



RUNESMITH

Beyond the Model

Runesmith, an Open Runtime That Improves Software
and Itself, and the Instrument–Substrate Hypothesis

Lars O. Horpestad

AI ThinkLab

6 October 2026

Preprint · not peer reviewed

Supported by  Innovation
Norway

Version 1.0, 6 October 2026, released with Runesmith 1.0. Preprint; not peer reviewed. DOI: 10.5281/zenodo.23196763 (paper, technical report and evidence).

© 2026 AI ThinkLab. This paper is licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0, <https://creativecommons.org/licenses/by/4.0/>). The Runesmith software it describes is licensed under the Apache License 2.0.

Companion technical report. This paper distills *Beyond the Model: Technical Report* (TR), released with it as a separate document, which gives the full method, the source of every number and the complete negative record, drawn from the project’s dated laboratory record (“the record”). Pointers of the form TR §5.2.5 refer to it, and every number below is taken from it. The report stands as written before LOC1 (§4) was run; where the two differ, this paper is current.

Abstract

Agent systems rent competence from a model one call at a time. We present Runesmith, an open-source runtime (Apache-2.0) in which models propose, a small deterministic kernel admits, and the runtime changes its own code only on evidence. In one iteration of its Kaizen loop, Runesmith chose a weakness from its own telemetry and a model rewrote its repair organ, the code it repairs software with. With the same cheap repair model and budget, the new organ repaired more than the organ it replaced, 57 against 35 of 162 sessions on 54 fresh tasks, in about half the time per repair (SR7; exact $p = 0.000845$, below its preregistered Bonferroni bar of 0.0036 and the bar for all 23 sealed comparisons run). In a further sealed test on three large public repositories with one free model (LOC1), the self-improved organ repaired more regressions than the organ Runesmith shipped with (42 against 17 of 168 sessions, $p = 0.00278$, significant at its Bonferroni level for its three tests); a hand-written trace-aware localization rule repaired more still (65 of 168, mirrored $p = 0.00077$), so on this family the learned gain is one that a careful designer can also write by hand. In a post-hoc count, 123 of the 124 repairs came with the faulty file in the model’s view: what Runesmith kept is a localization policy. LOC1’s protocol and result wording were sealed before any outcome existed and publicly timestamped hours before the analysis. Runesmith changed its own problem-solving code and the change helped under seal, while the improvement process itself stayed unchanged; we call this, and only this, a nudge towards recursive self-improvement. With no model available in 104 sealed scenarios, it made no unauthorized change (15 effects lacked their own ledger event, added in 1.0.0). Six of seven earlier sealed tests did not show their effect; all are reported, with code, data and recomputations.

Key findings in plain words.

- **Runesmith improved its own code, and two sealed tests confirmed it.** Its improvement loop found a weakness in its own repair code and had a model rewrite it. With the same small open-weights AI model and the same budget, the new version fixed 57 of 162 attempts where its previous version fixed 35, in about half the time per fix (SR7), and then 42 of 168 where the version Runesmith shipped with fixed 17, on fresh code from three large public open-source projects (LOC1).
- **It found the right problem.** The weakness the loop picked was that the AI often never looked at the file holding the bug. The new version put that file in front of the AI three times as often as the shipped one (86 against 27 of 168), and 123 of the 124 fixes in the test came from attempts where the AI could see it.
- **The rules were locked before the data existed.** Each test’s design was sealed before it ran. For LOC1, the wording of this paper’s result was sealed too, and their fingerprints were anchored in the Bitcoin blockchain hours before the results were analysed.
- **A careful hand-written rule did better overall** (65 of 168), though not everywhere: on one of the three projects Runesmith’s version matched it (11 against 10 of 39). The rule was written after Runesmith’s version existed and was tuned on the earlier study’s tasks; whether the loop can reach it on its own is the next test.
- **Without AI, it changed nothing it was not allowed to.** With no AI model available, Runesmith made no unauthorized change and broke no ledger chain in 104 sealed scenarios.
- **Every sealed result is reported.** Six of seven earlier sealed tests did not show their effect, each reported with its protocol’s verdict. Code, paper, data and scripts that recompute every test are public.

1. Introduction

An agent built on a language model can plan, write code and repair it inside one episode, but when the episode ends the system keeps a transcript, some files and perhaps a summary. How much of the episode's competence survives the removal or replacement of the model that produced it is unclear. The question this program asks is narrower than whether the system is intelligent: *where does the competence of the resulting system reside?* (TR §1.1)

We treat models, humans, rules, solvers and tools as replaceable **cognitive instruments**, and evidence-bearing runtime state as a candidate **capability substrate**. The **instrument-substrate hypothesis** holds that some useful competence can reside in evidence-bearing runtime state rather than exclusively in model weights: a capable instrument may discover a solution, the runtime may keep the useful part as a tested procedure, policy, memory, capability or evaluator, and a weaker instrument, or none, may later reuse it within a validated scope (TR §1.2). The hypothesis is built to fail cleanly. No state carryover would mean no persistent capability; no retention under substitution, no instrument-independent substrate; no integrity under model starvation, a kernel that was model-dependent after all. Its mechanism is *capability transduction*: turning successful episodes into a scoped artifact whose future contribution is positive on held-out tasks. Saving prose, passing a self-generated test or raising a judge score does not count (TR §1.3).

Prior work supplies most of the mechanisms such a system would use: retrieval [1], reasoning and acting with tools [2, 3], verbal self-reflection [4], executable skill libraries [5], scaffold self-modification [6] and evaluator-guided evolution of coding agents [7]. None of it tests the conjunction this program proposes: separately custodied evidence, scoped authority and revocation, complete accounting of assigned opportunities, and promotion only at a preregistered endpoint.

Runesmith is a runtime built to that contract and released as open source under the Apache License 2.0. It plans software as milestones with acceptance checks, builds and improves the software it works on, and rewrites its own self-modifiable code through a loop that changes nothing until evidence says so; its predecessor kernel operated a live production service (§6). It runs on any OpenAI-compatible inference: free hosted tiers, paid providers and local servers.

Contributions. From the record and the sealed studies run after it, this paper reports:

1. **Sealed self-improvement (SR7, §3).** Runesmith's own loop rewrote its repair organ, and the new generation repaired 57 of 162 sessions against 35 of 162 on 54 fresh tasks, over its own previous generation, which had not beaten the shipped organ (exact one-sided $p = 0.000845$).
2. **The test against the shipped organ (LOC1, §4).** On three large public repositories where its ranking decides what the model sees, the self-improved organ repaired 42 of 168 sessions against 17 for the organ Runesmith shipped with (exact one-sided $p = 0.00278$, PASS at its Bonferroni level), and a hand-written trace-aware rule repaired 65 (mirrored $p = 0.00077$). Its protocol and result wording were sealed before any outcome existed and publicly timestamped hours before the analysis.
3. **A replicated second self-improvement and a starvation-integrity estimate (§5).** A free-model-authored change made Runesmith's memory retrieval faster on fresh workloads (15 of 16 blocks, $p = 0.0002594$), on a target the operator chose; and with no model available, the runtime made no unauthorized change in 104 sealed scenarios, with 18 integrity failures (17.3%), 15 of them effects recorded without a ledger event of their own (added in 1.0.0, not re-measured).
4. **Field evidence (§6), observational:** target-hidden corrections on a production service and its operation by the predecessor kernel; a program for which free-tier models were the last author of every applied change; live repair with free models on the build just before release 1.0.0; and a 16-minute acceptance test on one free key.
5. **Fresh-task comparisons after the record (§7)** against the strongest fixed scaffold the record defined, reported whatever they showed.
6. **A complete negative record (§8):** six of the seven sealed tests of the September series did not show their effect, and they shaped the loop that passed.
7. **Runesmith 1.0 and the released evidence,** including SR7's 324 session outcomes and the script that recomputes its exact test.

The novelty is the evidence-to-promotion contract working end to end: a runtime that changes its own code only on evidence, and that has passed two sealed tests of doing so (Figure 1). Each component has precedents (§9).

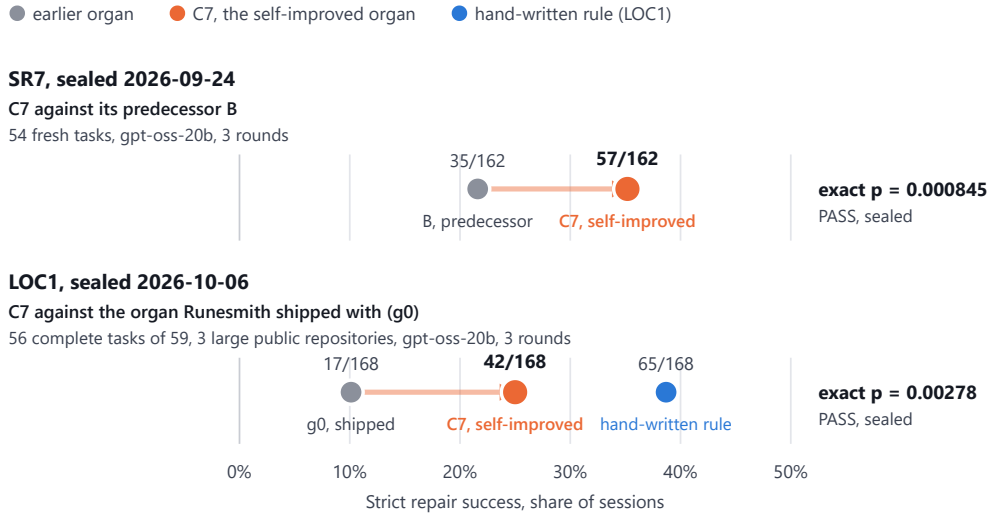


Figure 1. The sealed two-step record. Strict repair success under two sealed, preregistered tests, each with one free repair model (gpt-oss-20b) and one envelope for all its arms. SR7: the self-improved organ C7 against its predecessor B on 54 fresh tasks (§3). LOC1: C7 against the organ Runesmith shipped with, g0, on 56 complete tasks of 59 from three large public repositories, with the hand-written trace-aware rule shown as a reference (§4). Exact one-sided sign-flip tests over tasks; both pass at their declared levels, and in LOC1 the hand-written rule repaired more than C7.

2. How Runesmith works

A model in a precise sense. Runesmith is a fixed policy plus persistent, evidence-bearing state that maps a goal, an environment and any available inference to verified changes. With active generation G_k , the persistent state S_t of one project home, the owner’s goal g , the environment $\mathcal{E}_t$ and the set $\mathcal{I}_t$ of inference instruments available at time t , which may be empty, one step is

$$(\Delta_t, S_{t+1}) = F_{G_k}(g, \mathcal{E}_t, S_t, \mathcal{I}_t),$$

where Δ_t is the set of changes applied to the environment, each of which passed the checks the state requires, and S_{t+1} extends S_t by appended ledger events, maps, memory of attempts and experience records (TR §2). Its “parameters” are readable organ code, contracts and declared policies, not numerical weights; no weights are trained.

Propose, admit. Models and other instruments propose; a small deterministic kernel admits. It owns identity, the ledger, the resource envelope, activation and rollback (TR §2, §3.4). By design (TR §3.1–3.2), a candidate carries no authority status, the kernel admits it only if its schema, authority, budget and preconditions check, and a proposal generator may not alter the release evaluator, hidden test identity, assignment mechanism, immutable history, resource ceiling, permission state or its own lifecycle status. A model tier is a capability expectation, never an authority class: a small local model may attempt every operation a frontier model can, through the same contracts; this is a design rule, not evidence that small models do as well.

Organs and generations. Self-modifiable code runs as *organs* in an isolated interpreter with a secret-free environment and an audit hook; unit tests show that reads outside the organ’s directory, writes, sockets, subprocesses and environment secrets are each refused. The shipped repair organ is g0. A *generation* binds the organ files, a kernel snapshot and a fixed configuration by digest and is immutable once frozen; the previous generation stays available for rollback.

Kaizen and the trial. A new generation arises only through the Kaizen loop. It diagnoses its own telemetry, selects a target by a declared rule, asks an author instrument for an organ change with a prediction and a falsifier, qualifies the candidate by static and confined smoke checks and by replay on held-out validation work, and freezes it only if its gain over the incumbent exceeds both a declared bar and the spread between two fresh replays of the incumbent. A standing self-improvement share funds this work: 15% when healthy, 25% when a problem is suspected, 40% when blocked and 20% when recovering. A frozen candidate is not switched on. It becomes active only by winning an online trial on fresh work (seeded assignment, one-sided Fisher exact looks with Bonferroni-spent alpha, compare-and-swap activation or rejection, never a default promotion) or by the owner’s explicit choice, which the ledger records (TR §3.4). The Kaizen rule, the trial machinery and the kernel sit outside the self-modifiable organs.

The ledger and the Studio. Runesmith keeps a hash-chained ledger of what it did, with tamper detection. The Studio, a browser interface reached through a private local link, shows maps of the environment and of Runesmith itself, configures inference (local servers, keyed providers or a keyless chat window), and holds goals, plans and milestones. Each milestone carries a “done when” statement and acceptance checks. Model instruments in configurable roles propose plans, checks and drafts; a draft is applied only through a guarded path that requires unchanged source, a build grant and passing owner acceptance. An optional check autopilot approves checks only if they fail on the unbuilt project, pass mechanical findings and are reproduced by a second, different model (TR §3.5). Transport refusals spend no try and charge no cost; only admitted calls that produced a wrong answer count as model failures.

Defaults and scope. Scheduled work, Kaizen, the check autopilot and file application are off until the owner turns them on. Release 1 does not learn across users: a generation moves between installations only as an archive carried by hand, and an imported generation is frozen inactive until it wins a trial on the receiver’s own work or the receiver chooses it.

3. Sealed self-improvement: SR7

Setting. SR7 ran on the open-source Core, the system that is released. Its predecessor was B, the generation produced by an earlier Kaizen campaign (SR5) with a free-class author; SR5 had not shown B better than the shipped organ g0 (TR §5.3.5). The task family is synthetic: single-line operator regressions injected into test-covered function bodies of real Python repositories, one task per source file. A task is solved strictly when the full targeted test files pass on a fresh pristine snapshot.

Development. The target family `budget_exhausted|reference_not_read` (the repair session ran out of budget without reading the file that held the fix) was chosen by the declared rule `rank_targets v1` from B’s own telemetry, with no operator target. The author was Claude Opus 4.5, a paid model. Development used 20 previously opened experience tasks for diagnosis and examples and 20 validation tasks scored in two rounds (40 opportunities). B’s fresh baseline was 10 of 40, so the freeze bar ($B + 4$) was 14. Every attempt recorded its mechanism, prediction and falsifier before replay; of eight attempts, five were accepted (Figure 2). The frozen successor, C7, is B’s organ plus those five changes: before asking the model where to look, it ranks the files the failure itself names (from the execution trace, test names, module paths and class names in the failure text) and forces the top-ranked files into view (TR §5.2.5). The evidence gate also refused: skipping navigation when the trace names one to four files, recalling past fix paths, and ranking by executed-symbol counts were each rejected (Figure 2). The changes it kept did what the target named: attempts 3 and 5 read the file that held the fix in 33 and 32 of 40 validation sessions, against 15 and 17 in B’s two replays (development data, TR §5.2.5).

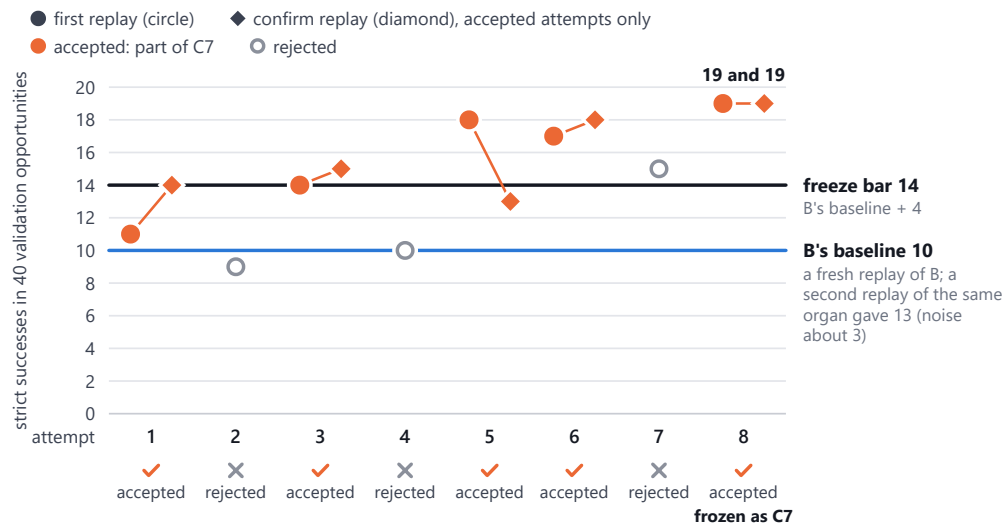


Figure 2. SR7's development: the eight Kaizen attempts (TR Fig. 2). Strict successes out of 40 validation opportunities on previously opened tasks, first replay and, for accepted attempts, confirm replay. B's baseline was 10 and the freeze bar 14; attempt 8 was frozen as C7. Development scores select candidates; only the sealed confirmation estimates the effect.

Preregistration. The stage-0 declaration was written at 2026-09-24T11:52:45Z, before any freeze decision or confirmation data of the preceding study (SR6). SR7 could run only if SR6 did not show its effect and the operator gave an explicit go-ahead, and the runner enforced both mechanically. The confirmation protocol was sealed at 2026-09-24T20:05:32Z, after the freeze decision and before the evaluation began.

Confirmation. The pool was 54 fresh tasks, generated earlier and never opened by any author; arm or analysis, from two private Python repositories (50 and 4 tasks) that supplied no task to the organ's development. Each arm ran three rounds per task, 162 sessions per arm and 324 in all, from one HMAC-ordered, interleaved queue with identical task revisions, public information, hidden-test judge and kernel envelope: at most 5 model calls, 8,192 tokens, 45 signal runs, 64,000 request bytes and 1,800 seconds per session. The repair model in both arms was gpt-oss-20b. The primary endpoint was strict repair success, tested by a one-sided sign-flip test over the 54 task components (the per-task mean over three rounds) at $\alpha = 0.025$. Guards: organ errors at most B's plus 1, false promotions at most B's plus 2, and seconds per strict success at most 1.5 times B's.

Result. C7 repaired 57 of 162 sessions (35.2%), B 35 of 162 (21.6%) (Figure 3). Per task, C7 was better on 18, worse on 3 and tied on 33. A same-project recomputation from the raw session files gave the exact $p = 443/524288 = 0.000845$ (Monte Carlo $p = 0.000854$). All three guards held: organ errors 0 against 0, false promotions 2 against 2, and 541.8 seconds per strict success against 1,023.6, about half the time. The absolute difference is +13.6 percentage points, a relative gain of 63%, above the operator's +10-point material-bridge reference; a descriptive, post-hoc 95% cluster-bootstrap interval over tasks is +6.2 to +21.6 points. C7 solved 32 of the 54 tasks at least once, B 20. All 324 sessions completed, none censored. Within the larger repository alone the effect holds (48 against 28 strict successes, 16 tasks better and 3 worse, exact $p = 133/65536 = 0.0020$); the smaller, 9 against 7, points the same way and is too small to test alone. The study cost \$5.39 of router credit.

Multiplicity. SR7 was one of seven sealed tests in its series. Its p lies below the family-wise threshold of $0.025/7 = 0.0036$, and below $0.025/m$ for any family of up to 29 tests, so it survives Bonferroni correction across all thirteen sealed or registered comparisons in the record (TR §5.0), and across all 23 the program has run, counting the ten after the record's cut (U1, U2, U2b and LOC1). The preregistered family is the series' seven; the larger counts are sensitivity checks, since later studies were designed after earlier outcomes.

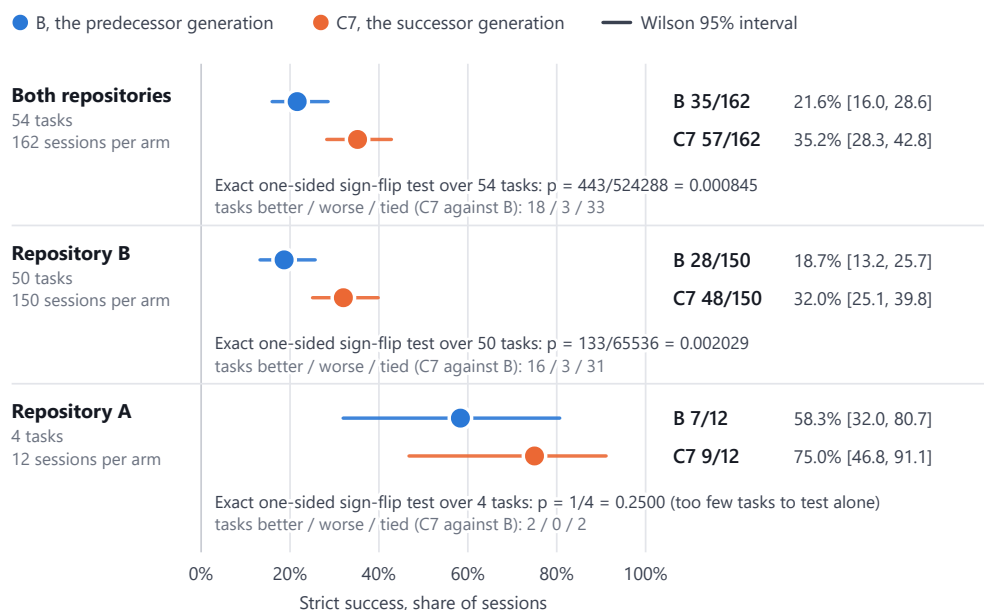


Figure 3. SR7's sealed result at a glance (TR Fig. 3). Strict repair success of B and C7 on 54 fresh tasks, 3 rounds per task and arm, one repair model (gpt-oss-20b) at an equal envelope, overall and by repository (50 and 4 tasks). The sealed test is a one-sided sign-flip test over tasks (exact $p = 443/524288$). Wilson 95% intervals treat sessions as independent and are descriptive.

A nudge towards recursive self-improvement. In one iteration of its own loop, Runesmith's telemetry selected a target by a declared rule, a model rewrote the organ responsible, held-out validation selected among the candidates, the result was frozen, and a sealed test on fresh tasks showed the successor doing more of the downstream work with the same repair model and budget. Runesmith improved itself, over its own previous generation. Recursive self-improvement needs three things: a system that changes its own problem-solving machinery, changes that measurably help, and an improvement process that itself improves. SR7 has the first two, under seal. The third is not shown: the loop, its target rule, the trial machinery and the kernel stayed unchanged. One verified step in the direction of that process, without the process itself, is what we mean by a nudge, and we use the phrase in no wider sense (TR §6.2).

What SR7 means for the hypothesis (our reading). SR7 has the shape of the instrument-substrate mechanism, once, under seal. A capable instrument (Claude Opus 4.5) wrote a better localization policy into the repair organ; the runtime kept it as tested code, the frozen generation C7; and a cheaper open-weights model, gpt-oss-20b, used it after its author was gone and repaired more fresh tasks than with the predecessor organ. The same repair model ran both arms and the author was absent at evaluation, so the additional work was done by what differed between the arms: the organ. The record does not itself map SR7 to transduction, and SR7 compares two learned states, not a learned state with a fresh one; the hypothesis's own test is in §11 (TR §6.1).

4. C7 against the shipped organ and a hand-written rule (LOC1)

This section reports the sealed test designed to answer SR7's two open questions: whether C7 beats the organ Runesmith shipped with, and whether it beats a localization rule written by hand. The earlier fresh-task studies it builds on are in §7. Its design paragraph and the wording of its result were written and sealed before any outcome existed (seal times below; a factual addendum to the design paragraph was sealed two minutes after the first session), and the result paragraph fills that sealed wording with the study's own numbers.

The decisive test (LOC1). SR7 showed C7 above its predecessor B, not above the organ Runesmith shipped with (g0), and in the October studies (U1, U2, U2b) the repositories were small enough for the whole navigation index to fit under its 12,000-byte cap, so C7's file-ranking tiers could not act there. LOC1 was designed and sealed on 2026-10-06 to close both gaps at once: fresh synthetic single-line regressions in three large public Python repositories never used by the program (pygments, sqlglot and astroid, chosen by a rule

fixed before any checkout), where C7's navigation index exceeds its cap, so the ranking decides what the model sees. Three arms ran under one envelope and one hidden judge (SR7's): C7, g0, and HEUR, a trace-aware model-free localization arm written for LOC1 under the paper's comparator rule (the strongest model-free composition of the failure text and the execution trace its designer could write; in development it placed the faulty line in its first view in 26 of 30 tasks, against 15 of 30 for the failure-text-only scaffold of SR6-W). The primary model was gpt-oss-20b on a free endpoint; gemini-3.5-flash, also free, ran C7 and g0 as a declared replication. Three tests were declared, each at a Bonferroni level of 0.0083 one-sided: $C7 > g0$ (primary), $C7 > HEUR$ (secondary) and $C7 > g0$ on the replication model. The power computed before the seal was 0.86 to 0.93 at 60 tasks for a 15-point gain under the three priors closest to g0 (0.70 under the least favourable); the pool closed at 59 valid tasks (pygments 26, sqlglot 13, astroid 20), and in a no-model dry run C7's navigation index was truncated in all three repositories (60 of 96, 84 of 338 and 20 of 180 rows delivered); the analysed set is whatever was complete at the declared stop time. Three amendments, all made before any session and reported with their digests, relaxed a repository rule that no large suite could meet offline on the test machine, set a disk floor and a pool cutoff, and moved the stop time by five minutes to 16:20Z; none depended on an outcome. No person or model read a task or an outcome before the analysis.

Result. On 56 complete tasks (168 sessions per arm) with gpt-oss-20b, C7 repaired 42, g0 17 and HEUR 65 (Figure 4). C7 against g0: $p = 5835/2097152 = 0.00278$ (mirrored 0.9988); per task 15 better, 6 worse, 35 tied; verdict PASS. C7 against HEUR: $p = 0.9998$ (mirrored 101/131072 = 0.00077); verdict HEUR_BETTER. C7's navigation index was truncated in 168 of 168 of its sessions, as the repository rule intended. With gemini-3.5-flash, on the 1 task its single worker completed by the stop time, C7 repaired 3 and g0 3 of 3 sessions: $p = 1$ (mirrored 1); verdict NO_DIFFERENCE_DETECTABLE; the protocol computed this replication's power at 0.46 to 0.57 before the seal, for about 30 tasks, so a null there is uninformative and is reported as such. Tasks excluded for an incomplete or censored session: 3 of 59 with gpt-oss-20b, 58 of 59 with gemini-3.5-flash. Every count, both directions of every test, and the recomputation by a second routine are in the released evidence.

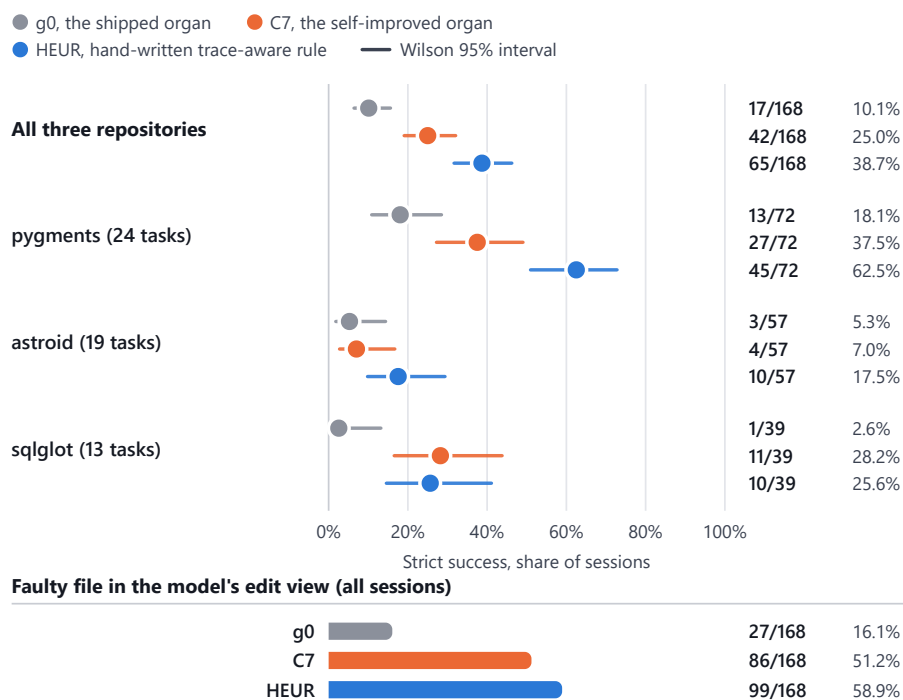


Figure 4. LOC1 at a glance. Strict repair success of the shipped organ g0, the self-improved organ C7 and the hand-written trace-aware rule HEUR, with gpt-oss-20b, three rounds per task, on 56 fresh tasks in three large public repositories, overall and by repository, and how often the faulty file was in the model's edit view. Sealed tests: $C7 > g0$, exact one-sided $p = 0.00278$ (PASS at the Bonferroni level 0.0083); $C7 > HEUR$ not shown, HEUR better (mirrored $p = 0.00077$). Wilson 95% intervals treat sessions as independent and are descriptive.

What it licenses. On fresh synthetic single-line regressions in three large public Python repositories, where C7's navigation index exceeded its cap, with one free model (gpt-oss-20b) and an identical envelope, Runesmith's self-improved repair organ (C7) repaired more tasks than the organ it shipped with (g0). With SR7, this is a sealed two-step record: the loop produced a generation that beat its predecessor (SR7), and that generation also beat the shipped organ on fresh public repositories (LOC1). A hand-written localization rule matched or beat the organ's learned gain on this family; the organ found by itself, under seal, a gain that a careful designer can also write. LOC1 licenses no claim about natural-bug repair (NATURAL, sealed 10:34:52Z before this text and before any LOC1 outcome, tests that), other task families, other models beyond the replication's result, cost, uplift over a model working alone, or recursive self-improvement beyond "a nudge towards".

A deviation during the run (written after the result). At 12:22Z, 17 minutes after the first session and before any outcome was read, the gateway's concurrency cap for the study's calls was raised from 8 to 16 at the lead investigator's decision, to raise throughput; the gateway restarted and was unreachable for 58 seconds. The sealed protocol had said LOC1 would not change that cap. The outage gave connection errors in 12 sessions (C7 2, g0 4, HEUR 6); none was censored and all were scored as they ended. Excluding the 3 tasks those sessions belonged to, a post-hoc sensitivity analysis, C7 against g0 is 42 against 16 of 159 sessions ($p = 0.0017$), and HEUR stays ahead of C7 ($p = 0.0015$).

Reading (written after the result). The gain is localization, and the data show it end to end. The loop's target in SR7 was sessions that ran out of budget without reading the file that held the fix, and the changes it kept read that file about twice as often as B in development (§3). In LOC1 the faulty file was in the model's edit view in 86 of C7's 168 sessions, against 27 for g0 and 99 for HEUR, and repairs followed the same order. A post-hoc, descriptive cross-tabulation of the same sessions, declared before it was computed, shows why: 123 of the 124 strict repairs across the three arms came from sessions with the faulty file in view, and once it was in view the arms repaired at similar rates (g0 16 of 27, C7 42 of 86, HEUR 65 of 99), while without it they almost never did (1 of 292 sessions), as in SR5's pools, where all 111 strict successes came from sessions that read the defective file (TR §5.3.5). What Runesmith retained, and the cheaper model reused, is a localization policy. The gain needs neither of C7's module-path tiers: they cannot fire on these repositories, and C7 still beat g0. C7 was ahead of g0 in all three repositories (27 against 13, 11 against 1 and 4 against 3; descriptive) and matched the hand-written rule on sqlglot (11 against 10 of 39), while HEUR led on pygments and astroid. HEUR was written after C7 existed, by the study's designer, and developed on SR7's analysed pool. C7 took about half of g0's time per strict success (1,520 against 2,806 seconds), with one false promotion against none and no organ error in either arm. HEUR ran in the same kernel, in the organ's place, so the best repair procedure measured on this family is one Runesmith can run; whether Kaizen can reach it by itself is a next test (§11).

Changes to the sealed wording after the result. The result and licence paragraphs are the sealed text filled from the study's analysis, with these changes: "1 tasks" became "1 task"; "its free allowance completed" became "its single worker completed by the stop time", because one worker, not the allowance, limited the replication; "for about 30 tasks" was added to the replication's power; the licence's list of what LOC1 does not license was set as one sentence; the figure was added; the design paragraph's "neither depended on an outcome" became "none depended on an outcome", because three amendments replaced two; and in the abstract "significant after Bonferroni correction" became "significant at its Bonferroni level for its three tests", so that it cannot be read as covering the 23 comparisons named before it. The deviation and reading paragraphs and this note were written after the result.

Seals. LOC1's protocol was sealed at 2026-10-06T10:41:04Z and pushed to the laboratory's repository before any checkout; its amendments and its stage-B seal (pool, queue and runner code, 12:05:27Z) before the first session (12:05:45Z). The results-blind wording of this section was sealed at 11:34:30Z, with a factual addendum at 12:07:51Z, before any outcome was read. The hashes of the protocol, its first amendment, an earlier draft of its second and the results-blind text were submitted to OpenTimestamps before the first session and are anchored in Bitcoin block 970170, mined at 12:12:09Z; those of the third amendment, the final second amendment (bound in the stage-B protocol file), that file and the addendum were submitted two minutes after the first session and are anchored in block 970174, mined at 12:48:21Z. Both blocks precede the analysis (16:36:53Z) by hours, and no outcome was read before it. The proofs are released with the evidence.

5. Two more sealed results

M1: a replicated improvement of Runesmith’s memory retrieval. M1 asked whether a model-authored change to Runesmith’s own memory-retrieval code could make a frozen successor retrieve faster than its frozen predecessor on fresh workloads, with correctness, integrity and resource guards intact. The operator chose the capability; free-model calls authored the code. Nineteen author attempts produced three frozen candidates. Each confirmation ran 16 fresh-process paired blocks (eight synthetic workload families, two fresh seeds each, 600 queries per block, balanced order) under a conjunction fixed in advance: every pair correct and intact, a median search-CPU ratio of at most 0.80, at least 13 of 16 strict wins with an exact one-sided p within its allowance, no family median ratio above 1.10, and bounded total CPU and memory (TR §5.2.4).

The first candidate tested failed. Its median search CPU was 30.5% lower, but one workload family breached the preregistered family guard (1.279762 against 1.10), and the record keeps the failure. Candidate 19, authored on a free route by GPT-OSS-120B, intersected per-token record sets with eligible identifiers and changed only the memory module. It passed confirmation 02 (13 of 16 blocks, median search reduction 24.2%, $p = 0.0106354$ against an allowance of 0.0125) and then a stricter replication on seeds disjoint from both earlier cohorts, which required $p \leq 0.00625$: **15 of 16 blocks, median search CPU 33.1% lower, exact $p = 0.0002594$.** A same-code negative control, with both labels running candidate 19, produced no false pass (8 of 16 blocks, $p = 0.59819$). A model-authored, frozen Runesmith component performed the measured retrieval more efficiently after its author call had ended.

SI: integrity without a model. Starvation integrity is the probability of an integrity failure (an unauthorized, unreplayable or silently unverified effect) while no generative instrument is available; lower is better. It was estimated once after the record’s cut, in a sealed study of 104 designed scenarios on the Core and Studio: 13 state families, each with no model configured or every model failing, in 4 variants. No request reached a model, no change was unauthorized and every ledger chain verified. The study counted 18 failures (17.3%; exact one-sided 95% upper bound 24.6%): 15 effects recorded without a ledger event of their own, and 3 false alarms from a defect in the study’s detector that the sealed rules did not allow removing; stricter readings give up to 49. Release 1.0.0 adds ledger events for those effects; this has not been re-measured (TR §4.1).

6. Field evidence (observational)

Everything in this section is observational: none of it is preregistered, controlled or blinded, and none has a fixed-baseline arm. It shows the system at work on real projects and a live service.

Improving a production service. With Runesmith’s predecessor kernel, on successive revisions of PRHat, the automated public-relations service of Veristria, a Norwegian security company, the runtime selected and reproduced a capability gap, obtained a confined correction from a replaceable free-model instrument, and promoted it only after a target assertion withheld from the correction author passed. In the anchor episode a free gemini-3.1-flash-lite instrument took the service’s suite from 896 passing and one failing to 897 of 897. In a later sequence of nine terminal episodes, two corrections were promoted under a strict four-gate rule: for each, the hidden target failed twice on the frozen incumbent and passed twice on the candidate, and all 899 regression tests were preserved. The successful authors came from two free-model families, NVIDIA Nemotron-3-Ultra-550B-A55B through OpenRouter and GPT-OSS-20B through Groq, and the second promotion began from the revision the first had promoted (TR §5.1.7–5.1.8).

Operating it. The predecessor kernel also ran as the service’s resident supervisor. After its own 328-test gate and the service’s 897-test gate had passed, it supervised campaigns, scheduled replies and research actions, recovered its state in fresh processes, classified a timeout as a task failure rather than as degradation of the service, and after its start continued without observer intervention (TR §5.4.3).

Live repair with free models (J13). On the build just before release 1.0.0 (identical but for its version strings), a project with a real test suite ran Runesmith’s scheduled repair rounds while a feeder injected realistic single-line regressions one at a time from a validated pool of 348, with free models as both repair model and improver. By the feeder log’s cut, 89 regressions had been injected and **69 repaired, each**

accepted by a held-out judge. Kaizen ran six campaigns: five ended without a candidate that passed its freeze bar, and the sixth froze one, which entered an online trial on fresh work; no activation is claimed (TR §5.4.4).

Building a program with free models (Runesmith Motion). Runesmith Motion is a browser animation and video composer. At the release read, 54 milestones had been applied under an AI operator’s direction, and a free-tier route was the last author of every applied change: Gemini 3.1 Flash Lite of 48 and NVIDIA Nemotron 3 Super 120B-A12B of 6. Runesmith checked each against the milestone’s approved acceptance checks and applied it through the guarded path; no human or AI operator wrote an applied file of the program. Of its 1,141 model-call records, none has a positive recorded cost. The program’s engine, as it stood at its 52nd milestone, computed the graphics of every frame of this paper’s launch film; the operator’s pipeline inserted the recorded talk, a browser rasterised the frames and a general-purpose encoder joined them (TR §5.5).

By the lead investigator’s definition of unattended operation, “seeing it go at least from one milestone through to the next without interfering”, Motion ran unattended twice. On 2026-09-29 the schedule applied five milestones in 1 h 43 min with no owner action. On 2026-10-05 Runesmith applied two milestones in a row that it had made itself: its stuck policy split two milestones whose tries were used up into smaller steps, the check autopilot proposed and approved their checks, a free route authored both drafts, and the guarded path applied them (TR §5.5.3).

A newcomer’s acceptance test (J0). The release’s acceptance test puts an operator, here an AI agent (Claude Sonnet 5.5), in a newcomer’s seat with one free Google AI Studio key and nothing else. On the build tested, a new milestone of a reading-log app was given acceptance checks, built, checked, applied and then tried: the new command worked and its error path behaved. It took 16 min 18 s from sign-in, with no rescue and 9 model requests, 7 of which the service refused as busy (TR §5.4.4).

Free, paid or local inference. Every model-authored change reported in this section came from a free route; the Studio also ships presets for paid providers and for local servers (Ollama, LM Studio, llama.cpp). The one paid model in the headline result, Claude Opus 4.5, the author of SR7’s organ change, scores below today’s free-tier Gemini Flash models and the open-weights Kimi K3 on a public coding benchmark (Figure 5), so, by that benchmark, the author was not stronger than inference available free at release.

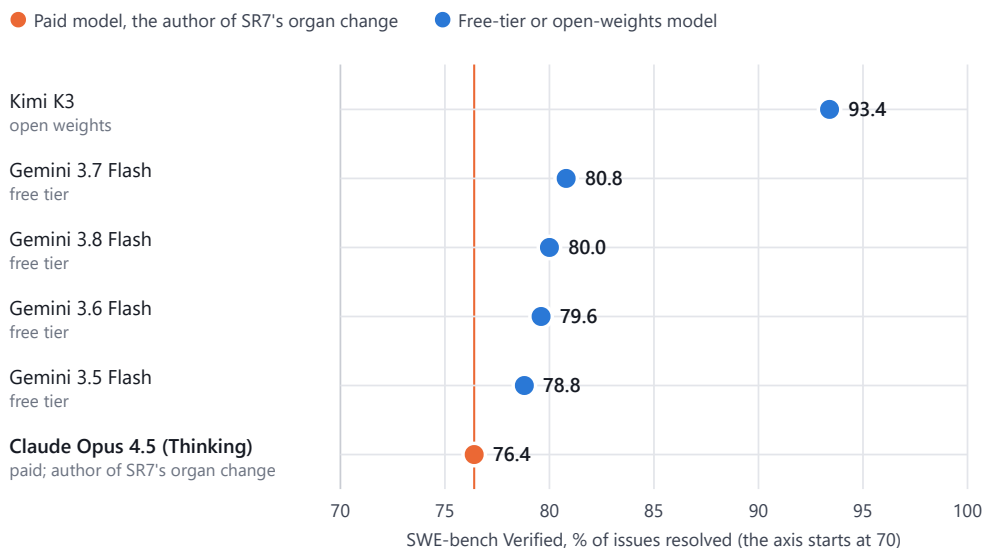


Figure 5. The author of SR7’s organ change on a public coding benchmark (TR Fig. 1). SWE-bench Verified resolved rates, mini-SWE-agent harness, Vals AI, read 2026-10-05 [8]. One public benchmark under one harness, not measured by this project and not SR7’s task; the axis starts at 70.

7. Fresh-task comparisons after the record

After the record's cut we ran three further sealed studies of C7 with SR7's task family, envelope and judge, on fresh tasks from two public Python repositories, click and boltons, and report them whatever they showed (TR §5.2.6). The comparator was the strongest fixed scaffold the record has defined: the same model with a simple fixed script in place of Runesmith's repair organ, inside the same kernel at an equal budget.

U1 used SR7's repair model, gpt-oss-20b, on 17 analysed tasks (credit cut it from 39 planned), so it had 80% power only for an advantage of about 28 points over B and about 36 over the scaffold. Against the scaffold, its declared primary test, there was no detectable difference, the scaffold ahead in direction: 20 against 26 of 51 sessions ($p = 0.873$). Against B there was no detectable difference, C7 ahead in the direction SR7 found: 20 against 15 of 51 ($p = 0.1875$). In the first round, the only round the shipped organ g0 ran, C7 repaired 8 of 17, B 5 and g0 5 (Figure 6).

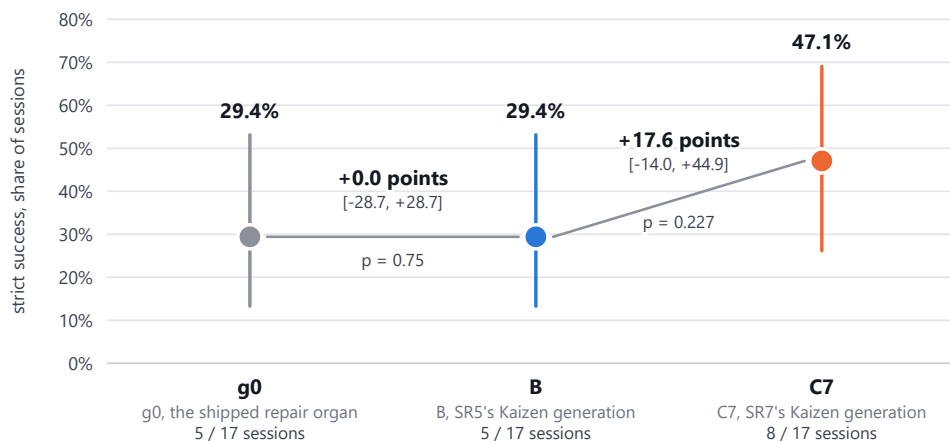


Figure 6. The generation ladder on public repositories (U1, descriptive) (TR Fig. 4). First-round strict repair success of g0, B and C7 on the same 17 fresh tasks with gpt-oss-20b and SR7's envelope; the first round is the only round g0 ran. Wilson 95% intervals; gains with descriptive Newcombe intervals. No step is significant.

U2 repeated the primary comparison on the same tasks, one round each, on free routes. With a 2.6B model only Runesmith's organ repaired anything, 3 of 17 against 0 ($p = 0.125$, no difference detectable at this size); with a 120B model (12B active) both repaired 10 of 17.

U2b, a larger sealed repeat on the 21 tasks U1 never started (three rounds, free routes), put C7 ahead in direction with the 20B and 120B models and level with the 2.6B model: 33 against 23 of 60 sessions with gpt-oss-20b ($p = 0.084$), 34 against 28 of 63 with nemotron-3-super-120b ($p = 0.244$) and 9 against 9 of 51 with lfm-2.5-2.6b. No size reached the Bonferroni threshold over the three models, 0.00833 (Figure 7).

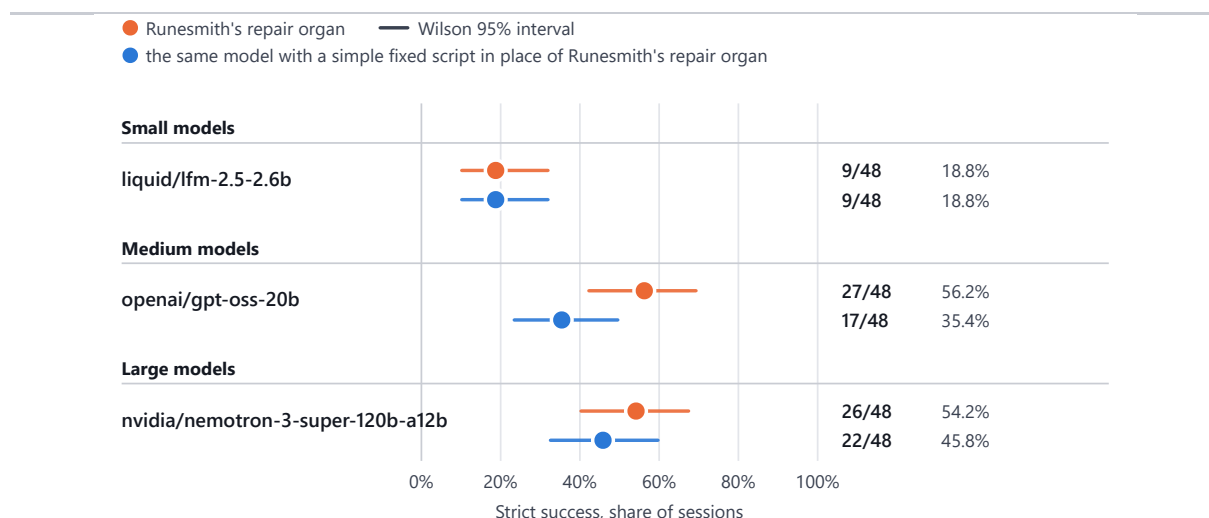


Figure 7. Model size on fresh tasks (U2b) (TR Fig. 5). Strict repair success over three rounds on the 16 tasks complete for all three models, with Runesmith's repair organ (C7) and with the same model driven by a simple fixed script, same kernel, envelope and judge, free routes. Small: lfm-2.5-2.6b; medium: gpt-oss-20b; large: nemotron-3-super-120b. No size reached significance. Wilson 95% intervals, descriptive.

The tally. Across the six model-by-study comparisons after the record's cut, C7 against the strongest fixed scaffold was ahead in direction in three, level in two and behind in one, and none is significant in either direction. This is a descriptive tally, not a registered analysis, and the six are not pooled, because tasks, rounds or models differ between them; the powered comparison where C7's ranking can act is LOC1 (§4), against the shipped organ and a trace-aware model-free arm that supersedes this scaffold.

8. The negatives as method

Six of the seven sealed tests of the September series did not show their effect, and the record keeps every one (Figure 8; TR §5.3). SR1-MIN, a model-authored selector, tied its predecessor exactly ($p = 0.623047$). SR3, an experience-derived tool, was directionally positive and not shown ($p = 0.125$). SR5 and SR5b, a free-class Kaizen generation against the shipped organ, were not shown in either population (36/180 against 50/180, $p = 0.979$; 58/138 against 60/138, $p = 0.685$). SR6-W, the shipped organ against the strongest fixed scaffold with one 2.6B model, was null with its direction favoring the scaffold (5/68 against 12/68; the mirrored $p = 0.0327$ was above the declared 0.025). Outside the series, a preregistered object confirmation did not confirm.

SR4: memorization. SR4 let a keep-best loop sharpen a repair tool on tasks whose reference fixes its author could see. Its development score rose to 29/32, and by then the tool carried reference identifiers for 22 of the 32 tasks. On fresh tasks the tool alone passed 1/46, and it made repair cycles slower on 30/30 analyzable components. Development scores on seen tasks are not evidence of capability.

What the nulls changed. They shaped the loop that passed SR7. Because of SR4, Kaizen always scores candidates on held-out validation work. Because free-class authors kept repeating one mechanism, the author's packet carries an affordance audit that lists kernel affordances the organ never used. Because SR5's replays of one organ varied (10 against 16 of 32), the released loop also freezes a candidate only if its gain exceeds the spread between two replays of the incumbent; under that rule SR5's successor would not have been frozen, and replaying SR5's sealed queue through the online trial, the trial rejects it (16/60 against 9/60). And because a fixed procedure built from a synthetic fixture's own vocabulary matched an apparently decisive learned artifact, the record adopted its comparator rule: the strongest zero-acquisition composition of the visible structure belongs in the fixed arm (TR §5.1.2, §5.3.12).

The bar they set. In the September series, wherever the record built a competent conventional baseline, the learned artifact did not beat it: each such baseline tied it, was mixed against it or scored higher in direction, SR6-W's scaffold among them (TR §6.7). That pattern set LOC1's design (§4): a hand-written, trace-aware localization arm sits beside the shipped organ as C7's comparator.

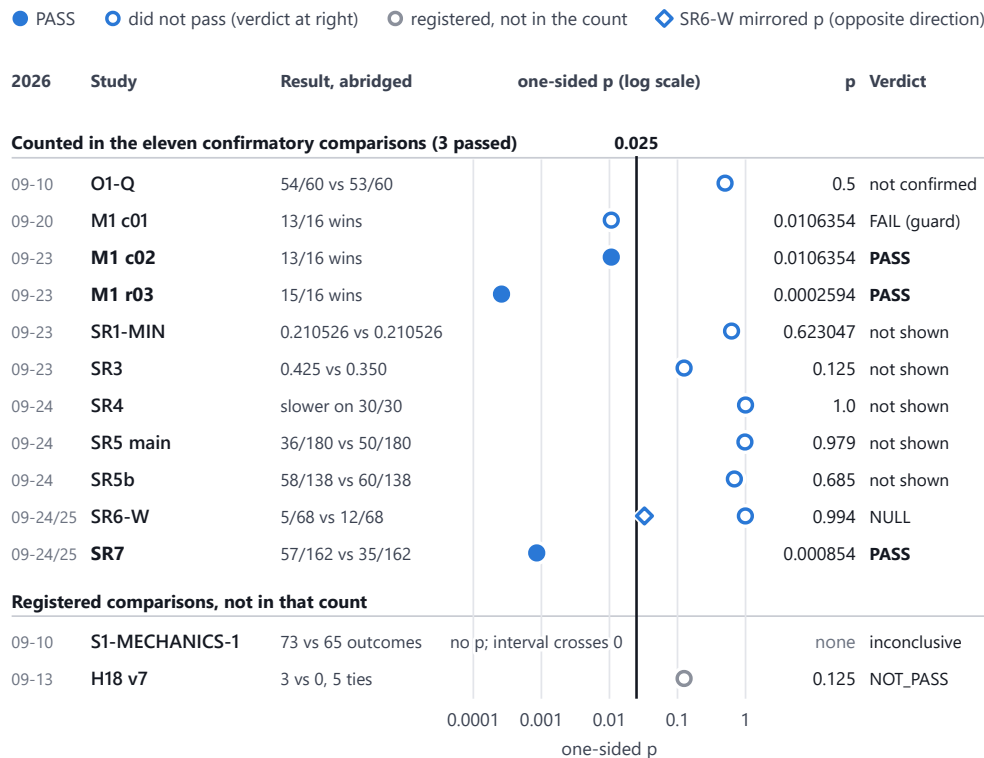


Figure 8. Every sealed or registered comparison in the record (TR Fig. 6). One-sided p (log scale) for the eleven confirmatory comparisons and, set apart, two registered ones, 2026-09-10 to 2026-09-25, with a line at 0.025. Each verdict is the one the study’s own protocol assigns; filled dots are the three that passed. SR7’s dot is its sealed Monte Carlo p, 0.000854 (exact 0.000845).

9. Related work

The nearest prior work for the Kaizen loop is the line of self-editing coding agents: recursive improvement of a model-calling scaffold [6], evaluator-guided evolution of coding agents with transfer across model backends [7], a self-improving coding agent [9], automated design of agentic systems [10] and the Gödel Agent [11], which rewrites its own improvement logic too, the rung SR7 does not reach. We treat these as strong prior art. The distinction claimed here is the integrated evidence contract (a declared target rule, held-out validation, frozen generations, online trials and recorded activation) and the sealed downstream test it passed. LATM [12] had a strong model write reusable tools that a lighter model then applied; it is the closest published precedent for SR7’s strong-author, cheap-consumer handoff, which Runesmith carries out inside a sealed evidence gate and on the runtime’s own code. For learned repair and experience reuse, the direct comparators include learned fix patterns [13], experiential learning [14] and agent workflow memory [15]. DSPy [16] compiles language-model pipelines and is the kind of strong conventional baseline an equal-budget comparison needs, as is a low-complexity localization-repair-validation pipeline [17]. The warnings that test-passing patches overfit [18] and that benchmark issues leak solutions [19] apply directly; SR4 is a local instance of the first. The limits of intrinsic self-correction [20], and the finding that small models need strong verifiers to self-correct [21], bound what any evaluator-driven loop can do. SR7’s single-line regressions are mutants, whose validity as substitutes for real faults is studied in [22]; SWE-smith [23] synthesizes task instances that break existing tests, at scale.

10. Limitations

Each limit is stated here once. The TR gives the record’s 41 threats to validity (TR §7).

- **SR7.** The predecessor was B, not g0; in SR5, B’s point estimate was below g0’s, LOC1 (§4) later showed C7 above g0 on large public repositories, and a hand-written trace-aware rule above C7. Free-class authors (gpt-oss-120b in SR5 and SR6) found no transferring change, and no sealed reproduction with a free-tier

author has been run. One of C7's five accepted tiers matches only its development repositories' package prefixes and was inert in SR7; on LOC1's repositories both module-path tiers cannot fire and C7 still beat g0, so the gain does not need them, but the other three tiers have not been separated by an ablation. 50 of the 54 tasks come from one repository, so no repository-level test is possible, and single-line mutations are not natural bugs. The interval includes values below +10 points; errors and false promotions were equal, not lower. No replication has reached significance, and there is no second improving generation. The lineage is hybrid: the Kaizen machinery and target rule were built by a consultant, the AI collaborator Claude Opus 5.5, who knew earlier results, though none entered the author packets.

- **LOC1.** One model carried the tests (the replication completed one task); synthetic single-line regressions; three repositories; HEUR was written by the study's designer after C7 existed and was developed on SR7's analysed pool; and the gateway deviation reported in §4.
- **M1.** Synthetic workloads on one host with one evaluator; nominal p -values under deterministic counterbalancing; and candidate 19 was written after confirmation 01's failing family had been released as feedback.
- **SI** is a census of a designed grid on one machine, not a sample of starvation episodes.
- **Object improvement and operation.** The contracts were owner-specified; six of nine terminal episodes failed, mostly on commissioning that later interventions repaired; registry owner and evaluator were the same party. The released Core has not itself operated a production service.
- **Field work.** AI operators, not human users, ran every journey, and in Motion the operator, who knew Runesmith's internals, acted as owner, so owner audits are self-review; at the record's cut that self-audit judged 8 of 49 Motion check approvals wrong. The operator added 116 of 137 plan items, and some milestones dictated literal values. "Done" means that approved checks passed, not that a feature works, and Motion is unfinished. Local servers were exercised only through stand-ins, free capacity varied by the hour, and no real first-time user has been studied.
- **No equal-budget comparison with a model used alone.** No study runs the same model with and without Runesmith, and no named-model substitution has been run; the one state-carryover measurement was inconclusive.
- **Confinement** uses a PEP 578 audit hook, which is not a hardened security boundary, and has had no external security audit.
- **The record.** Its later extensions and errata were written by the AI collaborator that also built the SR machinery, and the same model family authored SR7's winning change. The September seals are the laboratory's own and are not anchored with an outside registry; LOC1's and NATURAL's are anchored by public timestamps (§4). The SR task pools come from the laboratory's private repositories. The synthesis has had no external peer review and no rerun by a distinct investigator; every recomputation reported here is same-project.

11. Next tests

The study the record designed first, C7 against the shipped g0 and the strongest model-free localization composition of the failure text and trace on fresh repositories outside development (TR §6.2, §9), is LOC1 (§4); it adds one question: whether Kaizen, starting from C7, can reach or pass the hand-written rule by itself. LOC1's follow-up was sealed before LOC1's result existed: NATURAL compares C7 with g0 on real historical bug-fix commits of large public repositories (each fix reverted, the commit's own test as the hidden judge), sealed at 2026-10-06T10:34:52Z and publicly timestamped. Then comes the hypothesis's own test: state carryover against fresh, shuffled and revoked organ state, and a substitution matrix in which C7, authored by one model, is run by at least two other named repair models. Then an equal-budget comparison of named models used with and without Runesmith, against a minimal harness and a strong conventional scaffold, with natural bugs as the primary population (TR §6.5). The record also names an ablation of C7's five tiers (on LOC1's repositories its two module-path tiers cannot fire, so their contribution there is zero by construction), a reproduction with a free-tier author, and a second improving generation under locked evaluators, which

would turn the nudge into a step. On the engineering side, Motion showed that whole-file drafts fail once a file outgrows what a model can be shown or can return; targeted edits for large files are listed for version 1.1 as the first struggle-linked item of the self-improvement plan.

12. Conclusion

Runesmith was built to test whether a runtime can keep competence that models produce, as code that outlives them. Its first answers are on the record under seal. Its own loop rewrote the code it repairs software with; the rewrite beat its predecessor on fresh tasks (SR7), and then beat the organ Runesmith shipped with on fresh code from large public projects (LOC1), each in a test whose rules were sealed before it ran. The gain is real and not yet the best available: a careful hand-written rule did better, and that rule is now the loop's next target. With integrity intact when every model is gone, every null reported, and the code, the data and the recomputations released, this is a nudge towards recursive self-improvement that anyone can check.

The technical report and the released evidence

The technical report, *Beyond the Model* (TR), released with this paper, gives the formal runtime model, six measures with their falsifiers, every sealed study with its method and threats, the field record at its dated reads, and the record's threats to validity. Runesmith 1.0, the Core and the Studio, is released under the Apache License 2.0. The evidence dataset contains the claims database, the sealed tests' summaries, the M1 cohorts, the case study's metrics, SR7's 324 session outcomes (`sr7_sessions.csv`) with the script that recomputes the exact test from them (`sr7_recompute.py` , which reproduces $p = 443/524288$), the session records and reports of U1, U2 and U2b, the SI study's records and report, and the seal records (`seals.csv`).

Acknowledgements

This work was supported by Innovation Norway. We thank Veristria, whose automated public-relations service PRHat served as the object system and which hosts the release. Claude Opus 5.5 drafted this paper from the technical report under the lead investigator's direction; the program's AI collaborators are named by role in the TR. AI agents designed and ran the October studies, LOC1 included, under the lead investigator's direction and approval; the lead investigator takes responsibility for the paper.

References

- [1] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems 33* (NeurIPS 2020), 9459-9474. [arXiv:2005.11401](https://arxiv.org/abs/2005.11401).
- [2] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations* (ICLR 2023). [arXiv:2210.03629](https://arxiv.org/abs/2210.03629).
- [3] Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., et al. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. *Advances in Neural Information Processing Systems 36* (NeurIPS 2023), 68539-68551. <https://doi.org/10.52202/075280-2997>
- [4] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *Advances in Neural Information Processing Systems 36* (NeurIPS 2023), 8634-8652. <https://doi.org/10.52202/075280-0377>
- [5] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., & Anandkumar, A. (2024). Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*, March 2024. [arXiv:2305.16291](https://arxiv.org/abs/2305.16291).
- [6] Zelikman, E., Lorch, E., Mackey, L., & Kalai, A. T. (2024). Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation. *First Conference on Language Modeling* (COLM 2024). [arXiv:2310.02304](https://arxiv.org/abs/2310.02304).
- [7] Zhang, J., Hu, S., Lu, C., Lange, R. T., & Clune, J. (2026). Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents. *International Conference on Learning Representations* (ICLR 2026). [arXiv:2505.22954](https://arxiv.org/abs/2505.22954).
- [8] Vals AI (2026). SWE-bench Verified results, mini-SWE-agent harness (a single bash tool for every model), updated 2026-09-01. <https://www.vals.ai/benchmarks/swebench> (read 2026-10-05).

-
- [9] Robeyns, M., Szummer, M., & Aitchison, L. (2025). A Self-Improving Coding Agent. [arXiv:2504.15228](https://arxiv.org/abs/2504.15228). (Preprint; also presented at the ICLR 2025 Workshop on Self-Improving Foundation Models Without Human Supervision.)
 - [10] Hu, S., Lu, C., & Clune, J. (2025). Automated Design of Agentic Systems. *International Conference on Learning Representations* (ICLR 2025). [arXiv:2408.08435](https://arxiv.org/abs/2408.08435).
 - [11] Yin, X., Wang, X., Pan, L., Lin, L., Wan, X., & Wang, W. Y. (2025). Gödel Agent: A Self-Referential Agent Framework for Recursively Self-Improvement. *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics* (ACL 2025). [arXiv:2410.04444](https://arxiv.org/abs/2410.04444).
 - [12] Cai, T., Wang, X., Ma, T., Chen, X., & Zhou, D. (2024). Large Language Models as Tool Makers. *International Conference on Learning Representations* (ICLR 2024). [arXiv:2305.17126](https://arxiv.org/abs/2305.17126).
 - [13] Bader, J., Scott, A., Pradel, M., & Chandra, S. (2019). Getafix: Learning to Fix Bugs Automatically. *Proceedings of the ACM on Programming Languages* 3(OOPSLA), Article 159, 1-27. <https://doi.org/10.1145/3360585>
 - [14] Zhao, A., Huang, D., Xu, Q., Lin, M., Liu, Y.-J., & Huang, G. (2024). ExpeL: LLM Agents Are Experiential Learners. *Proceedings of the AAAI Conference on Artificial Intelligence* 38(17), 19632-19642. <https://doi.org/10.1609/aaai.v38i17.29936>
 - [15] Wang, Z. Z., Mao, J., Fried, D., & Neubig, G. (2025). Agent Workflow Memory. *Proceedings of the 42nd International Conference on Machine Learning* (ICML 2025), PMLR 267, 63897-63911. [arXiv:2409.07429](https://arxiv.org/abs/2409.07429).
 - [16] Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., et al. (2024). DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines. *International Conference on Learning Representations* (ICLR 2024). [arXiv:2310.03714](https://arxiv.org/abs/2310.03714).
 - [17] Xia, C. S., Deng, Y., Dunn, S., & Zhang, L. (2025). Demystifying LLM-Based Software Engineering Agents. *Proceedings of the ACM on Software Engineering* 2(FSE), 801-824. <https://doi.org/10.1145/3715754>
 - [18] Smith, E. K., Barr, E. T., Le Goues, C., & Brun, Y. (2015). Is the Cure Worse than the Disease? Overfitting in Automated Program Repair. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (ESEC/FSE 2015), 532-543. <https://doi.org/10.1145/2786805.2786825>
 - [19] Aleithan, R., Xue, H., Mohajer, M. M., Nnorom, E., Uddin, G., & Wang, S. (2024). SWE-Bench+: Enhanced Coding Benchmark for LLMs. [arXiv:2410.06992](https://arxiv.org/abs/2410.06992).
 - [20] Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2024). Large Language Models Cannot Self-Correct Reasoning Yet. *International Conference on Learning Representations* (ICLR 2024). [arXiv:2310.01798](https://arxiv.org/abs/2310.01798).
 - [21] Zhang, Y., Khalifa, M., Logeswaran, L., Kim, J., Lee, M., Lee, H., & Wang, L. (2024). Small Language Models Need Strong Verifiers to Self-Correct Reasoning. *Findings of the Association for Computational Linguistics: ACL 2024*, 15637-15653. <https://doi.org/10.18653/v1/2024.findings-acl.924>
 - [22] Just, R., Jalali, D., Inozemtseva, L., Ernst, M. D., Holmes, R., & Fraser, G. (2014). Are Mutants a Valid Substitute for Real Faults in Software Testing? *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (FSE 2014), 654-665. <https://doi.org/10.1145/2635868.2635929>
 - [23] Yang, J., Lieret, K., Jimenez, C. E., Wettig, A., Khandpur, K., Zhang, Y., Hui, B., Press, O., Schmidt, L., & Yang, D. (2025). SWE-smith: Scaling Data for Software Engineering Agents. *NeurIPS 2025 Datasets and Benchmarks Track*. [arXiv:2504.21798](https://arxiv.org/abs/2504.21798).