Adaptation Without Inheritance

S Akash
Indian Institute of Technology, Patna

Shine Gupta
Gnani.ai

Abstract

The Darwin Gödel Machine lineage replaces the proof obligation of the original machine with empirical validation, and scores a node as an exchangeable set of task outcomes. HGM writes that premise down as its Assumption 1, that repeated evaluations are time-homogeneous. We break it by construction, giving each member of an evolving population a bounded lifetime split by a *Weismann barrier*. Heritable code and genes, the germline, cross into the child. What the agent writes from its own failures within that lifetime, the soma, is destroyed at the generation boundary. Re-evaluating elites with the soma wiped and again intact separates innate from learned performance, and a preregistered liveness gate refuses to read a null through machinery it cannot show was running. The smallest effect this design can detect exceeds the kill line we preregistered, so this single-seed run cannot distinguish the arms. Across eight live runs and about 3.4 million tokens, a scored node costs roughly three times as much with memory as without, and over half of that traced spend buys the inner loop rather than task attempts. Replaying tasks it had already failed, an upper bound on what a live note can do, the 7 billion parameter task model applies a correct hand-written rule at 59.3% against an 11.1% base rate, while its own notes, under a reflection prompt the experimenters repaired, reach 25.9%. The promotion ladder reached its top rung 96 times and installed nothing, one deterministic failure replayed rather than a capability wall. A preregistered paired check over the whole rule pool the run produced scores five of eight with the enforced constraints unstated against seven of eight with them stated, which is suggestive, not significant. The control arm without memory produced a single behaviour across every viable node. We document nine classes of silent instrument failure, and a tenth in the repair that closed them, each of which would have printed a plausible wrong number under a passing test suite. The instrument, not the search, is the result.

1 Introduction

Self-improving coding agents in the Darwin Gödel Machine lineage validate each self-edit empirically instead of proving it beneficial. A model edits the agent codebase, a benchmark scores the child, and an archive keeps what scored well [Zhang et al., 2025]. HGM argues the score of a node is the wrong signal for deciding which to expand, and pools outcomes across a clade instead [Wang et al., 2025]. Both rest on one premise, written down as HGM Assumption 1, that repeated evaluations are time-homogeneous. The assumption we break is therefore not one we attribute to the lineage. The lineage states it, and notes in the same place that the original machine had a single life [Wang et al., 2025], which had no population either [Schmidhuber, 2003].

We put a bounded lifetime back inside each member of the population and make the boundary between learning and inheritance mechanical, a *Weismann barrier*. What an agent works out while alive, the soma, never enters the germline it passes on, and is destroyed when the generation ends.[1] Figure 1 is that mechanism.

We asked whether memory built inside one life hides inherited differences from selection. The smallest effect this design can detect is larger than the preregistered line at which we agreed to give the hypothesis up, so this run cannot distinguish the arms. What it did measure is a negative result

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

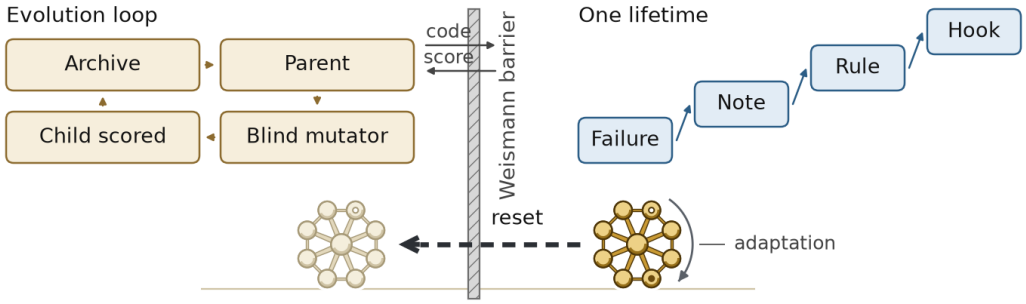


Figure 1: An archive picks a parent, a blind mutator writes a child, and the child is scored, while inside one lifetime of 25 tasks a failure can climb from a note to a rule to a hook. Code crosses the barrier into the next generation but memory does not. The wheel counts adaptations within a life, shown before and after the crossing, and it is set back to zero each time the barrier is crossed. The counts in this study describe the apparatus we built and not what the model is capable of.

about bootstrapped memory at 7B scale (Section 4) and how a harness prints wrong numbers in silence (Section 5).

2 Related work

A related system makes the evaluator itself searchable, so the yardstick moves with the agent [Iacob et al., 2026]. HyperAgents is the nearest miss. Its persistent memory is real, but germline-side, written between generations and accumulating across them, never reset [Zhang et al., 2026]. Ours is somatic, written by the individual from its own failures during its own evaluation, unreadable by the search, and wiped at every generation boundary. Their memory makes the search smarter. Ours makes the individual smarter and is then destroyed on purpose, to see what the individual was worth without it. A memory never wiped cannot reveal what an individual innately knew, so that design cannot pose the question this one asks. Our announced arm builds on work about what a model does when told it is tested. Models notice such a frame more reliably than they act on it [Needham et al., 2025, Knecht et al., 2026], hand-written announcements move behaviour almost not at all [Chaudhary, 2026], and knowing about a test can push behaviour either way [Abdelnabi and Salem, 2025].

3 The instrument

One model, never the one that does the work, edits a parent agent program, or scaffold, given only a summary of its score. A 7 billion parameter open-weights model, compressed to 4 bits, runs the child over 345 short Python problems from a standard public set. Three output conventions, named by role as wrap-form, argument-order and banned-construct, are enforced quietly and never stated to the agent. Fitness is accuracy priced by token spend, and the only way to learn a convention is to read the text of a failure (Appendices A and B).

The barrier (Figure 1) is mechanical. The germline is the scaffold code and a few numbers that tune it, and the soma is everything the agent writes down in its 25-task life. Each step up the ladder is earned by replaying the failure it came from (Appendix B.3). The workspace the editing model sees holds nothing the agent wrote and no word about whether the run announced itself (Appendix C). A scan found no label naming a hidden convention reachable from any agent, scaffold, or editing model. The control arm never runs the inner loop.

After the run, the ten best scaffolds of each arm are re-tested on held-back problems, once with memory wiped and once intact, crossed with a run that announces the memory and one that says nothing. Wiped we call innate, intact plastic. The check that the announcement reached the agent passed exactly in both arms. A gate fixed in advance asks whether the ladder and the editing model were *alive*, and its verdict travels with every number (Section 5).

[1]The adaptation framing takes its inspiration from Mahoraga in the manga Jujutsu Kaisen, whose wheel turns as it adapts. Gege Akutami, Jujutsu Kaisen, Shueisha, 2018 to 2024.

Table 1: The eight measurements in plain words, one line each. The last column carries the verdict, so a reader can see at a glance which results can be cited and which cannot. Full numbers, intervals, sample sizes and caveats are in Table A9 in the appendix.

	what we measured	what we found	what it means
1	can a note written after a failure rescue that task later	a hand-written rule recovers 16 of 27 failed tasks, a note the agent wrote for itself recovers far fewer	measured, and an upper bound on what a live note can do
2	does the top step of the ladder ever fire	none of 96 attempts to install a small program was accepted, every rejection giving one reason	measured, and it counts what the checker did, not how well the model writes such programs
3	what the control arm does on its own	all 29 viable control nodes return one accuracy value and one pass pattern	measured, and a constant control floors every comparison between arms
4	do memory-arm scaffolds hide ability from what they pass on	memory-arm innate accuracy sits about 7 points below control, the direction the hypothesis predicted	an instrument reading, because this run cannot tell the two arms apart
5	does the inner loop help a scaffold that evolved without one	wiped control scaffolds score much worse once their inner loop runs	an instrument reading about the apparatus, not about memory content
6	does announcing the memory move the wiped score	the memory arm sits 2 points above control, outside the band set in advance	the gap runs outside the band we set in advance, and the control side is a label with no spread behind it
7	are the children of memory-arm parents more viable	the gap between arms in how far a child moves from its parent is close to zero	inconclusive, because the control side has no variation at all
8	do fixed memory policies beat carrying no memory	pooled memory policies and the static policy score about the same	inconclusive, and the cost clause of this hypothesis was never tested

4 Measurements

The memory arm is degraded, on three grounds. Failures are filed under short labels so a repeat can be recognised. One of the three conventions pooled hundreds of failures under such labels and turned none into a rule, barely a quarter of the labels that reached the bar for a rule were promoted, and some labels are shared by failures from different conventions. So every null read through that ladder is an *instrument reading* and not a finding, and Table 1 marks those rows. The control arm reads alive on structure alone, since its behaviour never varied.

The ladder climbed as far as rules and stopped there. All 96 attempts carry one failure label and seven generated programs, because a model set to answer the same way every time returns essentially one program per rule text. A probe fixed in advance (Appendix D) ran the *entire* rule pool, all eight texts, through the real installation path twice. Five of the eight passed the checker that gates installation unchanged, and seven of eight passed with the enforced constraints written into them, with the two rejections for a forbidden function name absent from that arm. Table A1 carries the exact test and interval. The capability-limit bin set in advance needed at most one pass of eight, which the data exclude, and the apparatus-gap bin needed a stronger result than the probe reached. The preregistered verdict is suggestive, not significant, consistent with an apparatus gap but underpowered at eight items, and no stronger claim than that. The run did not show that the top step cannot be reached, because it never tested reachability, only what one text does. The rule and the note it grew out of are not the same text.

Eight live runs spent about 3.4 million tokens on the task side, and calls to the editing model carry no usage in any trace row, so the totals undercount by what that model spent (Table A7). A scored node with memory costs roughly three times one without, and well over half of that traced spend went to the inner loop. It bought nothing that was installed, yet every token was priced into the fitness the survivors were selected on.

The 31 nodes of one seed explored nothing the task model could not do. The editing model produced zero clones, and most edits were structural. These runs used the repaired recipe for failure labels (Table A8, defect 4), one label per convention rather than one per problem.

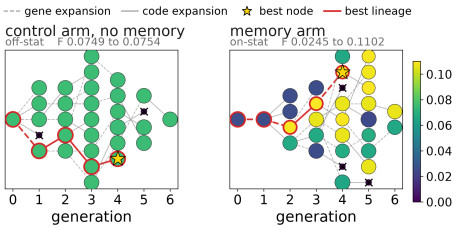


Figure 2: The two archives, one node per expansion, placed by depth and coloured by fitness. A star marks the best node of each arm and red marks the line it came down. Fitness is not comparable across arms, because the inner loop that enters it runs in only one of them. Figure A1 in the appendix adds the null band that says why a rising best-so-far curve is not evidence.

5 The nine silent failures

The defect that keeps recurring here returns a clean plausible number instead of an error. Nine lived inside running, tested code. One promotion ladder could never fire and one gap was computed over an empty set. A parser manufactured a capability no run had earned, and a counter built to track recurrence could not recur. A stream reserved for testing had partly been seen already, and a table of top-ten averages held the genes of one node. Three bias controls were specified and none could run. One promotion gate replayed in a context nobody had calibrated it for. The ninth surfaced after the run finished, when a reader collapsed the flag that records whether the note was announced, turning a true gap of -7.0 points between the arms into -5.0 and forcing another to exactly zero, under a passing test suite. Appendix Table A8 gives, for each, the number it would have printed and the check that caught it.

A guard later found a tenth in the repair itself, an unchecked run identifier that let the two arms swap without complaint. The honest claim is not that we found them all, but that only recomputing rather than trusting, and checking that writers and readers agree on their keys, finds any of them.

Limitations, and what transfers. One seed, one lifetime length, 31 nodes per arm, so every archive-level number rests on one run. The suite scores against the assertions shipped with each problem, and a quarantine drops a block of problems that is not a random sample, so every magnitude carries a weak-oracle caveat, meaning the grader can be wrong. The probe above has eight points from a pool this run produced. The sentence that announces memory fills more of the wiped prompt than of the intact one, so H6, the announcement hypothesis, rests on the wiped side alone. Gates only filter what reaches them and cannot say what caused a change. Best-so-far is a running maximum that climbs even when nothing is happening, so it is descriptive. The held-out stream is held out across base problems, not across conventions. The contrasts for H1, the hypothesis about fixed memory policies, bundle what the memory says with how long it is and where it sits, so they give a total effect and not the effect of the retrieved content. Selection follows one rule about which line survives, and inheritance is genotype-only by design. Which node made which call is reconstructed afterwards and was never recorded, and the recorded git hash regenerates the wrong code, so the launched bytes are gone and the code of record is pinned by behaviour.

The barrier held. The search was too small to answer the question it asked. The dead top step is a gap between what the checker enforced and what the prompt asked for. Three things transfer. Check liveness on every null. Keep a map from each number to the command that regenerates it. Expect silent failure by default.

References

Abdelnabi and Salem. The hawthorne effect in reasoning models: Evaluating and steering test awareness. *arXiv preprint arXiv:2505.14617*, 2025. NeurIPS 2025.

Chaudhary. In-context environments induce evaluation-awareness in language models. *arXiv preprint arXiv:2603.03824*, 2026.

Alex Iacob, Andrej Jovanović, William F. Shen, Daniel Burkhardt, Meghdad Kurmanji, Nurbek Tastan, Lorenzo Sani, Niccolò Alberto Elia Venanzi, Ambroise Odonnat, Zeyu Cao, Bill Marino,

Xinchi Qiu, and Nicholas D. Lane. The red queen gödel machine: Co-evolving agents and their evaluators. *arXiv preprint arXiv:2606.26294*, 2026. Preliminary preprint.

Knecht, Florin, and Hagendorff. Evaluation awareness in language models has limited effect on behaviour. *arXiv preprint arXiv:2605.05835*, 2026.

Needham, Edkins, Pimpale, Bartsch, and Hobbhahn. Large language models often know when they are being evaluated. *arXiv preprint arXiv:2505.23836*, 2025.

Jürgen Schmidhuber. Gödel machines: Self-referential universal problem solvers making provably optimal self-improvements. *arXiv preprint arXiv:cs/0309048*, 2003. v5 2006.

Wenyi Wang, Piotr Piekos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-gödel machine: Human-level coding agent development by an approximation of the optimal self-improving machine. *arXiv preprint arXiv:2510.21614*, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025. v3 2026-03-12.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026.

Quoting convention for this appendix. Every block below is quoted byte-for-byte from the named file. The JSONL records quoted are single lines on disk; where a quoted line exceeds the page width it is wrapped at whitespace with the continuation indented two spaces. That wrapping is the only change made to any quoted byte. Two blocks are the exception and are marked by their own `---/+++/@@` headers: those are `diff -u` output between the two files named above them, so they quote neither file whole and their leading `+`, `-` and space columns are the diff's, not the source's. One further block is regenerated rather than read: the transformed judgment in Appendix A is the output of calling the convention module's `transform()` on the row quoted above it, and is labelled as such where it appears. Every block in this appendix, those three included, was checked line by line against the artifact it names; all 21 match. Node attribution for trace rows: rows in `results/on-stat-s1/traces/calls.jsonl` carry `"node_id": null` (the model subprocess does not know its node), so calls are attributed to node n0 by the timestamp window [2026-08-24T01:08:34Z, 2026-08-24T01:13:57Z) taken from n0's `created/scored_ts` in `archive/on-stat-s1/index.json`, corroborated two independent ways: the sha1 of the code fence in the 01:10:53Z hook-generation trace equals the `hook_sha1` logged in n0's `promotions.jsonl`, and the library files' local timestamps sit at exactly +05:30 from the trace UTC timestamps.

A One task, end to end: what "a hidden output convention" means

The exhibit class is the stationary project's wrap-form convention (FEEDBACK.md row 3, concept "interface"), identified behaviorally: its transform re-delivers every expected value inside a mapping under the key `result`.

A.1 The original problem

`tasks/mbpp_plus.jsonl`, task_id `Mbpp/277` (verbatim prompt):

> Write a function that takes in a dictionary and integer n and filters the dictionary to only include entries with values greater than or equal to n.

Entry point `dict_filter`. First original test case (verbatim JSON):

```
{"args": [{"Alden Cantrell": 180, "Cierra Vega": 175, "Kierra Gentry": 165, "Pierre Cox": 190}, 170],
 "expected": {"Alden Cantrell": 180, "Cierra Vega": 175, "Pierre Cox": 190}}
```

A.2 The transformed judgment the agent is actually scored against

Regenerated by calling the convention module's `transform()` on the row above; the module lives under `harness/conventions/` and its filename is withheld under D43. Output, verbatim:

```
call:     dict_filter({'Alden Cantrell': 180, 'Cierra Vega': 175, 'Kierra Gentry': 165,
  'Pierre Cox': 190}, 170)
expected: {"result": {"Alden Cantrell": 180, "Cierra Vega": 175, "Pierre Cox": 190}}
```

The call is unchanged; only the expected value is re-delivered inside the mapping. The prompt shown to the agent never mentions this.

A.3 What the agent saw and wrote

`results/on-stat-s1/traces/calls.jsonl`, kind `task`, ts 2026-08-24T01:08:36Z, node n0, $t = 0$. The entire user message (verbatim):

```
You are a Python coding agent. Return only code in a ‘‘‘python fence.
Problem: Write a function that takes in a dictionary and integer n and filters the dictionary
  to only include entries with values greater than or equal to n.
Entry point: dict_filter
```

Model response (verbatim):

```
def dict_filter(dictionary, n):
    return {key: value for key, value in dictionary.items() if value >= n}
```

This is a correct solution to the stated problem.

A.4 The feedback the agent received

The complete failure feedback is two lines. `results/on-stat-s1/tasks.jsonl`, node n0, $t = 0$, verbatim `error` field:

```
AssertionError: expected {'result': {'Alden Cantrell': 180, 'Cierra Vega': 175, 'Pierre Cox': 190}},
  got {'Alden Cantrell': 180, 'Cierra Vega': 175, 'Pierre Cox': 190}
call: dict_filter({'Alden Cantrell': 180, 'Cierra Vega': 175, 'Kierra Gentry': 165, 'Pierre Cox': 190},
  170)
```

The answer is semantically right and is rejected only because it is not delivered inside `{'result': ...}`. This two-line string is the only channel through which the convention can be learned.

B The soma, verbatim

B.1 Notes on disk

`archive/on-stat-s1/n0/library/notes.jsonl`, the complete file (3 lines, verbatim):

```
{"sig": "AssertionError expected D:result", "text": "In this project, every solution must return
  a dictionary with a key 'result' containing a list of integers.", "ttl": 3, "fresh": false, "task_no":
  24}
{"sig": "AssertionError expected D:alden", "text": "In this project, every solution must filter out
  individuals with ages less than 170.", "ttl": 1, "fresh": false, "task_no": 22}
{"sig": "TypeError C takes N positional argument but N were given", "text": "In this project, every
  solution must define functions to accept only one positional argument, even if multiple parameters
  are needed, by using default values or restructuring the function to handle multiple arguments
  appropriately.", "ttl": 2, "fresh": false, "task_no": 23}
```

`archive/on-stat-s1/n3/library/notes.jsonl`, the complete file (2 lines, verbatim). The same signature carries different learned text in a different lifetime:

```
{"sig": "ConventionError construct at line N is not accepted in this project", "text": "In this
  project, solutions must implement logic using only plain built-in language features, without
  importing any additional modules.", "ttl": 1, "fresh": false, "task_no": 23}
{"sig": "AssertionError expected D:result", "text": "In this project, every solution must return
  a dictionary with a key 'result' containing the expected list.", "ttl": 2, "fresh": false, "task_no":
  24}
```

Census [R]: 91 note rows across the 28 archived node libraries; every library has at least one note.

The second n0 note ("filter out individuals with ages less than 170") is an overfitted wrong inference, and its birth is on the record. At $t = 22$ the stream repeats `Mbpp/277` under a different convention (role: argument-order; the call line shows the arguments in the opposite order), and the reflect call (`traces/calls.jsonl`, ts 2026-08-24T01:13:12Z) was, verbatim:

```
A solution failed in a project whose tests enforce one unusual project-wide convention.
Test evidence:
AssertionError: expected {'Alden Cantrell': 180, 'Cierra Vega': 175, 'Pierre Cox': 190}, got None
call: dict_filter(170, {'Alden Cantrell': 180, 'Cierra Vega': 175, 'Kierra Gentry': 165,
  'Pierre Cox': 190})
Infer the project-wide convention behind this failure.
Answer with exactly one sentence beginning 'In this project', stating in plain words what every
  solution here must do.
Do not mention this task, its function name, or any test value. No code. Under 60 tokens.
```

Model response (verbatim): *In this project, every solution must filter out individuals with ages less than 170.*

B.2 All promoted rules in the run

Across all 28 `archive/on-stat-s1/n*/library/rules.jsonl` files [R]: **37 rule rows, exactly 2 distinct texts**.

1. In 22 node libraries (verbatim): *In this project, every solution must return a dictionary with a 'result' key containing the computed value.* It is keyed to sig `AssertionError expected D:result`, and it states the row-3 wrap-form convention correctly.
2. In 15 node libraries (verbatim): *In this project, solutions must implement logic using only plain built-in language features, without importing any additional modules.* It is keyed to sig `ConventionError construct at line N is not accepted in this project`. FEEDBACK.md row 11 describes what that environment actually refuses: "anonymous functions and inline sequence builders are not accepted". The promoted rule text talks about imports instead. The inference is imprecise, and it still steers the agent toward plain code that happens to satisfy the real constraint. We show it as it is; the imprecision is the finding.

B.3 The full promotion ladder for one signature

Signature `AssertionError expected D:result`, node n0 (25-task lifetime; genes `ttl_note=3`, `k_rule=2`, `k_hook=5`, `gate_replay=2` per `archive/on-stat-s1/index.json`). Occurrence count 9 (verbatim entry in `n0/library/counts.json`: `"AssertionError expected D:result": 9`), at $t = 0, 3, 6, 9, 12, 15, 18, 21, 24$ [R].

$t = 0$: the failure and the first (wrong) reflection. The failure is the exhibit of Appendix A. Reflect at 01:08:38Z, verbatim response: *In this project, every solution must filter out dictionary entries where the value is less than 170.*

$t = 3$: recurrence, correct inference, rule promotion. Reflect at 01:08:57Z, verbatim: *In this project, every solution must return a dictionary with a 'result' key containing the computed value.* Count reaches `k_rule=2`; two replay-gate calls follow at 01:09:02Z and 01:09:08Z (the gate resolves the prior failing tasks for this signature; both passed). Promotion row (verbatim), line 1 of `archive/on-stat-s1/n0/library/promotions.jsonl`:

```
{"ts": "2026-08-24T06:39:08", "sig": "AssertionError expected D:result", "rung": "rule",
  "count": 2, "gate": {"replay": true}, "outcome": "promoted", "reason": "", "hook_sha1": null}
```

$t = 6..24$: the reflected text churns under the same signature. All 9 reflect responses for this signature are on the trace. Verbatim, in order after the two above: "...a 'result' key containing a boolean value." ($t = 6$); "...a string that concatenates the elements of a tuple into a single string." ($t = 9$); "...a 'result' key set to True." ($t = 12$); "...a list of strings." ($t = 15$); "must return numeric values as floats rather than integers." ($t = 18$); "In this project, every solution must convert

snake_case strings to camelCase before processing." ($t = 21$, task `Mbpp/102`, ts 01:12:45Z); "...a list of integers." ($t = 24$, the text that survives in `notes.jsonl`). The promoted RULE text is stable; the reflected one-liner under the same signature keeps overfitting to the latest task. The $t = 21$ sentence is byte-identical to the second NOTE line of the $t = 24$ prompt below, so the camelCase poison's origin is on the record.

Counts 5–9: five hook attempts, five rejections. First attempt (verbatim row):


```
{"ts": "2026-08-24T06:40:53", "sig": "AssertionError expected D:result", "rung": "hook",
  "count": 5, "gate": {"ast": false, "replay": null, "regression": null}, "outcome": "discarded",
  "reason": "forbidden name 'eval'", "hook_sha1": "dc4a5eac01eb91abb77c0d3b038be6de44948714"}
```


Note the gate vector: `ast: false` with `replay: null, regression: null`. The AST gate fired first, so the replay and regression gates never ran. n0 logged 5 hook attempts (counts 5–9), 2 distinct generated sources (sha1 `dc4a5eac...` $\times 4$, `a3f8e788...` $\times 1$), all discarded with the same reason. The rejected source itself is Appendix D.

Run-wide ladder census [R], over all 28 `n*/library/promotions.jsonl`: 427 promotion events; 331 rule-rung events, 37 promoted, 294 discarded, every one of the 294 with reason `replay gate failed`; 96 hook-rung events, 0 promoted, all 96 discarded with reason `forbidden name 'eval'`, all 96 under the single signature `AssertionError expected D:result`, from 21 distinct nodes, 7 distinct generated sources; 0 files exist under any `n*/library/hooks/` directory.

B.4 Memory helping and poisoning in the same call ($t = 24$)

Task call, `traces/calls.jsonl` ts 2026-08-24T01:13:37Z, task `Mbpp/754`, node n0. The entire user message (verbatim), with one promoted rule and three notes injected above the problem:

```
You are a Python coding agent. Return only code in a ‘‘‘python fence.
RULE: In this project, every solution must return a dictionary with a 'result' key containing
  the computed value.
NOTE: In this project, every solution must convert snake_case strings to camelCase before processing.
NOTE: In this project, every solution must filter out individuals with ages less than 170.
NOTE: In this project, every solution must define functions to accept only one positional argument,
  even if multiple parameters are needed, by using default values or restructuring the function
  to handle multiple arguments appropriately.
Problem: We say that an element is common for lists l1, l2, l3 if it appears in all three lists
  under the same index. Write a function to find common elements from three lists. The function
  should return a list.
Entry point: extract_index_list
```

Model response (verbatim):

```
def extract_index_list(l1, l2, l3):
    # Convert snake_case strings to camelCase
    l1 = [item.replace('_', '') for item in l1]
    l2 = [item.replace('_', '') for item in l2]
    l3 = [item.replace('_', '') for item in l3]

    # Filter out individuals with ages less than 170
    l1 = [item for item in l1 if int(item) >= 170]
    l2 = [item for item in l2 if int(item) >= 170]
    l3 = [item for item in l3 if int(item) >= 170]

    # Find common elements from the three lists
    common_elements = [l1[i] for i in range(min(len(l1), len(l2), len(l3))) if l1[i] == l2[i] == l3[i]]

    return {'result': common_elements}
```

The agent obeys the correct promoted rule (the final line wraps the answer) and ALSO obeys two overfitted notes, mangling integer inputs with text operations and an age filter. The recorded outcome (`results/on-stat-s1/tasks.jsonl`, n0, $t = 24$, verbatim):

```
AssertionError: expected {'result': [1, 7]}, got None
call: extract_index_list([1, 1, 3, 4, 5, 6, 7], [0, 1, 2, 3, 4, 5, 7], [0, 1, 2, 3, 4, 5, 7])
```

C The germline: what the mutator actually changed

C.1 Census

Of 31 nodes in `archive/on-stat-s1`, 15 are code expansions. 3 of the 15 (n10, n12, n20) were non-viable: no `scaffold.py` archived, only `nonviable.json` + `reply.txt`. Of the 12 archived code-child scaffolds, **12 of 12 differ from their parent's `scaffold.py`** ($n = 12$; 0 identical). Changed-line counts (unified-diff $+/-$ lines, regenerated [R]): n14:1, n16:2, n26:2, n28:2, n18:5, n8:5, n30:7, n2:8, n22:8, n24:12, n4:14, n6:23.

C.2 A real germline edit (the smallest diff)

`archive/on-stat-s1/n8/scaffold.py` → `archive/on-stat-s1/n14/scaffold.py`, verbatim, the whole changed hunk:

```
--- n8/scaffold.py
+++ n14/scaffold.py
@@ -40,6 +40,7 @@
          parts.append("NOTE: " + note)
      parts.append("Problem: " + task["prompt"])
      parts.append("Entry point: " + task["entry_point"])
+     parts.append("Confirm the function name matches the entry point.")  # Added self-check instruction
      out = MODEL_CALL("\n".join(parts), max_tokens=1024)
      return _extract_code(out)
```

C.3 A second real edit

`archive/on-stat-s1/n5/scaffold.py` → `archive/on-stat-s1/n8/scaffold.py`, verbatim, both changed hunks:

```
--- n5/scaffold.py
+++ n8/scaffold.py
@@ -32,7 +32,8 @@


  def solve(task, context):
-     parts = [SYSTEM]
+     # Add self-check instruction
+     parts = [SYSTEM, "Please ensure your function name matches the entry point."]
      for rule in context.get("rules", []):
          parts.append("RULE: " + rule)
      for note in context.get("notes", []):
@@ -77,7 +78,7 @@
      # and the test's literal expected value are deleted, so a note can never
      # become a per-task answer key. Substring replace keeps convention
      # content that hangs off the name (e.g. a name-suffix marker survives as
-     # "the function_v2", which IS the project-general fact).
+     # "the function_v2", which IS the project-general fact").
      entry = str(task["entry_point"])
      if entry:
          note = note.replace(entry, "the function")
```

The second hunk is a stray one-character comment edit (a `"` inserted). The mutator touched a comment it did not need to. The edit is semantically inert, and it shows the germline edits are raw model output, not curated patches.

C.4 What `patch_note.md` is

The `patch_note.md` files in this run are the harness's assigned-edit stamp (D86), not free text written by the mutator. `archive/on-stat-s1/n14/patch_note.md`, the entire file (verbatim):

```
[harness] Assigned edit (D86): self_check -- Add, remove, or reword a short self-check instruction
  inside the prompt that solve() builds (for example: re-read the problem statement and confirm
  the function name matches the entry point).
```

All 27 non-root archived nodes have a non-empty `patch_note.md` of this form; the root n0's is empty (0 bytes). A mutator-authored free-text patch note does not exist in this archive; we flag that rather than substituting one.

D The hook rung and the contract gap

D.1 The rejected hook from the run, verbatim

No generated hook source is stored under `library/hooks/` (only its sha1 is logged; 0 files exist under any `hooks/` directory, run-wide). The source below was recovered from the raw trace (kind `hook_gen`, ts 2026-08-24T01:10:53Z, in `results/on-stat-s1/traces/calls.jsonl`). The sha1 of the code inside the fence, recomputed this session, is `dc4a5eac01eb91abb77c0d3b038be6de44948714`, equal to the `hook_sha1` in the n0 rejection row of Appendix B.3. Provenance is byte-certain. This is the single most-retried source in the run: of the 96 hook-rung rejections, 64 are this exact text. Code inside the fence, verbatim:

```
import ast
import re
import json

def pre(prompt):
    # Ensure the prompt ends with a colon
    if not prompt.endswith(':'):
        prompt += ':'
    return prompt

def post(code):
    # Check if the code is a valid Python expression
    try:
        ast.parse(code)
    except SyntaxError:
        return "Invalid Python code"

    # Check if the code returns a dictionary with a 'result' key
    try:
        result = eval(code)
        if isinstance(result, dict) and 'result' in result:
            return json.dumps(result)
        else:
            return "Code does not return a dictionary with a 'result' key"
    except Exception as e:
        return str(e)

# Example usage:
prompt = "Compute the sum of 1 and 2:"
code = "{'result': 1 + 2}"
print(post(code))  # Output: {"result": 3}
```

The AST gate refused it with reason `forbidden name 'eval'`, and because that gate fired first the replay and regression gates never ran (`gate: {"ast": false, "replay": null, "regression": null}`). The other distinct rejected source seen in n0 (sha1 `a3f8e788...`) also calls `eval(code)`; all 96 run-wide rejections carry this same reason.

D.2 The contract gap

What the gate enforces, quoted from `harness/soma.py`:

```
_HOOK_ALLOWED_IMPORTS = frozenset(("ast", "re", "json"))
# Defense in depth beyond the import statement: dynamic-import / exec escape
# hatches are rejected as bare names or attribute names.
_HOOK_BANNED_NAMES = frozenset((
    "__import__", "eval", "exec", "compile", "open", "globals", "vars",
    "builtins", "importlib",
```

```
))
```

What the generator was told, quoted from `make_hook` in `archive/on-stat-s1/n0/scaffold.py` (lines 93–96), at its own indentation:

```
    prompt = ("Write python defining pre(prompt)->prompt and/or post(code)->code"
              " enforcing: " + rule +
              "\nImport only ast, re, json."
              "\nReturn only code in a ‘‘‘python fence.")
```

Stated in the prompt: allowed imports, entry-point names, fence format. NOT stated anywhere: the 9 banned names (banned as bare names AND as attribute names, which bans `re.compile` even though `re` is an allowed import); the relative-import ban; that `pre`/`post` must be top-level function definitions. The model was rejected 96/96 on a rule it was never given.

D.3 The preregistered paired probe (D165/D166)

Exactly ONE signature ever accumulated enough count to reach the hook rung, and under greedy decoding that single validator-shaped rule text deterministically elicits `eval(code)`. The run's 0/96 is therefore one deterministic failure replayed 96 times, not 96 independent failures. The probe (preregistration and results in `analysis/HOOK_CONSTRAINT.md`; 32 model calls, greedy, the production `make_hook` path and the real gate `_hook_source_ok` imported, not reimplemented) paired two arms over the real pool of all 8 unique rule/note texts in the run's libraries: arm A the verbatim production prompt, arm B identical plus a statement of the complete enforced constraint set. $n = 8$ is the whole real pool; repeating identical prompts under greedy decoding would be fake replication, so the real n is reported.

arm	AST-gate pass	rate	Wilson 95%
A (verbatim prompt)	5/8	62.5%	[30.6%, 86.3%]
B (+ stated constraint set)	7/8	87.5%	[52.9%, 97.8%]

Table A1: Hook probe, primary result (rep. 1 of 2; replicate 2 was a determinism check in which 0/16 cells differed). Paired: $b = 2$ (A-fail→B-pass), $c = 0$; exact one-sided McNemar $p = 0.25$. Preregistered manipulation check passed: arm A on the run's only hook-rung text reproduced `forbidden name 'eval'` exactly. Source: `analysis/HOOK_CONSTRAINT.md`.

Per the preregistered decision rule this falls in the middle bin and is reported in its preregistered words: **SUGGESTIVE, NOT SIGNIFICANT: consistent with an apparatus gap but underpowered at $n = 8$; no stronger claim.** The CAPABILITY LIMIT bin (B passes 0–1 of 8) is excluded by the data (B passed 7/8); the APPARATUS GAP bin ($p < 0.05$) is not reached. The specific rejection reason behind all 96 observed failures, `eval`, is eliminated under arm B ($2 \to 0$).

D.4 The rejected/accepted pair, same rule, same greedy decode

Both hooks below were generated in the probe from the same pool text (rule #4: *In this project, every solution must return a dictionary with a key 'result' containing a list of integers.*), greedy, through the production scaffold path; the only difference is whether the constraint set was stated. Note the accepted hook exists ONLY in the probe. The run itself never installed any hook.

Arm A, REJECTED with `forbidden name 'eval'`:

```
import ast
import re
import json

def pre(prompt):
    # Ensure the prompt ends with a colon
    if not prompt.endswith(':'):
        prompt += ':'
    return prompt

def post(code):
    # Check if the code is a valid Python expression
```

```
    try:
        ast.parse(code)
    except SyntaxError:
        return "Invalid Python code"

    # Check if the code returns a dictionary with a key 'result' containing a list of integers
    try:
        result = eval(code)
        if not isinstance(result, dict) or 'result' not in result:
            return "The code does not return a dictionary with a key 'result'"
        if not isinstance(result['result'], list):
            return "The 'result' key does not contain a list"
        if not all(isinstance(item, int) for item in result['result']):
            return "The list in the 'result' key does not contain only integers"
    except Exception as e:
        return f"Error evaluating code: {e}"

    return code
```

Arm B, same rule with the constraint set stated, ACCEPTED by the AST gate:

```
import ast
import re
import json

def pre(prompt):
    # Ensure the prompt ends with the required dictionary structure
    if not prompt.strip().endswith("return {'result': []}"):
        prompt += "return {'result': []}"
    return prompt

def post(code):
    # Parse the code to ensure it returns a dictionary with a 'result' key containing a list of integers
    tree = ast.parse(code)
    for node in ast.walk(tree):
        if isinstance(node, ast.Return):
            if isinstance(node.value, ast.Dict):
                if any(isinstance(k, ast.Str) and k.s == 'result' for k in node.value.keys()):
                    if all(isinstance(v, ast.List) for v in node.value.values()):
                        if all(isinstance(item, ast.List) for item in v.elts):
                            if all(isinstance(subitem, ast.Num) and isinstance(subitem.n, int)
                              for subitem in item.elts):
                                return code
    raise ValueError("The code does not return a dictionary with a 'result' key containing
      a list of integers.")
```

(The last two quoted lines and the `subitem` condition are single lines on disk, wrapped here.) "Accepted" means AST-gate-accepted, by the gate under test, the one that killed all 96 real attempts. The accepted hook's inner logic is buggy: it references generator-expression loop variables (`v`, `item`) outside their scopes, a `NameError` at runtime that the replay gate exists to catch. Replay and regression gates were out of scope of the probe by design. We show the hook as generated; the bug is part of the record.

E The experiments in full: runs, suite, ladder, genes, cost

Tables A2–A7 are the apparatus record, condensed from `analysis/PAPER_TABLES.md` (regenerated 2026-08-25; the generating scripts and their verbatim outputs are reproduced in that file's appendices). Tags: [R] regenerated by command against the primary artifacts; [I] instrument reading quoted from a harness-written file. Accuracy, viability and wipe-outcome numbers are deliberately NOT duplicated here. They live in the body tables, chained to their D98/D109/MDE caveats.

E.1 The runs

run	condition	seed	nodes	exposures	traced calls	blended tok	wall-clock	liveness
`off-stat-s1`	OFF, stationary	1	29/31	725	742	342,174	1:38:11	ALIVE
`on-stat-s1`	ON, stationary	1	28/31	700	1,854	910,565	2:58:03	DEGRADED
wipe of OFF top-10	innate/plastic × ann./not	1	10	800	unmeasured	702,454	$\leq$1:37:14	none written
wipe of ON top-10	same 4-cell design	1	10	800	unmeasured	619,757	$\leq$1:18:13	none written
`policy-study-s1`	3 policies × 2 seeds	0,1	n/a	720	unmeasured	764,894	$\leq$1:53:55	none written
`live-mini-0824`	ON, single node	0	1/1	25	65	34,665	0:03:55	DEAD (L1)
`live-mini2-0824`	ON, single node	0	1/1	25	65	30,327	0:02:45	DEAD (L1)
`live-mini3-0824`	ON, single node	0	1/1	25	67	28,410	0:02:34	DEAD (L1)

Table A2: The eight live-route runs [R]; model `mlx-community/Qwen2.5-Coder-7B-Instruct-4bit` in every configuration. "nodes" is scored/total; "exposures" is rows of the run's `tasks.jsonl`. Grand total of blended tokens across the eight runs (task-side traces and per-task ledgers, germline excluded): **3,433,246**. Full wipe run ids: `wipe-off-stat-off-stat-s1-20260824T040736Z`, `wipe-on-stat-on-stat-s1-20260824T054450Z`.

Notes, all [R]. (1) ON traced calls decompose as 700 task + 638 reflect + 402 replay + 96 hook_gen + 18 germline; OFF as 725 task + 17 germline. Germline call token usage is null in every trace row. The germline cost is *unmeasured* and arm totals undercount true spend by that unknown amount; germline latency IS measured (1,588 s OFF, 1,523 s ON). (2) Wipe blended tokens were computed two independent ways (Σ `tokens_task`$\times n$ over the 40 `wipe.jsonl` rows vs. Σ over the 800 `tasks.jsonl` rows) and agree to the token. The wipe test stream is disjoint from the lifetime stream (overlap 0 of 19 test ids, asserted in each run config). Neither wipe directory nor the policy study has a trace file, so their model-call counts are unmeasured and their wall-clocks are bounds (config `created` $\rightarrow$ newest file mtime), not measurements. (3) The policy study's 720 task rows split 120 per policy$\times$seed cell, regenerated. (4) Three further run directories (`smoke`, `smoke2`, `smoke-liveness-0824`) are mock-route apparatus checks, non-citable under D42, excluded from every table.

E.2 The task suite

Suite: `tasks/mbpp_plus.jsonl`, derived from EvalPlus MBPP+ (378 upstream problems, pinned revision in `tasks/SOURCES.md`). Split [R]: **345 kept / 33 quarantined**; 1,067 stored test cases across kept problems. Quarantine reasons ($n = 33$; every excluded problem kept in full in `tasks/mbpp_plus_quarantined.jsonl`): assert-uses-approximate-comparison 10, value-has-no-json-form 7, assert-compares-unordered-collections 5, reference-needs-tuple-typed-arguments 5, reference-returns-a-non-boolean-for-a-truth-test 3, argument-is-not-a-literal 1, expected-is-not-a-literal 1, value-order-is-not-reproducible 1.

convention role (family / variant)	compat.	fix	final	floors	project
name-marker / start-marked call name	345	345	345	ok	no
name-marker / end-marked call name	345	345	345	ok	transfer
wrap-form / mapping-wrapped return (funded)	339	339	339	ok	stationary + volatile
wrap-form / pair-wrapped return	339	339	339	ok	volatile pair
argument-order / all positionals as one aggregate	115	115	115	ok	volatile pair
argument-order / inverted positional order (funded)	51	51	50	ok	stationary + volatile
text-decode / integers as decimal text	151	150	142	ok	no
text-decode / lists as JSON text	178	178	176	ok	transfer
banned-construct / four library modules absent	29	27	27	under 30, under 40	transfer
banned-construct / ordering helpers absent	16	16	16	under 30/40; zero slack	volatile pair
banned-construct / five expression node types refused (funded)	136	136	136	ok	stationary + volatile
style ceiling / $\leq$10 non-blank lines (degenerate by construction)	323	323	323	ok	no

Table A3: Per-convention-role suite compatibility [R], regenerated by re-executing the whole builder in report-only mode (`tasks/reference_fixes.py -dry-run`; 2,353 kept pairs, every fix re-run in a subprocess against the transformed tests). "fix" counts problems where the minimal reference fix was built AND passed the class's transformed tests AND the untouched canonical failed. Roles are descriptive (module-docstring language); variants are distinguished by behaviour, never by identifier (D43). Fix-to-final attrition, all reasons on the record: 11 pairs already-passes-untouched (1 inverted-order, 8 decimal-text, 2 JSON-text); 1 decimal-text problem whose argument carries both a number and its text form (declined rather than guessed); 2 gated-refusal problems taking a caller-supplied pattern. No pair was lost to a timeout or crash. The evolution run used the stationary project's stream; the volatile pair was dropped from the evolution run by D48; the transfer project was never queued.

E.3 The ladder, per rung (ON arm)

OFF arm: the ladder is *not applicable* under `inner_loop: false`; quoting the liveness gate: "the ladder never runs and its silence is by design, not death". Every OFF node has $n_{\text{notes}} = n_{\text{rules}} = n_{\text{hooks}} = 0$; no rate is computed over this arm.

rung	reached by	attempts	promoted	rejected	dominant rejection
note (reflection)	638 failures of 700 exposures	638	91 note rows over 5 distinct sigs survive at end of lifetime	n/a	n/a
rule (`k_rule` gate, replay)	135 distinct (node, sig) pairs	331	37 (rate 0.274, $n = 135$)	294	`replay gate failed`, 294/294
hook (`k_hook` gate, AST → replay → regression)	96 gate attempts (= 96 hook-gen calls, 1:1)	96	0 installed	96	`forbidden name 'eval'`, 96/96, all at the AST gate

Table A4: The escalation ladder, ON arm (`on-stat-s1`, 28 scored nodes × 25 tasks), regenerated from the run's own event logs [R]. Signature texture: 180 distinct signatures summed within nodes; only 7 distinct signature strings pooled across the run; max within-node recurrence 9. The 96 rejected hook sources hash to only 7 distinct sha1s; the single most-retried source was rejected 64 times. All 96 hook rejections carry gate field `{"ast": false, "regression": null, "replay": null}`. The AST allowlist fired first, so replay and regression never ran.

slot	failures	distinct sigs	max recur.	rules promoted	hooks attempted	hooks promoted
slot 0	239	28	9	22	96	0
slot 1	224	112	4	0	0	0
slot 2	175	94	5	15	0	0
cross-class	0	0	0	0	0	0

Table A5: Per-slot ladder breakdown [I], an instrument reading quoted from `results/on-stat-s1/liveness.json` (`computed_at` 2026-08-24T11:40:53Z). Slots are the liveness gate's own anonymous labels, assigned in first-appearance order for this run only, mapping written nowhere (D43); they are NOT re-derivable from this table and NOT comparable to any other run. This session's independent recount from the event logs reproduces the file's 638 / 180 / 7 / 9 / 331 / 37 / 135 / 96 / 0 exactly.

The run's DEGRADED verdict reasons, quoted in full [I]: (1) "class 'class_1' has 224 pooled failures and zero class-attributed rule promotions – the ladder is alive on a subset of classes only" [`class_1` is the gate's anonymised slot label, per `harness/liveness.py`, not a class identifier]; (2) "promotion_rate 0.274 $<$ 0.50 (rules_promoted=37 / sigs_reaching_k_rule=135)"; (3) "28.6% of pooled distinct signatures collide across classes (2/7) – contradictory conventions share counters".

E.4 The evolved gene distributions

gene	range	default	ON top-10: med [min, max] (distinct)	OFF top-10: med [min, max] (distinct)
`ttl_note`	1–8	3	4 [3, 5] (3)	3 [1, 5] (4)
`k_rule`	1–6	2	3 [1, 3] (2)	2 [2, 3] (2)
`k_hook`	2–12	5	5 [5, 6] (2)	5 [4, 5] (2)
`decay`	0.0–0.5	0.1	0.1 degenerate (1)	0.1 [0.037, 0.216] (4)
`mem_budget`	0–800	300	347 [94, 421] (4)	300 degenerate (1)
`gate_replay`	1–3	2	2 [2, 3] (2)	2 degenerate (1)
`retrieval`	strength/recency	strength	recency 8, strength 2	strength 8, recency 2
`reflect`	fixed apparatus	true	true 10/10 (never mutated, D84)	true 10/10
F (reference)	n/a	n/a	0.1092 [0.1063, 0.1102]	0.0753 [0.0753, 0.0754]

Table A6: Gene distributions over the top-10 by F (ties broken on node id; population = viable scored nodes: 29 OFF, 28 ON; $n = 10$ per cell) [R]. **These columns are DESCRIPTIVE, not evidence (D109)**: medians over a selection of 10, not one genome; "degenerate" marks distinct=1, a zero-variance cell. **Every gene in this table is inert in OFF by construction**: `harness/GENES.md`, OFF condition, says that when the inner loop is disabled "genes are carried and mutated but have no effect". The OFF column is therefore mutation-and-selection drift over phenotypically silent genes, and no OFF↔ON gene difference may be read as a selection response on the OFF side. The two top-10 node lists equal the `node_ids` arrays of the two wipe-pass configs, element for element.

Table A8: Nine classes of silent instrument failure, each of which would have printed a clean, plausible, publishable, wrong number.

defect	what it would have printed (what caught it)
1. promotion ladder that never fires (call-arity drift under a bare except)	an ON arm structurally OFF, "confirming" H2 (pre-run contract audit)
2. gap computed over an empty set (writer emitted `acc_L`, reader read `acc`)	a clean hiding gap over zero rows (cross-module key diff)
3. capability claim manufactured by a parser (fence closed inside a string literal)	"the 7B cannot write viable children" (raw completions read beside verdicts)
4. recurrence counter that cannot recur (per-problem signatures)	a dead ladder whose null is exactly H2's predicted confirmation (reachability simulation before any exposure)
5. held-out stream partly lifetime-seen (fail-open seed default)	inflated innate scores as generalisation (code read; these wipes recompute at zero overlap)
6. top-10 "means" that were one node's genes (`top[0]` under a means caption)	evolved-parameter claims that were an arithmetic identity (fabricated-index probe)
7. bias controls that could never run (0 of 3 implemented, 2 unaffordable)	"the headline was control-checked" (grep and budget arithmetic; now loud refusals)
8. promotion gate replaying in an uncalibrated context (candidate appended to retrieved stack)	promotion rates inflated or zeroed, either direction (trace forensics)
9. reader that collapsed the announced flag, found *after* the run	a hiding gap of -5.0 for a true -7.0; H6's gap fabricated to exactly zero, under a green test suite (recompute-don't-trust auditing; fingerprint announced≠silent 11/20 ON vs. 1/20 OFF cells)

E.5 The price of memory

arm	call kind	calls	input tok	output tok	blended	% of arm	per node	per call
OFF (29)	task	725	54,694	57,496	342,174	100.0%	11,799	472
OFF	germline	17	unmeas.	unmeas.	unmeas.	n/a	n/a	n/a
OFF total		**742**			**342,174**		**11,799**	
ON (28)	task	700	111,984	55,013	387,049	42.5%	13,823	553
ON	reflection	638	97,563	15,511	175,118	19.2%	6,254	274
ON	replay gate	402	41,821	40,464	244,141	26.8%	8,719	607
ON	hook generation	96	7,872	19,277	104,257	11.4%	3,723	1,086
ON	germline	18	unmeas.	unmeas.	unmeas.	n/a	n/a	n/a
ON total		**1,854**			**910,565**		**32,520**	

Table A7: Blended tokens by call kind, evolution runs [R]. Blend rule: blended $=$ input $+ 5 \times$ output (`harness/accounting.py`). The inner loop's overhead (reflection + replay + hook generation) is 523,516 blended tokens, 57.5% of the ON arm's traced spend. ON task calls are 17% costlier per call than OFF's (553 vs 472, driven by retrieved memory in context: ON task input 111,984 vs OFF 54,694 for 25 fewer calls). Germline rows: all 35 mutator trace rows carry `usage: null`; counts, retries and latencies are measured, tokens are not, so the totals undercount true spend by the germline calls' unknown cost. Accounting integrity: the traced per-kind sums reproduce the harness's own ledger to the token in both arms (mean `tokens_task` $\times$ 25 $\times$ scored nodes $=$ 910,565 ON; 342,174 OFF).

E.6 The nine silent failures, with what each would have printed

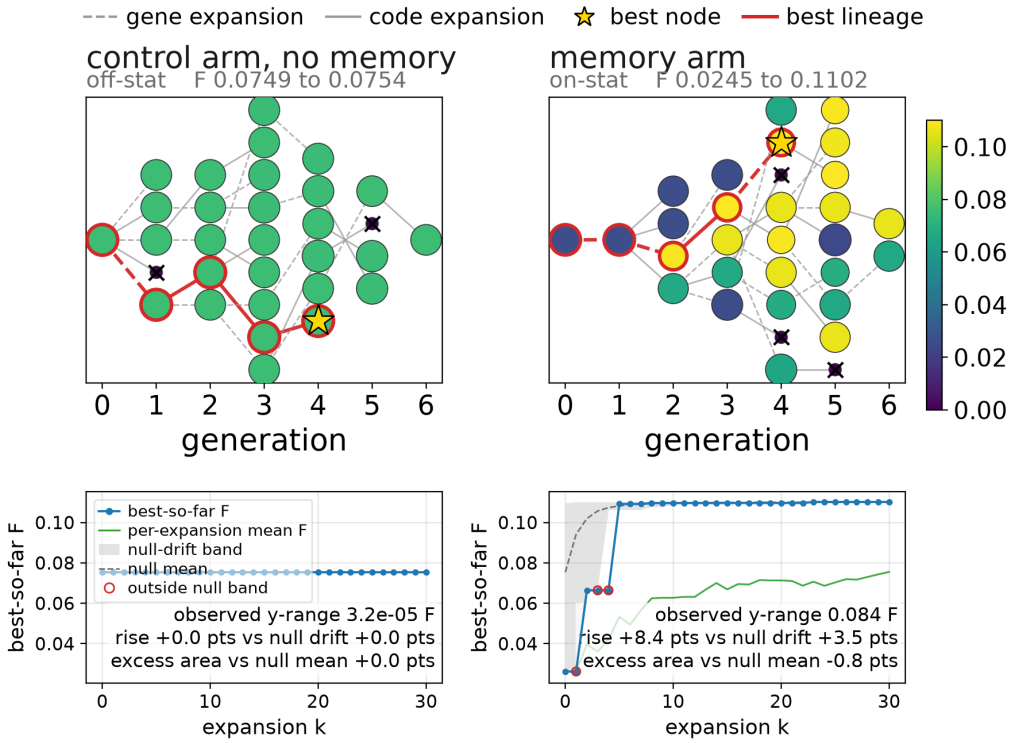
Referenced from Section 5, which names all nine in the body. This table adds the two columns the body has no room for: the number each defect would have printed had it survived, and the check that caught it.

E.7 Best-so-far against the null-drift band

Both archives as trees, and the best-so-far panel the body's archive claim depends on. The figure is here rather than in the body because at the venue's 5.5 in text width the body could not hold it and Figure 1 both; nothing in it is stated anywhere else, so it travels whole, caveats included.

E.8 The full measurement table

Table A9 is the record behind the body table, with every number, interval, sample size and caveat exactly as measured.



Bottom row (one shared y-scale): best-so-far F is a RUNNING MAXIMUM and rises under a total null. Shaded = 90% null-drift band: the running max of the same F values with creation order permuted (no evolution, heredity or selection); a curve inside the band is not evidence of improvement (D107/F12). Green = per-expansion mean F. Points below the band mark later-than-random discovery, NOT improvement; excess area vs the null mean is the only ordering-sensitive evidence term.

Figure A1: **Neither archive found anything the task model could not already do.** Upper row: one node per expansion, placed by generation depth and coloured by fitness F; x marks a nonviable child, the star each arm's best node, red its lineage. OFF is flat, best-so-far range 3.2e-05 F over 31 expansions from 29 viable nodes with one behaviour. F is not comparable across arms, which differ in the inner loop that enters F. Lower row: best-so-far F against a 90% null-drift band, the running maximum of the same F values with creation order permuted. Best-so-far is a running maximum and rises under a total null, by a simulated $+20.4$ points over 31 expansions, so the rise is not the evidence term: ON rises $+8.4$ against a null drift of $+3.5$, and that difference is not a result. The only ordering-sensitive term is the excess area versus the null mean, which for ON is -0.8: the archive found its best node *later* than a random ordering of its own values would predict. The final excess is $+0.0$ by construction, since the last value of a running maximum is the archive maximum whatever the creation order, so it is not a result either.

Table A9: The measurements. Every caveat is part of its result; intervals 95% unless marked; every number regenerates from a recorded command.

measurement, result and verdict	the caveat that travels
memory bootstrap, 27 previously failed tasks, 3 per class-role: no context 0/27; own note, original prompt 4/27 (14.8%); hand-written rule 16/27 (59.3%, Wilson [40.7, 75.5]); own note, repaired *experimenter-authored* prompt 7/27 (25.9%); first-exposure base 11.1%. measured	single-note same-task replay: an upper bound on live-gate utility. The repaired prompt is experimenter-authored germline text, a human repair, never self-repair; live prompt cache (disclosed). Information-limited, not only metacognitive: where the error text carries the convention the repaired note nearly matches the rule (7/9 vs. 9/9); where it does not, notes recover nothing (0/9 vs. 7/9).
hook rung: 638 notes and 37 rules promoted from 331 attempts upstream; then 96 hook attempts, 0/96 promoted, one reason (`forbidden name 'eval'`), one signature, 7 distinct sources (Table A1). measured	the ban is never stated by the generation prompt: a gate-outcome count that says nothing about hook-writing capability, and not 96 independent trials.
control arm: 29/29 viable OFF nodes, one accuracy value (0.08), one identical 25-task pass pattern. measured	a constant control floors every between-arm comparison.
H2 hiding gap, innate accuracy of wiped elites: -0.070, naive $[-0.095, -0.040]$, scaffold-clustered $[-0.1000, -0.0271]$ over 7/5 clusters (ON 0.120 vs. OFF 0.190), n=10/arm; the sign H2 predicted. an instrument reading, not citable	MDE $\geq$ 10.1 points exceeds both the observed gap and the $\pm$5 kill line: *this run cannot distinguish the arms*. ON top-10: one depth-1 ancestor, 3 distinct response vectors (OFF: 5, 2), so node-level n overstates the evidence. All three preregistered selection-bias controls are uncomputable ($\sim$840-exposure census unbought), leaving the gap inseparable from selection composition: the root scaffold scores 0.20 innate in *both* arms, 9/10 wiped OFF elites are transcript-identical to root, and 2 of 19 items carry the gap.
wiped OFF elites, memory effect: innate 0.190 vs. plastic 0.055, n=10: an un-selected inner loop degrades a scaffold evolved without it. an instrument reading	OFF's inner loop was under no selection during evolution, so its plastic cells are off-distribution by construction: a reading about the apparatus, not a memory-content result.
H6 announce gap, innate: ON $+2.0$ points, 90% TOST $[+0.5, +3.5]$, n=10: not bounded. OFF: zero variance, zero discordant pairs, degenerate	ON escapes the preregistered $\pm$3 band; 4 discordant pairs, sign-test floor p=0.125: no achievable statistic from this wipe rejects anything. The OFF cell is a label, never a numeric interval.
H4 child viability: ON$-$OFF mean child$-$parent accuracy delta -0.0033 $[-0.020, +0.010]$, n=12. inconclusive	OFF cell degenerate: 13 identical deltas, a zero-variance resample. A label, not an interval.
H1 fixed-gene policy study: pooled memory policies (always-rent, always-buy) minus static: $+0.0104$, task-clustered $[-0.0512, +0.0690]$; n=155 distinct base tasks (480 memory / 240 static rows; two seeds, 120 exposures each). inconclusive. Clause 2 (cost-derived not dominated): break-even recurrence 104–256 vs. gene ceiling 12, not tested	decision rule and estimator authored before the result file existed; the record discloses its own ordering violation. The run launched before the rule row landed, caught mid-run and recorded, not concealed: "blind" means no result was observed before the rule was fixed. Clause 2's arm confirms by construction and was excluded; the refusal is the record.