaluations of the same program (Table 21).

Skill. On MATH the single-shot 1.6 pp gap sits on a testbed with zero dependence: with retrieval disabled, GPT-5-nano scores 87.5/72.8/90.2 on algebra/geometry/number theory, matching or exceeding the best evolved bank (87.1/72.8/90.7), and GPT-4o-mini shows the same (Table 22). Before that check, flip decomposition already attributes 77–86% of the 22–35 forgotten problems per transition to sampling, and the rebuild control produces 26 flips against 35 for a real transition (Table 23). The memory-retrieval variant of the skill channel, in which the bank's entries are retrieved as memories, shows a single-shot erosion of 83.9 → 81.6 on the same benchmark and models; a 2.3 pp single-run gap on a testbed whose memory contributes nothing measurable cannot be attributed to memory. On LongMemEval, where no memory scores 0 and the dependence gap is +43 pp, facts accuracy across three stages is 42.9 → 38.1 → 47.6 → 42.9, but rebuilding the *same* bank twice flips 15/63 answers and moves accuracy by 4.7 pp; no transition exceeds this floor (Table 24).

Model. We trained the STaR+LoRA pipeline with 5 seeds × {vanilla, EWC, L2, replay} × 3 stages on Qwen3-0.6B and Llama-3.2-3B (n=2000 per evaluation), and re-evaluated every adapter greedily with a 768-token budget, strict extraction, and non-terminating generations re-generated at 2048 tokens and scored as wrong (Table 25, Appendix M). The single-shot evaluation sampled at temperature 0.7 with a 192-token budget and a fallback extractor that reads the last isolated letter of a truncated output, and its truncation flag was set per batch rather than per item, so the single-shot Llama numbers tracked how many batches contained a non-terminating generation rather than anything about the model. Under the clean evaluation Qwen's anatomy accuracy moves by $+0.1 \pm 1.5$ pp over two transfers (vanilla; $2\sigma_{\text{seed}} = 2.0$ pp), EWC's retention change is 0.8 pp better than vanilla's, inside the floor, and the L2 penalty prevents learning as well as forgetting: its biochemistry and dental accuracy equal the frozen anatomy adapter's (35.6 and 27.9 against 39.2 and 31.9 for vanilla). Llama loses 2.1 ± 2.0 pp under vanilla and 3.6 ± 1.5 pp under EWC, both above $2\sigma_{\text{seed}}$, while learning nothing: its biochemistry and dental accuracy after training equal the frozen adapter's, and the untrained model is above every trained cell. What moves in both models is not knowledge of anatomy but the answer marginal, the subject of §E.1.

What this leaves standing. Four artifact classes account for the single-shot erosion and mitigation: sampling and episode variance, silent failure scoring (truncation on MATH and MedMCQA, infrastructure failures on τ^2), rebuild variance, and selection on noise. Two structural facts sit underneath: zero dependence on MATH, and a do-nothing floor on τ^2 airline above every workflow we evolved. No mitigation effect exceeds its floor; the one erosion that does, 2–4 pp on Llama, appears under EWC as well as vanilla, on a model that learns nothing from the new stages, and is carried by the answer marginal rather than by the questions (§E.1). The two silent-failure classes are the hand-written form of the silencer that the optimizer of §B.1 wrote for itself; the retention numbers they produced were not measuring the erosion they were meant to measure, and could not have seen the loss of repairability that was actually happening.

J THE AUDIT PROTOCOL AS A CHECKLIST

For a claim that state R_t erodes (or that a method preserves) capability on D_i:

1. **A1 Dependence.** Report $P_{i,\cdot}$ with the mutable state removed. Proceed only if the dependence gap is positive by more than the repeat floor.

2. **A2 Failure separation.** Report, per cell, the count of API errors, empty outputs, truncated generations and parse failures. Never fold them into the accuracy denominator silently. Report $\mathrm{corr}(\text{accuracy}, \text{truncation rate})$ across cells; if $|r| > 0.5$, the numbers are not about capability. Excluding failed items from the denominator is not a fix: it creates the opposite bias (Appendix M).
3. **A3 Flip decomposition.** For each forgotten task, re-answer $k \geq 3$ times under both states; classify as noise if any re-answer reverses the flip. Calibrate with the flip rate of tasks solved in all k under the early state.
4. **A4 Rebuild control.** Re-instantiate the early state once more (rebuild memory, re-sample the evaluation, re-train with the same seed where applicable) and measure flips against itself. Report every transition's forgotten/gained counts next to this null.
5. **A5 Power.** From the per-cell standard deviation σ measured in A3/A4, report the MDE of the design used for the claim: $\mathrm{MDE} \approx (z_{1-\alpha/2} + z_{1-\beta})\,\sigma\sqrt{2/k}$ for a two-cell comparison with k repeats each.

Floors: dependence gap (A1) and do-nothing floor (score of an agent that returns immediately; compute analytically from the task criteria and confirm empirically).

K Workflow channel: additional details

K.1 Repeat evaluation

Table 19: Repeat evaluation of frozen τ^2 workflows on the simple split (mean $\pm$ sd over k runs; zero infrastructure failures in all cells). The last column is the single-shot value of the same configuration.

Executor	Domain (n)	Seed	Vanilla	Regularized	Single-shot van. / reg.
GPT-5-mini, k=5	airline (12)	0.733 ± 0.062	0.817 ± 0.033	0.750 ± 0.105	—
	retail (24)	0.833 ± 0.046	0.867 ± 0.017	0.858 ± 0.020	—
	telecom (20)	0.450 ± 0.045	0.500 ± 0.071	0.460 ± 0.049	—
GPT-5.1, k=3	airline (12)	0.778 ± 0.039	0.694 ± 0.171	0.694 ± 0.039	57.1 / 64.3
	retail (24)	0.458 ± 0.102	0.486 ± 0.052	0.542 ± 0.118	43.3 / 56.7
	telecom (20)	0.117 ± 0.024	0.217 ± 0.024	0.267 ± 0.062	25.0 / 37.5

Table 20: Minimum detectable effect (80% power, α=0.05) for a two-cell comparison on τ^2 simple splits at the measured per-cell $\sigma = 0.047$.

k repeats per cell	1 (single-shot)	3	5	10	20
single domain	18.7 pp	10.8 pp	8.4 pp	5.9 pp	4.2 pp
three domains pooled	10.8 pp	6.2 pp	4.8 pp	3.4 pp	2.4 pp

Table 21: Validation score by round for the GPT-5.1 configurations and the round that the deployment rule selects. Round 0 is the unmodified seed.

Domain	vanilla, rounds 0–3	regularized, rounds 0–3	selected (vanilla / regularized)
airline	.33 .33 .00 .33	.50 .33 .50 .33	round 0 / round 0
retail	.50 .38 .44 .50	.44 .44 .38 .44	round 0 / round 0
telecom	.64 .43 .00 .50	.57 .64 .64 .57	round 0 / round 1

Artifacts are the seed workflow (round 0), the best vanilla round and the best regularized round for each domain, taken from the round directories of the single-shot runs. Evaluation uses the simple split, slice $[0.2, 1.0]$ as in the single-shot runs, GPT-5.1 as user simulator, a maximum of 40 interaction steps, and an evaluator patched to record per-task scores and to count infrastructure failures separately (all cells in Table 19 have zero). Per-task pass/fail flip rates between repeats of the same artifact are 25–55%. The same artifacts evaluated in April 2026 (single run, GPT-5-mini) scored 0.500–0.750; in September 2026 they score 0.817–0.867 with sd 0.02–0.05, a shift far outside episode noise that we attribute to unpinned model aliases.

K.2 POWER

Median per-cell sd over the 72 repeat evaluations is 0.047 (range 0.017–0.171); Table 20 uses this value with $z_{0.975} + z_{0.8} = 2.80$. For the GPT-5.1 cells alone the median is 0.062, and the corresponding single-shot MDE is 24.6 pp.

K.3 THE EVOLVED ARTIFACT IS USUALLY THE SEED

The deployment rule loads the round with the highest validation score in `results.json`, ties to the earliest round. Validation sets have 6 (airline), 16 (retail) and 14 (telecom) tasks, so validation scores move in steps of $1/6$ to $1/16$. Prompt and graph similarity are computed with a token-level sequence matcher on `prompt.py` and `graph.py`; round 0 is a copy of the seed by construction. For the GPT-5-mini artifacts used in the repeat audit, the deployed rounds and their similarity to the seed are: airline vanilla round 1 (0.20 / 0.72, a large rewrite), airline regularized round 0 (1.00 / 1.00), retail vanilla round 1 (1.00 / 0.97), retail regularized round 2 (1.00 / 0.88), telecom vanilla round 2 (0.88 / 0.95), telecom regularized round 1 (0.32 / 0.80).

K.4 DO-NOTHING FLOOR DERIVATION

τ^2 airline and retail use `reward_basis` = [DB, COMMUNICATE] (retail additionally NL_ASSERTION); telecom uses ENV_ASSERTION. For airline the `communicate_info` field is empty on all simple-split evaluation tasks, so the reward reduces to whether the final database equals the target. A task whose expected actions contain no write (only `get_*`, `search_*`, `calculate`) has target = initial state and is passed by an agent that returns immediately. The twelve airline/simple evaluation tasks (ids 5, 6, 9, 16, 26, 27, 28, 36, 40, 46, 48, 49) contain writes only in 16 (`update_reservation_flights`) and 40 (`update_reservation_passengers`); floor = $10/12$. Retail evaluation tasks with no write: 50, 65 (simple) and 25, 57 (complex). Telecom's environment assertions require the user to perform device actions under the agent's guidance, so the static count is not informative; the measured floor with a workflow that returns `done()` on every step is $0/20$ (two runs). Per task, the dead workflow passes every airline/simple task except 16 and 40 in all five runs; the template passes 5, 16, 26, 28, 36, 46 in all runs and never passes 48.

K.5 CROSS-DOMAIN CHAIN (W1): CONFIGURATION

Training uses each domain's complex split with the first 20% as the optimizer's training set and the remainder held out; four optimisation rounds per stage; the stage-$s{+}1$ optimizer is initialised from the *last* round of stage s (not the best) via a `--graph-path` chaining option added to the training script; GPT-5.1 optimizer and user simulator; GPT-5-mini executor; $k{=}5$ evaluations per simple split and $k{=}3$ on the trained complex split after every stage. Validation scores by round were $(0.33, 0.17, 0.17, 0.17, 0.17)$ for airline, $(0.50, 0.50, 0.50, 0.50, 0.50)$ for retail and $(0.21, 0.21, 0.50, 0.36, 0.29)$ for telecom. Total cost including evaluations was approximately $120.

L SKILL AND MEMORY CHANNELS: ADDITIONAL DETAILS

Table 22: MATH skill channel (GPT-5-nano, bank capacity 30). Rows: evaluation domain; columns: state after each stage. The no-memory baseline matches or exceeds every evolved state. GPT-4o-mini, no memory / own memory / reading GPT-5-nano's memory: algebra $84.0/84.5/82.9$; geometry $53.0/49.9/49.5$; number theory $74.4/75.0/74.3$.

Eval (n)	no memory	@algebra bank	@geometry bank	@number-theory bank
algebra (1000)	87.5	87.0	85.8	86.3
geometry (479)	72.8	—	69.7	72.0
number theory (540)	90.2	—	—	90.7

MATH pipeline. Bank capacity 30; Contriever retrieval with top-5 skills and top-5 memories during training, top-10 memories at evaluation; designer triggered every 50 chunks, at most 3 new

Table 23: Flip decomposition on MATH (algebra evaluation, n=1000): forgotten problems classified by three re-answers under each state. The rebuild control is the same bank with memory rebuilt twice.

Transition	forgotten	noise	constr. loss	interference	retr. masking	gained
algebra → geometry	35	29 (83%)	4	2	0	23
algebra → number theory	28	24 (86%)	3	1	0	21
geometry → number theory (geometry eval)	22	17 (77%)	3	2	0	—
rebuild control (algebra, bank unchanged)	26	—	2	—	—	23

Table 24: LongMemEval skill channel (GPT-5-nano; accuracy in %). Rebuild: the updates bank with memory rebuilt a second time. Facts flips, forgotten/gained: seed → facts $9/6$; facts → temporal $8/14$; temporal → updates $11/8$; updates → rebuild $6/9$.

Eval (n)	no memory	seed (3 skills)	@facts (21)	@temporal (30)	@updates (30)	rebuild
facts (63)	0.0	42.9	38.1	47.6	42.9	47.6
temporal (54)	0.0	—	—	24.1	27.8	—
updates (32)	0.0	—	—	—	46.9	—
multi-session (54)	—	—	29.6	—	22.2	—

skills per cycle, 2 reflection iterations; 500 training problems per stage; evaluation on up to 1000 held-out problems per domain. The no-memory baseline sets the retrieval budget to zero. The flip decomposition re-answers each flipped problem three times under the early context, the late context, and the late context with the masked or lost memory items restored, and classifies the flip by which restoration recovers the answer; the noise calibration is the single-answer flip rate of problems solved in all three re-answers (0.7–4.0% depending on domain). The capacity-8 run reproduces the flat matrix: algebra 86.8 at its own bank, 87.1 after geometry, 86.4 after number theory; the final bank held 4 algebra, 2 geometry and 2 number-theory skills because usage counts favour incumbents.

LongMemEval pipeline. Stages facts (93 training instances), temporal (79), updates (46); evaluation sets of 63, 54, 32 questions plus a 54-question multi-session set; GPT-5-nano executor at low reasoning effort with a 16k completion budget; bank capacity 30 (Table 24) or 10 (Table 17). The rebuild control re-runs memory construction from the same final skill bank on the same conversations and re-evaluates. Transitions: seed → facts forgot 9 gained 6; facts → temporal $8/14$ ($p \approx 0.17$ against the rebuild null); temporal → updates $11/8$.

Eviction policies (S4). The bank enforces its capacity on load and at every insertion; the policy chooses the victim by lowest usage count (LFU), earliest insertion (FIFO), uniformly at random, or least-recently-used (LRU, using a use-clock updated at retrieval). Under LFU a skill inserted in the current designer window has usage 0 and is the next victim, so in the temporal stage 16 insertions changed only 2 entries. The updates-stage banks of the LFU and FIFO runs were rolled back to their last window checkpoint by a resume after an API outage and are identical to the temporal-stage banks; their updates@updates cells therefore evaluate the temporal bank on the updates split and score 0.500 like the others.

M MODEL CHANNEL: ADDITIONAL DETAILS

STaR self-training with LoRA on MedMCQA, anatomy → biochemistry → dental, 5 seeds, methods vanilla / EWC / L2 / replay (Qwen only), $n = 2000$ per evaluation; each stage generates rationales on the stage's training split, keeps those with the correct final answer, and fine-tunes the adapter on them, with EWC's Fisher weights re-estimated after each stage. Single-shot evaluation: temperature 0.7, 192 new tokens, three-level answer extraction whose last level takes the last isolated letter in the text. Truncation flags under this setting ranged from 27% to 72% and moved with fine-tuning; the flag was set from the padded length of the batch, so it marked every item in a batch that contained one non-terminating generation, and it is this per-batch flag that correlated with accuracy at $r = -0.91$ to -0.96 across the Llama cells of the single-shot evaluation. Per item, non-termination is rare: $\leq 0.9\%$ in every Llama cell and 0.2–10.1% in Qwen's.

Greedy re-evaluation. All 100 adapters were re-evaluated greedily with a 768-token budget and strict extraction (only a `Final Answer:` line or `\boxed{}`), plus no-fine-tune and freeze (anatomy adapter on later domains) baselines, 231 evaluations of $n = 2000$. An item whose generation does not stop within the budget is re-generated with a 2048-token budget, which under greedy decoding equals having used the larger budget throughout; an item that still does not stop, or has no extractable answer, scores 0 with the full n as denominator. (Excluding unparsed items from the denominator instead biases the accuracy upward when the unparsed items are the hardest, so it is not a valid fix for A2.) Table 25 gives the matrices; Qwen's no-fine-tune row is 0 under strict extraction because the base model does not write the final-answer line ($1973/2000$ answers are read by the fallback; 24.4% under the lenient extractor). Per-item non-termination on anatomy rises along the chain for Qwen under vanilla ($0.4 \to 3.2 \to 6.1\%$), EWC ($0.4 \to 2.1 \to 4.3\%$) and replay, and not under L2 (0.4% throughout) or for Llama ($\leq 0.3\%$); mean generation length doubles ($142 \to 276$ tokens). Flip decomposition on anatomy between the first and last stage: Llama vanilla forgets 256 and gains 213 items per seed, and of the 1005 items all five seeds answer correctly after the first stage, 110 are lost per seed under vanilla, 119 under EWC and 16 under L2; two seeds at the same stage disagree on 17–23% of items. For Qwen the corresponding counts are 319 forgotten against 321 gained (vanilla), with seed-pair disagreement of 23–25%.

Table 25: Model channel under the clean evaluation (greedy, 768 tokens, strict extraction, non-terminating items re-generated at 2048 tokens and scored as wrong if still unterminated; mean ± sd over 5 seeds, %). "freeze" is the anatomy adapter evaluated on the later domains; "nf" is no fine-tuning. The last row is the change in anatomy accuracy from the first to the last stage.

Qwen3-0.6B	nf	freeze	vanilla	EWC	L2	replay
anatomy @ anatomy	0.0	30.8 ± 1.0	30.8 ± 1.0	30.6 ± 1.4	30.4 ± 1.2	—
anatomy @ biochem.			31.3 ± 0.7	32.7 ± 0.7	30.5 ± 0.8	31.9 ± 1.2
anatomy @ dental			31.0 ± 0.8	31.6 ± 1.1	30.5 ± 1.0	32.1 ± 0.6
biochem. @ biochem.	0.1	35.3 ± 2.0	39.2 ± 1.8	40.0 ± 1.7	35.6 ± 2.2	38.4 ± 1.8
biochem. @ dental			39.3 ± 0.6	40.4 ± 0.5	35.4 ± 1.8	40.6 ± 0.2
dental @ dental	0.1	28.1 ± 1.7	32.0 ± 1.1	32.1 ± 0.8	27.9 ± 1.1	32.0 ± 0.9
Δ anatomy, first → last			$+0.1 \pm 1.5$	$+0.9 \pm 0.8$	$+0.1 \pm 0.5$	—
Llama-3.2-3B	**nf**	**freeze**	**vanilla**	**EWC**	**L2**	
anatomy @ anatomy	70.3	68.5 ± 0.6	68.5 ± 0.6	69.5 ± 0.4	69.1 ± 0.5	
anatomy @ biochem.			67.7 ± 0.7	68.1 ± 0.8	69.1 ± 0.6	
anatomy @ dental			66.4 ± 1.6	65.9 ± 1.3	69.1 ± 0.5	
biochem. @ biochem.	79.3	78.6 ± 0.6	78.8 ± 0.9	78.8 ± 0.6	78.5 ± 0.8	
biochem. @ dental			76.3 ± 1.6	76.1 ± 1.0	78.4 ± 0.4	
dental @ dental	53.8	52.3 ± 0.7	51.5 ± 1.5	51.6 ± 0.6	52.5 ± 0.8	
Δ anatomy, first → last			-2.1 ± 2.0	-3.6 ± 1.5	-0.0 ± 0.6	

Dose escalation. The four settings of Table 14 change, cumulatively, the number of STaR iterations per stage ($1 \to 4$, keeping the earlier iterations' training rows within a stage), the learning rate ($2 \times 10^{-4} \to 8 \times 10^{-4}$), the LoRA rank ($16 \to 64$, α scaled with it), and finally drop the carry-over of earlier iterations' rows; three seeds each, Qwen3-0.6B, same chain and evaluation. Self-generated rationales use 192 tokens at temperature 0.7, and from the second iteration on 90–100% of them reach that budget; after the first stage 14–25% of evaluation generations do not terminate within 768 tokens, falling to $\leq 2\%$ by the third stage. The pre-registered erosion criterion (anatomy drop $> 2\sigma_{\text{seed}}$ with flips concentrated on items all seeds had answered correctly) is met by no setting.

Compute. The model-channel experiments used about 1,000 A40 GPU-hours in total: 719 for the re-evaluation of the 100 adapters (434) and the 2048-token continuations (284), and 279 for the intensity sweep (106), the four arms of Table 15 (127), their development probes (11), the Llama-3.2-3B chain (29) and smoke tests (6); about 60 of these hours were spent on attempts that failed or timed out and were re-run. The open-weight probe runs of §E.2 used a further 25 GPU-hours (15 for the full-rule batch, 10 for the calibrated batch), not counting two attempts that produced no data.

N THE INSTRUCTION-FOLLOWING PROBE: ADDITIONAL DETAILS

Task generation. The base task: "You generate a daily orders report from a JSON list of orders. Each order has id, customer, date (ISO), amount (USD) and status (paid / refund / pending). Every

listed order line must include the order id, customer, date, amount and status. Output only the report text, no explanations." followed by "Follow ALL of these conventions:" and the fifteen rules of Table 9. Tasks are generated from 15 names (including *liam o'brien* and *fatima al-sayed*), random dates in March 2026, amounts, and statuses with a 40% pending/refund share; each rule has a deterministic checker over the output text and the input orders (e.g. `eur` recomputes the converted amounts, `sorted_date` parses the dates in output order).

Rewrite prompts. *Feedback* (also used by feedback+gate; the ladder arms of §C.3 append one block to each INCORRECT run and change nothing else: *count* appends "CONVENTIONS SATISFIED — n/15"; *anon* appends "FAILED CONVENTIONS (ids) — #7, #12"; *name* appends "FAILED CONVENTIONS — eur, total_first"; *spec* appends "FAILED CONVENTIONS —" followed by the full statement of each failed rule, one per line, verbatim from Table 9):

```
You are improving the instruction prompt of an automated
report-writing agent.  Below is the CURRENT PROMPT, followed
by several recent runs (input, the agent's output, and
whether the output was judged correct).  Rewrite the prompt
so that future outputs are judged correct more often.
Return ONLY the complete new prompt inside a ```prompt
fenced block.
```

Consolidate:

```
You are maintaining the instruction prompt of an automated
report-writing agent.  Rewrite the CURRENT PROMPT to be
clearer, better organized and more concise while preserving
its intent.  Return ONLY the complete new prompt inside a
```prompt fenced block.
```

The gate arm compares the round's all-rule pass rate on the 24 evaluation tasks with the best so far and restores the best prompt if it is lower; because this is the same set on which the arm's outcome is reported, its final pass rate is subject to selection on evaluation noise (§7).

Validity. Execution uses reasoning effort "low" with a 6000-token completion budget; the presence judge uses 8000 tokens and the rewriter 16000. Any empty output is flagged and excluded rather than scored, and a rewrite that returns nothing keeps the previous prompt with a flag. The archived invalid run used a 1200-token budget; GPT-5-nano exhausted it while reasoning, $22/24$ evaluation outputs were empty, and within eight rounds the feedback arm had rewritten the prompt into a defensive stdin-parsing specification that treated every input as an empty array and printed "Total: 0.00". The presence judge's noise floor is 2.8% rule flips per round under GPT-5-nano and 0% under GPT-5-mini (control arm).

Open-weight rewriters. Qwen3-8B, Llama-3.1-8B-Instruct, Qwen3-1.7B and Llama-3.2-3B-Instruct are served with vLLM (bf16, temperature 0.7, Qwen3 thinking disabled) and act as executor, rewriter and presence judge, with the rewriter budget of 16,000 tokens and a gate that decides on a separate six-task validation set. The full-rule batch (three seeds, all fifteen conventions, five arms) is the one described at the end of §E.2; its seed all-rule pass rates are 0.00–0.08. For the calibrated batch each model first runs the control arm for five rounds with all fifteen conventions (three seeds); the conventions with mean compliance ≥ 0.90 are kept, and the lowest would be dropped until the seed all-rule pass rate on the kept set reaches 0.5 (no drop was needed). Kept sets: Qwen3-8B header, two_decimals, total_exists, date_dmy, prefix, rfd, total_first, separator; Llama-3.1-8B the same without date_dmy (rfd, at 0.86, was kept from the list fixed before the calibration run); Qwen3-1.7B header, two_decimals, total_exists, prefix; Llama-3.2-3B header, two_decimals, total_exists, prefix, total_first, separator. The seed prompt, the checkers, the presence judge and the feedback blocks use the kept set only (the count block reads "n/k" for k kept conventions), and the loss criterion is applied to the kept conventions. Validity: no empty outputs in 65,520 evaluation calls; 0–0.7% of completions hit the length limit; the presence judge returned no parse in 9 of 735 rounds for Llama-3.1-8B and 5 for Llama-3.2-3B, and never for Qwen; control-arm presence flips are 0 for three models and 1.8% for Llama-3.2-3B. Under accept-all rewriting the Llama-3.2-3B prompt reaches 55,000–72,000 characters in three runs (the rewrite budget), while Llama-3.1-8B's prompts stay under 13,100 and
```

Qwen's under 3,300; consolidate shrinks the Llama prompts to 142–568 characters. Of the 25 Llama runs with a permanent loss, 22 end with an output that is not a report (a Python program in 18, a JavaScript program in one, a markdown document with headings and tables in two, and in one a code fence around an otherwise intact report); the three with an intact report lose the `Total:` placement, the separator (deleted from the prompt), and the two-decimal rule through an ISO timestamp with milliseconds in the date field.

Presence versus behaviour. In the strong-rewriter runs presence stays at $15/15$ while edge-case compliance drifts (Table 12); in the 50-round collapse the presence judge stopped recognising the EUR rule three rounds before behaviour diverged. Presence is therefore reported only as a diagnostic.

O ACCEPT-ALL VERSUS GATED AFLOW (W2): ADDITIONAL DETAILS

Configuration: EvoAgentX AFlow optimizer, airline complex split for training (validation 6 tasks, one validation round per optimisation round), executor GPT-5-mini, user simulator GPT-5.1, optimizer GPT-5-nano with a 16k completion budget, 20 rounds, convergence check disabled. The accept-all arm overrides parent selection to return the latest scored round; everything else is stock. The optimizer's output is validated (parseable graph, valid prompt placeholders) and falls back to the parent on failure, which happened at rounds 19 and 20 of the accept-all arm.

Gated arm. Validation scores by round: .33, .50, .50, .33, .00, .50, .33, .17, .17, .00, .00, .50, .00, .33, .33, .33, .50, .17, .17, .00, .00; rounds 4 and 10 cost \$0 (dead children). Parent tree: round $0 \to \{1, 11, 16, 18\}$, $1 \to \{2, 3, 4, 6, 8, 12, 19\}$, $2 \to \{5, 7, 9, 10, 13, 14, 15, 17\}$. AFlow deploys the highest-scoring round (ties to the earliest), round 1, whose only change from the template is a `done()` fallback when action extraction returns nothing. Round 10 of the gated arm has prompt similarity 0.015 and graph similarity 0.147 to the template (a "simplify the workflow" rewrite); round 20 replaced {`observation`} and the other placeholders with the literal strings `OBSERVATION` etc.

Cost. Training both arms cost under \$20 in executor calls (the accept-all arm's validation runs cost \$0 from round 10); the 46 evaluations cost approximately \$80.