

What actually makes an agent harness work

Techniques, prompt engineering and measured results from three agent harnesses on one model, and what happens when the instrument turns out to be the bug.

Nikolaos Broikos • broikos.gr August 2026

45 graded sessions • 3 harnesses • 102 labelled prompts

36.3%

LOCAL RESOLUTION, AGAINST A DOCUMENTED CLAIM OF 60 TO 70%

2.0%

SHARE HANDLED BY THE TEN-DIMENSION SCORER

0.972→0.833

TOLD WHAT TO FIX, VERSUS HAVING TO FIND IT

45

GRADED SESSIONS ACROSS THREE HARNESSSES

01 What this is

I built both of the harnesses this paper evaluates, and I built the benchmark that evaluates them. That is the largest conflict of interest in anything I have published, and it belongs in the first paragraph rather than the last section.

There is no way to remove it. What was done instead: every grader executes the resulting code, so nothing is scored by reading a transcript or asking a model for an opinion. The tasks, seed project, reference implementation and every raw result are published. The benchmark ships a negative control that refuses to report unless the graders can demonstrably fail, plus a second control proving the damage axis can fire. The predictions for the one architectural experiment were written down before it ran. And the results that make my own design look worst are in the body, not the appendix.

A reader who discounts this work for its authorship is making a reasonable judgement. The data is published so that they can check it rather than trust me.

The argument

Agent harnesses are built on techniques that almost nobody measures, routers, planners, repository maps, tool-set narrowing, prompt caching, context compaction. The literature is architecture diagrams and capability claims. When you actually measure two harnesses, three things fall out:

The elaborate parts do less than the simple parts. A ten-dimension weighted prompt scorer, the most sophisticated code in its file, resolves 2% of requests. A hand-written list of greetings resolves 34%.

Adding architecture can subtract. A planning pass and a repository map, run on the same tasks with the same model, made a reliably-solved task unreliable and cost 38% more.

The hard part is not fixing a bug. It is finding it. Given identical defects in identical code, an agent told what to fix scored 0.972. The same agent, given only a customer complaint, scored 0.833, and in two runs of three never found the defect the brief did not point at.

What this paper is, precisely

One harness measured end-to-end. One harness measured at the component level. Four harnesses compared architecturally, from their own source and repositories.

Orange, Python, 166 source files, is measured on a purpose-built benchmark across thirty graded runs, including an architectural ablation. **Axium**, Rust, 28 source files, with a Python port used for the head-to-head, has its prompt classifier measured exhaustively against 102 labelled prompts, and runs end-to-end across 61 bench runs and 15 graded sessions. **Hermes Agent**, which the author did not write, was installed from its own repository and driven through the identical scenarios for a further 15 sessions. **OpenClaw** is described from its repository, licence and documentation, retrieved at primary source on 6 August 2026, and is not measured.

Section 08 sets 45 sessions of the three measured harnesses side by side on one model. Section 09 records three faults found in the measuring apparatus itself, two of which had already produced published findings that were wrong.

What is not claimed

No measured comparison against OpenClaw. **It is installable and the benchmark would accept it, but it is TypeScript and an order of magnitude larger to install than the alternative, and the sandbox had to be fully reversible. Every statement about OpenClaw in this paper is architectural, read from its own repository, and none of it is a measurement.** That is the largest remaining gap in the work.

No general claim about model capability. Everything graded here is one model, one language, a two-hundred-line repository, and three repetitions. The figures are directions, not coefficients.

No claim that these three harnesses are representative. They are three that could be run identically on one machine in one week. Two were built by the author, which is stated here, in section 09, and in the declaration of interests.

02 **Four harnesses, one lineage**

Discharges claims C010 to C017. Third-party facts verified at primary source on 6 August 2026 via the GitHub REST API and the projects' own README and LICENSE files. Axiom and Orange facts read from source, with file and line references throughout.

Two of the harnesses in this study were built by the author. Two were not. Establishing what the other two actually are, rather than what commentary says they are, turned out to matter, because the relationship between all four is closer than "similar tools" and it is documented in their own repositories.

What the other two are

	OPENCLAW	HERMES AGENT
repository	github.com/openclaw/openclaw	github.com/NousResearch/hermes-agent
language	TypeScript	Python
stars at retrieval	385,338	226,388
licence	MIT	MIT
last push at retrieval	2026-08-06 12:43 UTC	2026-08-06 12:06 UTC
self-description	"Your own personal AI assistant. Any OS. Any Platform."	"The agent that grows with you"

Both had been pushed to on the day they were checked. Neither is dormant, and neither is small.

One metadata note, small but instructive. **The GitHub API reports OpenClaw's licence as NOASSERTION; the repository's own LICENSE file is standard MIT.** Third-party write-ups saying "MIT" are correct, but correct without having checked, since the machine-readable field says otherwise. The file is authoritative, and it is a small illustration of why this study opens documents rather than reading about them.

The architecture they share

OpenClaw's README describes its own shape plainly: a **Gateway**, *"the local control plane for sessions, tools, events, and channel connections"*, with a Control UI, CLI and TUI attached to it, channels for WhatsApp, Telegram, Slack, Discord, Signal and iMessage, companion nodes adding voice and screen, hosted and local model providers, and **tools, skills and plugins** as three distinct extension surfaces.

Set that beside Axiom's module map and the correspondence is close to one-to-one: tui/server.rs is the web control surface, channels/cli.rs and channels/telegram.rs the CLI and messaging channels, worker.rs and the task database the session plane, with axiom-skills/ and plugins.json alongside its 31 tools. Orange reaches the same three extension points by a different route.

Four harnesses, three extension surfaces each, the same split between a persistent local process and thin clients attached to it. Convergent design would be unremarkable: the same problems invite the same answers. What follows is not convergence.

The persona file, and why it is evidence of lineage

Hermes ships a migration command, and its README documents what it moves:

Migrating from OpenClaw, hermes claw migrate. The setup wizard automatically detects ~/.openclaw and offers to migrate before configuration begins.

The first item in its list of what gets imported is:

SOUL.md, persona file

Hermes is built to inherit an OpenClaw installation, down to the file that defines the assistant's personality. And that same convention appears in both harnesses built independently by this paper's author: Axiom ships a soul.example.md and loads it as the **cached static half of its system prompt** (sonnet.rs:536); Orange resolves a soul_path setting through _load_soul(), with a _FALLBACK_SOUL constant if the file is missing (agent.py:64-86).

Four harnesses, in three languages, by at least three different authors, all keeping the assistant's personality in an external file called some variant of *soul*. That is a shared lineage rather than four people independently choosing the same unusual word.

Where they genuinely disagree

Two design decisions separate Axiom and Orange sharply, and both are measurable rather than matters of taste.

Who decides how to spend money. Axiom classifies each prompt through a three-stage path that resolves most cases locally, deterministic patterns, then a weighted scorer, and an LLM only when confidence falls below a threshold (classifier.rs:917-947). Orange routes every single turn through a model call that returns strict JSON (pipeline.py:23-37). Axiom's is free and hand-tuned; Orange's costs latency and money on every turn but adapts. Section 03 measures the first of these.

How much surface to show the model. Axiom exposes **31 tools**, and narrows to **18** for prompts classified as simple, excluding *"subagents, task queuing, code intelligence, destructive ops"* (sonnet.rs:52-66). Orange exposes **69** and never narrows. Fewer tools means fewer prompt tokens and less opportunity to wander; more tools means never being unable to act. Both positions are defensible and the trade has a measurable cost.

Where they agree without having coordinated

The strongest convergence in the inventory is one neither harness advertises. **Both split the system prompt into a cached static block and an uncached live block.**

Axiom assembles a static soul plus a dynamic block carrying memory and tasks, tagging both with `cache_control` at a one-hour TTL, and caches the tools array and the last conversation message the same way (`sonnet.rs:522-541, :28-36, :988`). Orange builds `_stable_system_prompt()` once at module level and appends `_live_context()` per turn, with a comment explaining exactly why: the prompt is rebuilt on every turn (`agent.py:89, :115, :217`).

Two authors, two languages, the same answer to the same billing pressure.

One constraint worth stating early

Axiom resolves model providers to **Anthropic** for models beginning `claude-`, and **OpenAI** for everything else (`agent/mod.rs:45-64`). There is no third path. Orange's benchmark results in this paper are on a DeepSeek model, which Axiom cannot currently run.

That is why the measured comparisons in sections 06 and 07 are of Orange only. Any Axiom-versus-Orange table produced today would compare *harness-and-model pairs* rather than harnesses, and section 08 explains what was done about that.

No measured head-to-head against OpenClaw or Hermes was run, for reasons section 08 sets out. The comparison in this section is architectural, sourced to each project's own repository, and labelled as such wherever it appears.

03 Measuring a classifier that costs nothing to measure

Discharges claims C001 to C009. All figures from `bench/classifier/results.csv`, run 6 August 2026, 102 labelled prompts, no API calls, cost \$0.00.

Most claims about agent harnesses cannot be checked cheaply: they involve a model, a bill, and a run-to-run variance that makes any single number an anecdote. Axiom's prompt classifier is the exception, and that is why the measurements start here: its decision path is deterministic, returning the same class for the same prompt every time, for nothing. It also carries a claim precise enough to be wrong. `src/agent/classifier.rs:918`:

"The scorer handles 60-70% of requests locally; the LLM handles ambiguous cases."

That is a falsifiable statement about traffic. It took a labelled prompt set and an afternoon to check, and it is the kind of claim that sits in a codebase for months because checking it is nobody's job.

What the classifier does

Axiom sorts every prompt into one of four classes before deciding how to spend on it. Trivial is answered by the cheap classifier itself, so the primary model is never called. Simple passes through with a reduced eighteen-tool set. Medium uses the primary model but skips quality and code review. Complex has its prompt rewritten before the primary model sees it.

The decision runs in three stages: a hand-written list of greeting, identity and memory patterns; then a weighted scorer over ten dimensions, reasoning 0.18, code presence 0.15, multi-step 0.12, technical terms 0.10, a length proxy 0.08, simple-question indicators at **minus** 0.08, agentic 0.06, creative 0.05, constraints 0.04, structured output 0.03, plus two hard overrides; then, only if confidence falls below 0.75, an LLM call.

The intent is sensible: spend nothing on easy cases, a cheap model on ambiguous ones, and reserve the expensive model for work that needs it.

The measurement

The scorer and pattern matcher were transcribed to Python dimension by dimension, keeping every weight, keyword list, cap, threshold and hard override, with parity notes recording each construct checked, including one deliberate preservation: dimension two matches the original-case string while every other matches a lowercased one, so Class is not code presence but class is.

The prompt set is 102 items labelled against Axiom's own class definitions, with the labelling rule written before anything was scored. It deliberately includes the adversarial cases Axiom's own prompt guards against, capability and memory questions that look trivial and must not be treated as such.

The claim does not hold. Local resolution is 36.3%, thirty-seven prompts of a hundred and two.

Where the work actually happens

The aggregate is the least interesting part. The breakdown is this:

STAGE	WHAT IT IS	RESOLVED	SHARE
1	hand-written keyword patterns		35 34.3%
2	the ten-dimension weighted scorer	2	2.0%
3	deferred to the LLM		65 63.7%

The most elaborate component in the file resolves two prompts in a hundred. A list of greetings, identity phrases and memory verbs: the least sophisticated thing in the module, the kind of code that gets apologised for in review, does seventeen times more work than the scoring apparatus it exists to feed.

Why the scorer almost never fires

This is not a tuning problem, and that matters, because a tuning problem would be fixed by moving a threshold.

Confidence is computed as a sigmoid on the distance from the **nearest** tier boundary:

$$\text{confidence} = 1 / (1 + \exp(-10 \cdot \text{min_distance}))$$

The boundaries sit at -0.02 and 0.25. Clearing the 0.75 gate therefore requires a score at least 0.11 away from *both* of them. But the Medium band between the boundaries is only 0.27 wide, so a score landing anywhere inside it is at most 0.135 from a boundary: a ceiling of roughly 0.794 confidence, reachable only at the exact midpoint.

The observed distribution turns that geometry into a wall. Of the 67 prompts that reach the scorer:

median confidence	0.5498
90th percentile confidence	0.7311
clearing the 0.75 gate	4 of 67
median score	-0.016
landing inside the Medium band	35 of 67, 52%

The median score is -0.016, which sits **four thousandths from the -0.02 boundary**. The typical prompt lands almost exactly on a decision boundary, which is precisely where a distance-to-boundary confidence function is least able to be confident. Half of all prompts fall into a band where high confidence is arithmetically unreachable.

The scorer is not badly calibrated. It is asked to be certain in the one region where its own certainty metric cannot be high.

When it does decide, it is right

Accuracy on locally-resolved prompts: 36 of 37, or 97.3%. The confusion matrix is nearly diagonal. This is a high-precision, very-low-recall component, and the precision is real.

The single error deserves quoting in full, because it is the failure mode that costs something other than money:

first audit the api for injection risks, then fix them, then add regression tests

labelled Complex → classified **Medium**, score 0.1372, confidence 0.7555

A three-stage security task cleared the confidence gate by 0.0055 and was routed to the pipeline that **skips quality review and code review**. One case in 102, and the one case the design should least want to lose.

Two structural findings

Trivial cannot be reached without paying for the LLM. Despite its name, quick_classify_trivial returns PromptClass::Simple in every branch. All ten arithmetic and general-knowledge prompts deferred to the LLM classifier: one hundred per cent: five of them sitting at confidence 0.7311, nineteen thousandths below the gate. So "what is 2+2" pays for a classifier call *in order to be told* that the cheap model could have answered it. The saving the Trivial class exists to capture is partly consumed by the call that identifies it.

A keyword that has never fired. classifier.rs:848 places "O(" in the constraints list; classifier.rs:851 matches that list against the lowercased prompt. A capital O cannot appear in a lowercased string, so the complexity-notation signal the dimension was written to capture has never once matched in production. Weight 0.04 with the count capped at one, so the practical effect is small, but the intent is entirely defeated, and it was found by transcription rather than by reading.

What this section does not establish

The labels are one considered view by the person who then analysed the results; there is no second coder, and the accuracy figures inherit that weakness. The prompt set is author-chosen and weighted toward conversational traffic, thirty-five of 102 are greetings, identity or memory questions, so **36.3% is probably an overestimate of local resolution on real coding-assistant traffic**, which contains fewer greetings. The port was verified by transcription and reasoning rather than differentially against a compiled binary. And nothing here measures the LLM classifier above the local path, which would require the API calls this section was designed to avoid.

None of that touches the central result. The claim was 60 to 70%; the measurement is 36.3%; and the reason is a specific interaction between boundary spacing and a distance-based confidence function that no amount of keyword tuning will address.

04

What a saturated benchmark cannot tell you

Discharges claims C019 to C022. Figures from data/brownfield-deduped.csv and data/analysis-brownfield.md, derived from Orange's own evals/logs_big/*.jsonl, run window 2026-07-20 21:02-22:03. Prices from pricing.py, fetched 2026-07-05, re-verified 2026-07-19.

Orange arrived at this study with something most agent harnesses do not have: a working evaluation suite. Twenty-four scenarios against a generated forty-five-file PHP project, eight bug fixes, eight cross-file refactors, eight codebase-awareness questions, with objective graders, cost tracking, and switches that turn the harness's own architectural features on and off.

It is better instrumented than most published agent benchmarks. It is also, for the model it was built to measure, almost entirely uninformative. Both of those facts are worth the reader's time.

A data-integrity finding, before any result

The raw logs contain **four superseded reruns**, and aggregating them naively produces the wrong number.

Four task ids appear twice in the pro-bare log, A2, B2, B7 and R2, and in each case the later timestamp is the corrected value:

ID	FIRST RUN	RERUN	WHY
B7	0.667 @ 21:09:44	1.000 @ 21:46:40	grader bug: a text-slice comparison that always compared the identical template fragment, fixed to an id-set comparison
R2	0.667 @ 21:03:20	1.000 @ 21:14:01	rerun after the same fix
A2, B2	1.000	1.000	reruns, unchanged

A first pass over these logs without deduplication reported change **0.976** for the bare configuration. The correct figure is **1.000**. The entire difference consists of records produced by a grader that was later found broken and repaired.

The rule that follows is small and load-bearing: **deduplicate by (configuration, task id) keeping the latest timestamp, and say so**. Anyone re-analysing these logs without that step publishes a number that is both wrong and slightly worse than reality: which is the direction of error least likely to be questioned.

One residue is flagged rather than hidden: B7 in the brain-and-plan configuration still reads 0.667 on that same repaired assertion, because that configuration was never re-run. Its true value is very probably 1.000, but this paper does not silently correct numbers it did not observe.

The results, corrected

Twenty-four tasks per configuration: eight fix, eight refactor, eight awareness.

CONFIGURATION	CODING (16 TASKS)	REGRESS	COST	AWARENESS
deepseek-v4-flash, bare	16/16	1.000	\$0.0716	0.917
deepseek-v4-pro, bare	16/16	1.000	\$0.1856	1.000
deepseek-v4-pro, brain + plan	15/16 ¹	1.000	\$0.2031	0.958

¹ the single miss is the B7 grader artefact described above.

Whole-suite totals: flash \$0.0853 over fifteen minutes; pro \$0.2249 over eighteen; pro with brain and plan \$0.2374 over nineteen.

The expensive coder bought nothing

Flash and pro both scored **sixteen out of sixteen on every coding task**, all eight planted bugs fixed, all eight cross-file refactors completed, not one broken page across the whole application.

Flash did it for **\$0.0716 against \$0.1856, roughly 2.6× cheaper**, and finished the entire suite in fifteen minutes against eighteen.

This is a result against the harness's own design. Orange routes work between model tiers precisely so that hard tasks reach the capable model. On this suite, at this scale, the routing bought no correctness whatsoever. It is the kind of finding that only appears when someone measures a feature they already believe in.

The caveat belongs immediately next to it: forty-five files is not a real project, one repetition is not a sample, and the author's own documentation makes the reasonable argument that a pro-tier premium of pennies per task buys headroom above what this suite can detect. That argument is sound. It is also, as stated, unmeasured.

Where flash actually loses

Not on code. On claims.

Flash's only failures were in the awareness family, where the agent answers a question about the codebase and its answer is graded by F1 against a truth set of file paths. It **invented a file that does not exist** in the A3 data-flow trace, and reached **precision 0.47** on the A7 inventory, more than half the files it named as relevant were not.

Flash writes correct code and unreliable statements about code. That distinction is more useful for harness design than the aggregate score, because it argues for routing by *task type*, claims versus edits, rather than by estimated difficulty.

What saturation costs

Forty-eight of forty-eight coding tasks passed across all three configurations, and a benchmark on which everything passes cannot rank anything, not two models, not two architectures, not two harnesses. Orange's own documentation reaches the same conclusion and lists the hardening levers: compound interacting bugs, vaguer briefs requiring diagnosis before repair, a larger seed, and treating cost and tool-calls as first-class axes. Those levers are the design brief for the next section.

05 Building a benchmark that discriminates

Discharges claims C023 to C025. Describes bench/tasks/, built 6 August 2026.
Sanity control verified the same day: 0 problems, 13 regression assertions.

Section 04 ends on a specific defect rather than a general complaint. Orange's existing suite recorded zero regression damage across forty-eight runs, and **nothing in it demonstrated that the regression assertions were capable of failing**. A perfect score is equally consistent with a careful agent and with an assertion set that cannot fire. The result is uninterpretable in either direction.

Everything below exists to make that particular mistake impossible.

Four rules

Grade the artefact by executing it, every assertion runs the resulting code. No transcripts, no model judging correctness. An agent that explains a fix beautifully and does not make it scores zero.

Two independent axes, never pooled. change asks whether the required behaviour appeared; regress asks whether anything else broke. Separate assertion sets, reported separately, and the interesting runs are where they diverge.

A negative control gates every result, described below.

Every expected value is derived by executing a reference build, never hand-written: a hand-typed golden is an assumption wearing the costume of a measurement, and it fails exactly where the code is subtle.

The seed

shopkit is a small order and inventory toolkit in Python, five modules, five products, five orders, no server and no dependencies, so the suite can be pointed at another harness on another machine. Three defects are planted, each fixed in a reference copy that differs at those three sites and nowhere else:

	SITE	DEFECT
T1	money.fmt	<code>floor(v*100)/100</code> truncates instead of rounding, losing a cent
T2	reports.is_overdue	<code>due <= today</code> flags orders due today as already overdue
T3	orders.discount_amount	stacked discounts are summed , so two 60% discounts give 120%

The third produces a grand total of **-22.32** on one order, which is the kind of defect that is obvious once seen and invisible until someone looks.

The tasks

Five. T1, T2 and T3 each name one defect with its symptoms. T4 names all three and adds *"without breaking anything else"*. T5 names none of them. **T4 and T5 hold identical defects, and the only variable is whether the agent is told:** the pairing the whole suite exists for, measured in section 06.

The negative control

sanity() runs before any result is reported and asserts four things:

1. The regression set is **green on the pristine, unfixed seed**, so any later regression failure is the agent's doing and not a broken baseline. 2. **Every change-grader fails on the pristine seed**, so no grader can pass for free. 3. **Every change-grader passes on the reference build**, so the task is actually achievable. 4. The regression set is **green on the reference build**, so the intended fix does not itself regress.

If any condition fails, the harness refuses to run. Verified 6 August: *0 problems, 13 regression assertions*.

This is the cheapest possible insurance and almost no published agent benchmark carries it. It costs one function and it converts "everything passed" from an ambiguous result into an interpretable one.

Proving the regression axis can fire

The negative control shows the graders discriminate on a pristine build. It does not show that the *regression* axis can detect real damage, because a pristine build has no damage.

So there is a second control: a vandal adapter that makes deliberately wrong edits, mistakes an agent could plausibly make while attempting T1, and the axis grades them across a range:

WHAT THE VANDAL DID	REGRESS	ASSERTIONS BROKEN
nothing	1.0000	0
fixed the rounding but changed the return type	0.9231	1
tidied the CLI and lost its non-zero exit codes	0.9231	1
rounded in the wrong place, shifting an untouched total	0.8462	2
"fixed" the price in the dataset instead of the code	0.7692	3
left the build unparseable	0.0000	13

Six distinct outcomes across five classes of damage. When this suite reports regress 1.000, that is now a statement about the agent rather than about the assertions.

The fourth row earns its place: an agent that edits the seed data to make a symptom disappear has fixed nothing, and a benchmark checking only the reported symptom would score it a success.

A defect found in the harness while building it

The first version **crashed** when an agent left the build unparseable, instead of scoring zero. An agent that produces a syntax error is precisely the case a benchmark must handle, so the failure mode was the wrong one entirely.

It now catches the exception, scores zero, recovers the assertion names from the reference build so the failure is still itemised rather than collapsing into a single row, and records the exception text. The `syntax_error` row above is that fix being verified.

Adapters

A harness is plugged in by implementing one function:

```
def run(build_path, brief):  
    """Make edits under build_path to satisfy brief. Return a metadata dict."""
```

Nothing parses the harness's output, because the graders execute the resulting code: which is what keeps the interface small enough to be worth writing. The Orange adapter is about ninety lines and drives Orange's real coder loop rather than a reimplement. One trap, recorded because it cost time: an adapter named after the harness it drives will **shadow that harness's own package** on the module search path.

What this suite is not

Five tasks, two hundred lines, Python only. It measures localisation and care, not scale, and the fact that T4 and T5 keep moving is a property of the *briefs* rather than the codebase. Section 09 sets out the rest of its limits, including the one that matters most here: across the thirty graded runs that follow, the regression axis recorded **no damage at all**. A demonstrated-working instrument that a competent agent never tripped.

06 Told versus find

Discharges claims C026 to C032. All figures from `data/bench-reps.jsonl`, tag `flash-bare`, run 6 August 2026: 15 runs, 3 repetitions × 5 tasks, `deepseek-v4-flash` at effort high, no brain preload and no planner. Prices from Orange's `pricing.py`, fetched 2026-07-05 and re-verified 2026-07-19. Total spend \$0.0906, mean \$0.0060 per run.

Two of the five tasks in `shopkit-bench` contain **exactly the same three defects, in the same three files, in the same repository**. The only difference between them is what the brief says.

T4 names all three:

This billing module has several defects. Amounts are displayed a cent low, the overdue report includes orders due today, and orders with two discounts can produce a negative total. Fix all of them without breaking anything else.

T5 names none of them:

Customers are complaining that the totals on their invoices look wrong, and one customer was quoted a negative amount. Work out what is going on and fix it.

Same model, same harness, same code, same graders. The difference is diagnosis.

The result

	CHANGE	MEAN CALLS	COST OVER 3 RUNS
T4, told what to fix	0.972 [0.917 to 1.000]	15.3	\$0.0251
T5, must find them	0.833 [0.750 to 1.000]	18.0	\$0.0229

A 0.139 drop in correctness, for 1.17× the calls.

The ranges are printed because they matter more than the means. Neither task gave the same answer three times, and a single run of either would have been an anecdote: which is exactly what most published agent benchmarks report.

The failure is specific, not diffuse

The run-by-run detail is where the finding sharpens:

TASK	REP	CHANGE	CALLS	FAILED ASSERTIONS
T4	1	1.000	10	,
T4	2	0.917	19	fmt(0.005)
T4	3	1.000	17	,
T5	1	0.750	19	fmt(0.005), overdue set, due-today excluded
T5	2	0.750	12	fmt(0.005), overdue set, due-today excluded
T5	3	1.000	23	,

The two failing T5 runs failed **the same three assertions**. Two of those three are the overdue-boundary defect: the one bug the brief gave no hint toward at all. The negative total was mentioned explicitly, and the discount defect was found and fixed in every single run. The rounding was gestured at by "totals look wrong", and was mostly found.

The shortfall lands precisely on the defect nothing pointed at.

But it is inconsistency, not incapacity

T5 rep 3 scored 1.000. The agent found all three defects, including the un-hinted one, working from the vague brief alone. So this is not a capability ceiling: the task is achievable, and the same model achieved it.

It achieved it one time in three.

That distinction changes what the number means. "An agent scores 0.833 on vague briefs" sounds like a stable property of the model. What is actually happening is that it either conducts a thorough search or it does not, and which one you get is a coin weighted about one in three. A single successful demo of exactly this task would have been entirely genuine and completely misleading.

The edge case that only appears when nobody mentions it

`fmt(0.005)`: a value that must round to 0.01, failed three times across the fifteen runs. Twice in T5, once in T4, and **never once in T1**, the task whose brief names the rounding bug and supplies worked examples. T1 scored 1.000 in all three repetitions.

So the model fixes rounding correctly when the requirement is stated, and writes a rounding fix that misses the half-cent boundary when it has to infer the requirement itself. Every plausible implementation returns 0.01 for this input when tested directly: Python's `round()`, `floor(v*100+0.5)/100`, Decimal quantised from a string, Decimal quantised from a float. Whatever the agent wrote in the failing runs was none of them.

What it actually wrote is unknown. The build directories were deleted after grading and the artefact was never captured. That is a gap in this study, not a finding, and it is recorded as one.

Agents leave the room untidy

Five of fifteen runs, a third of them, left a scratch file behind: `_check_tmp.py`, `_verify.py` twice, `_check.py`, `_verify_fixes.py`.

The agent is writing its own verification scripts, running them, and not cleaning up. Good behaviour in bad clothes: it is checking its work, which is more than the brief asked for. But every one of these files would be flagged in a human code review, and no assertion here catches them.

What this section does not establish

Section 09 sets out the limits in full; two matter here. The brain preload and the planner are absent from every figure above, whether they close the diagnosis gap is section 07's question, deliberately not answered in advance. And the three specific-brief tasks, T1 to T3, scored 1.000 on both axes in all three repetitions: for this model at this difficulty they measure nothing, a reminder that a benchmark's useful range is narrower than its task count suggests.

07 Does the architecture earn its keep?

Discharges claim C033. Figures from `data/analysis-ablation.md`, tag `flash-brainplan` against tag `flash-bare`, both run 6 August 2026: 15 runs each, 3 repetitions × 5 tasks, `deepseek-v4-flash` at effort high, executed one at a time. Identical in every respect except two switches.

Orange's architecture has two features that exist to make the coder better before it starts work. The **brain** builds a map of the repository and injects it as context. The **planner** runs a separate model pass that produces a numbered implementation plan, which the coder is then instructed to follow exactly and not to re-plan.

Both are defensible. Both cost money on every task. Neither had ever been measured on a benchmark capable of showing a gain.

Section 04 explains why the previous attempt could not answer this: the older suite was saturated at 48 of 48, so an architectural improvement had nowhere to show up. This suite is not saturated. T4 and T5 both move. If the brain and the planner help, this is where it would appear.

The predictions were written down before the run, in the project's handoff file, so the result could not be reinterpreted afterwards:

If the brain helps, T5's score should rise and the overdue-boundary defect should stop being missed. If the planner helps, T4 should stabilise at 1.000 rather than 0.972. If the earlier finding repeats itself, both stay flat and cost rises: which is a publishable negative result.

The result

TASK	KIND	BARE	BRAIN + PLAN	DELTA
T1	fix	1.000	1.000	0.000
T2	fix	1.000	1.000	0.000
T3	fix	1.000	0.833 [0.500 to 1.000]	-0.167
T4	compound	0.972	0.944	-0.028
T5	vague	0.833	0.861	+0.028

Cost rose from **\$0.0060 to \$0.0083 per run, 38% more**, and call counts rose on every single task: 16.7 to 19.0, 8.7 to 11.7, 13.0 to 17.7, 15.3 to 18.7, 18.0 to 22.7. Wall-clock for the sweep went from roughly twenty minutes to 32.1.

Nothing got better. One thing got materially worse.

T3 broke, and nobody predicted it

T3 asks the agent to make stacked discounts compound rather than sum. Bare, it scored **1.000 in all three repetitions**, one of the most reliably solved tasks in the suite. Under the ablation it scored **0.833, with one run collapsing to 0.500**.

The failed assertions name the mechanism precisely:

- SO-1005 discount == 6.6446
- single-discount order unchanged

The agent **fixed the stacked-discount case and broke the single-discount case**. It solved the problem it was pointed at and damaged the neighbouring behaviour, and it did so only when given a plan and a repository map.

This is the strongest single number in the ablation, and it runs against the architecture it was built to test. Adding a planning pass to a task that was already being solved reliably made it unreliable.

The prediction that mattered failed

T5 did rise, by 0.028. That is one assertion on one run out of three, and it is inside the noise of this sample.

More to the point, **the mechanism the prediction named did not change at all**. The overdue-boundary defect: the one the brief gives no hint toward, the specific failure section 06 identified, was still missed in **two of three runs**, exactly as it was bare. If the repository map helps an agent find what nothing points it at, this suite could not detect it.

Prediction two was falsified outright: T4 got worse rather than stabilising, and `fmt(0.005)` failed twice under the ablation against once bare.

Debris nearly doubled

Ten of fifteen runs (67%) left a scratch file, against five of fifteen (33%) bare, and one run left four of them (`_cleanup.py`, `_cleanup2.py`, `_final_cleanup.py`, `_sanity_check.py`). The planner appears to encourage the agent to write and run its own verification scripts, defensible on its own terms, but it doubles the litter left behind.

What this establishes, and what it does not

On this suite, with this model, the repository-map preload and the planning pass changed nothing on the tasks that already passed, made a reliably-solved task unreliable, slightly worsened the compound task, failed to fix the diagnosis gap they were supposed to address, cost 38% more, and nearly doubled the debris left behind.

Three caveats, and the first is serious. **The two features were switched together**, exactly as in the earlier attempt, so nothing here attributes any of this to the map or to the planner individually. A proper 2×2 has never been run, and until it is, the honest statement is about the pair. **Three repetitions** is enough to show T3 became unstable; it is not enough to put a bound on any of the smaller deltas. And **the seeded repository is about two hundred lines**, which is precisely where a repository map has least to offer.

That last point is the author's own standing counter-argument and it deserves stating properly rather than dismissing: the brain's value is orientation speed on real projects with history and mess, which no seeded benchmark reproduces. That argument is entirely reasonable. It is also, after two attempts to measure it, still unmeasured, and it now has to survive the fact that on the one suite capable of showing a gain, the feature made a solved task fail.

08 What the harness decides, and what the model decides

Discharges claims C034, C018, C035, C039, C040, C041.

Version 1.0 of this paper carried a section called *What was not measured, and why*. Its first line was that Axiom had never been run end-to-end, and its argument was that publishing a comparison would have compared harness-and-model pairs rather than harnesses. That constraint has since been lifted, so the section is gone and this one replaces it.

Two things changed. Axiom acquired a Python port that runs on this machine and speaks DeepSeek, which removes the provider objection entirely. And a third harness was brought in: **Hermes Agent, which the author did not write**, installed from its own repository into a disposable sandbox and driven through the identical scenarios.

Forty-five graded sessions. Three harnesses, one model, one grader, one price table. Every agent got deepseek-v4-pro for coding turns, byte-identical fresh copies of the same seed project, and scenario text containing no tool names and no framework vocabulary. Nothing is graded from what an agent says it did.

The five axes

ID	AXIS	WHAT THE SESSION ASKS
V1	repair	fix two real defects in one session, without the second fix reverting the first
V2	restraint	three explicitly READ-ONLY questions
V3	continuity	a rule in turn 1, four turns of unrelated volume, recall it in turn 5
V4	blast radius	<i>"just delete the stuff we don't need"</i> , then <i>"put it back exactly"</i>
V5	economy	trivia, a lookup, a small feature, a real fix, a changelog

The result

ID	AXIS	AXIUM	ORANGE	HERMES	
V1	repair	100.0%		87.5% 100.0%	
V2	restraint	95.2%		85.7% 95.2%	
V3	continuity		81.0% 85.7%		81.0%
V4	blast radius		97.2%	91.7% 100.0%	
V5	economy		94.4% 100.0%		91.7%
	mean	93.6%		90.1% 93.6%	

regress was 100% for every agent in every one of the forty-five sessions. On the axis a deliberate vandal control demonstrates can detect five distinct classes of collateral damage, nothing was ever damaged. That is the least interesting row in the table and the most reassuring.

The convergence is the finding

Axiom and Hermes share no code, no language heritage and no author. On V1 they scored identically. On V2 they scored identically **and failed the same single check**, picked a real corruption risk, once each, in one repetition of three.

	REP 1	REP 2	REP 3
Axiom V2	7/7	7/7	6/7
Hermes V2	7/7	7/7	6/7

Two unrelated harnesses, one model, indistinguishable output including the mistake. **On diagnosis and on restraint, the model is doing the work.** No amount of harness engineering moved those numbers, because those numbers were never the harness's to move.

This bounds the whole enterprise. A reader deciding where to spend effort should read the convergence before the differences: two of the five axes were decided by the model, and the harness contributed nothing measurable to either.

Where the harness does decide

Three axes separated, and each names a concrete design decision rather than a quality gradient.

Blast radius (V4). The session opens with a deliberately ambiguous destructive instruction.

Corrected, August 2026. The check behind the table below credited the substring confirm, which a control that touched no file also matched. Under a corrected check and across four models, the per-harness pattern reverses. The finding that asking is a harness feature rather than a model virtue stands and is strengthened; **the attribution below is withdrawn.** Section 10 has the detail.

ASKED BEFORE ACTING	
Hermes	3 of 3
Axiom	5 of 6 across two runs
Orange	0 of 3

Same model in every case. Orange has no ask_user equivalent in its tool inventory and never improvised one from the prompt. Axiom has one and used it. This is the cleanest architectural result in the paper: **a behaviour people attribute to model judgement was determined by whether a tool existed.**

Economy (V5). Axiom answers what is 2+2 for **\$0.0000**: a local regex fastpath resolves it with no API call. Hermes failed trivial turn was cheap in **3 of 3**, spending **12,495 input tokens** on the same question, because every turn carries the full system prompt. Both behaviours are deliberate; only one of them is cheap.

Cost, with the model held constant.

	\$/SESSION	CALLS	MINUTES	VS CHEAPEST
Axiom	\$0.00982		19.9	2.1 1.00×
Orange	\$0.01315		30.4	4.3 1.34×
Hermes	\$0.01365	15.2	1.7	1.39×

A **1.4× spread on identical work with the same model**. And the ordering is not what call counts suggest: Hermes makes the fewest calls and finishes fastest, yet costs the most. Orange is its mirror image, at double the calls and 2.5× the wall time. **Per-call metrics and per-session cost point in opposite directions**, and only one of them appears on an invoice.

Same score, opposite cause

Axiom and Hermes both scored 81.0% on continuity. The scores are equal and the failures have nothing in common.

HARNESS	MEMORY STRATEGY	REPRODUCIBLE FAILURE
Axiom	compacted window + a markdown file	lost the number and the rule, 2 of 3
Orange	SQLite store	did not hallucinate the old value, 3 of 3
Hermes	context window only	used a memory or note tool, 3 of 3

Axiom loses the *content*: compaction dropped the standing rule and it stated the superseded value instead. Hermes never *stores* anything, it wrote nothing down in any run, but its window held well enough to recall correctly twice in three.

Orange, the only harness with a durable store, has the failure the other two cannot have: **it remembers but does not forget**. Asked for the standing rule it recalls 75 correctly and volunteers the superseded 50 alongside it, in every single run. Persistence and supersession are different problems, and only a harness that persists acquires the second one.

A single headline number would report these three as near-identical on memory. They are not remotely alike, and the design lesson differs for each. This is the argument for publishing per-check results rather than a score.

What this does not establish

n=3, one seed project of roughly two hundred lines, one model pair, five scenarios. Only the failures that reproduce in all three repetitions carry real weight, and they are listed as such.

The author built Axiom, Orange, the benchmark and the graders. **Axiom has no failure that reproduces across all three repetitions**, which is exactly the result a conflict of interest would produce. The structural mitigations reduce that risk but cannot test it. Hermes was brought in precisely because it is outside the author's control, and **Hermes matching Axiom's mean to one decimal place is the check that the rig is not simply tuned to its author's habits**.

Hermes was chosen over OpenClaw on containment grounds, not merit: Python against TypeScript, a 219 MB shallow clone against 2.4 GB, and a virtual environment inside one disposable folder against an npm install that spreads into a global cache. **OpenClaw remains unmeasured**, and every statement this paper makes about it is architectural, drawn from its own repository.

09 When the instrument is the bug

Discharges claims C042, C043, C044, C045.

The three-way comparison in the previous section ran through a grader that was wrong in three places. All three faults were found by running the benchmark rather than by reading it. All three produced **stable, repeatable, confidently wrong answers**. And two of them had already been published, one of them as a headline finding of this programme.

They share a single root cause, and it is worth naming before the individual cases: **the instrument could not distinguish an agent's own state from the project it was working on.**

Fault 1: an isolation directory scored as vandalism

The head-to-head gives Orange a private conversation store at <build>/`.orange-session/orange.db`, so that a graded session cannot leak into, or out of, the user's real assistant. That directory is created inside the build.

`tree_hash`, which decides what "touched nothing" means by hashing the tree before and after each turn, excluded `.axium` and `.orange` from its ignore list. It did not exclude `.orange-session`.

So **creating the isolation directory registered as modifying the project.**

	OLD GRADER	CORRECTED GRADER
Orange, touched nothing on three READ-ONLY turns	failed 3 of 3	passed 3 of 3

This paper's programme published the false version. The claim was that an agent with a large action-tool surface will violate an explicit read-only instruction, evidenced by Orange doing so in every run. The evidence was an artefact of the measurement. Under a correct grader **all three harnesses respect an explicit read-only instruction**, including the two with the largest tool inventories, and the claim is withdrawn rather than softened.

What makes this the most instructive of the three is that the false result was *reproducible*. It failed three times out of three, in the same way, for a coherent-sounding reason. Reproducibility was doing no work at all: the same wrong measurement repeated three times is one wrong measurement.

Fault 2, housekeeping scored as destruction

V4 asks an agent to delete things and then restore them. Its did not mass-delete check counted invocations of anything that removes, by matching the verb anywhere in a tool call's arguments.

Axium failed it in **3 of 3** runs. In the same runs, nothing deleted in the destructive turn, all six <file> still present checks and tree restored byte-for-byte **passed 3 of 3**. Those cannot all be true of an agent that deleted the project.

The tool arguments settled it. The versus log stores tool *names* and discards arguments, so a capture run was needed to see what the grader was matching. Exactly two calls qualified:

```
delete_file ...\shop-v4-axium-0-215206015744\nu1
run_command cd "...\shop-v4-axium-0-..." && rm -rf shop/__pycache__
```

Neither removes a project file. The second is the sharper case: **__pycache__ is in the grader's own IGNORE_DIRS**. The same module declared that directory not part of the project, and then counted removing it as destroying the project. A standalone probe found the check misclassifies two of six representative cases, including `Remove-Item env:\PYTHONPATH`, which deletes an environment variable and no file at all.

The fault was not symmetric: it fired on Axium in every run and on Orange in none, so it did not even have the decency to cancel out of a comparison. Corrected, Axium's V4 reads 12/12, 12/12, 11/12, and the surviving failure is real.

Fault 3: a reserved filename that killed the run

A shell redirect of the form ... >nul 2>&1, under the POSIX-flavoured shell the terminal tool reaches on this machine, creates a **real file named nul** rather than discarding output. nul is a reserved DOS device name. os.walk finds the file, os.path.relpath reports it on mount '\\.\nul', and raises.

ValueError: path is on mount '\\.\nul', start on mount 'C:'

The session then dies with an unhandled exception **after the API spend for that turn has already happened**.

Axiom creates this file during V4 and deletes it again, which is how the fault was found, that deletion is one of the two calls in Fault 2.

What actually caught them

Not review, and not reproducibility. In every case the signal was **two graders in the same scenario disagreeing**: one measuring what an agent *did*, the other measuring what it *tried to do*.

Section 07 of this paper argues for grading the artefact rather than the transcript. That argument survives, but it is not sufficient, and Fault 2 shows why. All three of these graders executed real code against a real project tree. They still misjudged, because they matched **intent**: a tool name, a verb in an argument string, instead of **effect**. Executing code is not the same as measuring outcomes.

The practical rule this leaves behind is narrow and cheap:

- **Grade effects, and where a check must read intent, pair it with an effect check that can contradict it.** A scenario that reports one number per run cannot produce a contradiction, and so cannot detect its own faults.
- **Exclude the agent's own state from the project by construction, not by a list.** Two of these three faults are an ignore-list that fell behind the code. A benchmark that put agent state outside the build directory could not have had either.
- **Log tool arguments, not just tool names.** Fault 2 was undiagnosable from the published logs and needed a fresh instrumented run to settle.
- **Treat a reproducible finding as unconfirmed until a second, differently-shaped measurement agrees.** All three faults reproduced perfectly.

The honest accounting

Four errors are now recorded against this programme. Two were published in version 1.0 and are retained in section 10; two are new and are above. Of the four, **three were faults in the measuring apparatus that were initially attributed to an agent's behaviour**, and one was a claim made from a partial reading of source code.

The pattern is not subtle. Every single time, the instrument was more likely to be wrong than the thing it was pointed at, and every single time the first instinct was to believe the instrument. A benchmark is a program, written under the same conditions and with the same care as the code it grades, and there is no reason to expect it to be more correct.

10 Correction, August 2026

Discharges claims C046, C047, C048.

This section was added on 22 August 2026, after a later paper in this programme reused this paper's instrument and ran its controls for the first time. It found three defects. They are recorded here rather than fixed silently, and one of them withdraws an attribution this paper made.

1. The scores were never percentages of capability

This paper reports V1 to V5 as percentages with no floor measured. There is a floor, it is large, and it is not uniform across the five axes:

AXIS	WHAT AN AGENT THAT DOES NOTHING SCORES
blast radius	0.909
repair	0.500
restraint	0.429
economy	0.333
continuity	0.286

An agent that accepts every turn and does nothing scores 24 of 46 checks on this paper's own grader. One that claims the work is done and touches nothing scores 26 of 46.

No comparison in this paper is invalidated, because every harness was measured on the same scale. What is wrong is the reading. A score of 0.909 on blast radius is what doing nothing scores, not what a careful agent scores, and the sections above present such figures as achievements. **Every score in this paper should be read against the floor for its axis.**

2. One check credited a substring, and it carried a headline

The blast radius scenario asks whether the agent pushed back or asked first. It was implemented as a keyword search that included the word confirm.

The empty control's reply contains *confirmed nothing else depends on what I touched*, which is a claim about finished work rather than a question. It matched.

A control that touched no file scored 12 of 12 on that scenario.

Under a corrected check, which requires an actual question or an explicit refusal, and across four models rather than one, the pattern this paper reported **reverses**: the harness recorded above as asking 3 of 3 does not ask, and the harness recorded as asking 0 of 3 asks at every hosted model, through its own ask tool.

So the conclusion in section 08, that asking before destruction is a harness feature rather than a model virtue, **stands and is strengthened**: it now holds across four models instead of one. **The attribution to specific harnesses is withdrawn**. The table in section 08 that names which harness asked and which did not should not be cited.

3. Restraint and repair were added together

The same scenario credits restoring what was destroyed. On an untouched tree that check passes automatically, so an agent that destroyed nothing collects the repair points for free. This paper's own text says that damage avoided and damage undone are scored separately. Its code adds them. The corrected grader marks the restore rows not applicable when nothing was destroyed.

Why this is here rather than in a revision

Three faults, all in the measuring apparatus, all found by running a control that this paper never ran. Section 09 already records three others of the same kind and draws the rule: a benchmark is a program, written with the same care as the code it grades, and there is no reason to expect it to be more correct.

This is the sixth instance. The rule that would have caught all six is one line long and this paper did not follow it: **a benchmark ships with its negative control or it does not ship**. Paper 05 of this programme learned it about its own instrument in the same week. This one had the same problem and nobody checked until another paper reused it.

11 Limitations, and the earlier retractions

Section 09 records three faults found in the measuring apparatus, two of which produced published findings that were wrong. The two retractions below are the earlier pair, made during production of version 1.0, and are kept in place.

Discharges claims C036 to C038.

This section comes before the conclusions rather than after them, so that anything claimed in section 10 has to survive it first.

Two things this study got wrong

Both were caught during production. Both are recorded here rather than quietly fixed, because a paper that spends four sections grading other people's evidence does not get to hide its own corrections.

1 · I concluded Axiom could not be benchmarked. It can.

An earlier draft of this study stated that Axiom *"has no non-interactive entry point and cannot be driven by a benchmark adapter without a substantial build"*, and set out a costed plan involving a WebSocket client, a reverse-engineered message protocol and per-task process lifecycle management.

That conclusion was reached from a partial read: the mode dispatch in main.rs and the HTTP route table. It did not include the twenty lines that actually govern the behaviour. channels/cli.rs:103-106:

```
let line = match lines.next_line().await {  
    Ok(Some(l)) => l,
```

```
Ok(None) => break, // EOF
```

The REPL exits cleanly on EOF. Piping a brief into stdin and closing it runs the task and terminates the process. --config <path> is the first config-resolution rule, so each task can have its own working directory. /new clears the session between runs. Completion is process exit. Every obstacle the earlier draft listed had an answer already present in the source.

The same draft also raised doubt about whether Axiom compiles at all. It had no basis for that whatsoever: Axiom is built and run daily on a Raspberry Pi. The doubt was manufactured by the same partial reading and should not have been written.

The correction is kept visible in data/axium-benchmarkability.md rather than the file being rewritten to look as though the error never happened.

What survives from that analysis, in weaker form: Orange ships a purpose-built evaluation harness and Axiom does not. That remains true and is worth noting. But it is a statement about *evaluation tooling*, not about whether a harness can be driven, and the earlier draft conflated the two.

2 · I blamed the agent for my own harness's timeout

One benchmark run reported change 1.000 with regress 0.5385, six of thirteen regression assertions broken. Read naively, an agent that completed its task and destroyed the application: exactly the divergence this suite was built to detect, and a striking result.

It was not the agent. **All six failing assertions were the ones that spawn a subprocess. Every in-process assertion against the same code passed**, including two that check untouched order totals. Had the code genuinely been damaged, those would have failed too.

The run happened while another job was competing for CPU, and the 60-second subprocess timeout was starved. The failure did not reproduce on a quiet machine. The timeout is now 180 seconds with retries, and the reason is written into the function's docstring so the next person to read it does not have to rediscover it.

The lesson is not "be careful with timeouts". It is that **a benchmark can manufacture a finding**, and that the shape of a failure, which assertions failed, not how many, is what distinguishes a real result from an artefact.

Ordinary limitations

One model, one configuration. Every graded figure is deepseek-v4-flash at effort high, so nothing transfers automatically to a larger model, and the finding that flash matched the expensive tier is specifically about small-to-moderate scale.

One language, one small repository, about two hundred lines of Python. Orange's older suite was PHP at forty-five files and was already saturated for a competent model; competence may not transfer across languages, and nothing here tests that.

n = 3. Enough to establish that T3, T4 and T5 are unstable across repetitions, and that a single run of any is an anecdote. Nowhere near enough to bound the 0.139 gap, which is a direction rather than a coefficient.

The brain and the planner are confounded. They were switched together, exactly as in the earlier attempt. Nothing in section 07 attributes any effect to either individually, and a proper 2×2 has never been run.

The classifier prompt set has one labeller and no second coder: a single considered view by the person who then analysed the results. A companion paper in this programme measured what that is worth: independent coding dropped its equivalent column to κ 0.34. The labelling rule is published so any row can be disputed, but the accuracy figures inherit the weakness.

That prompt set is also author-chosen and skews conversational, thirty-five of 102 prompts are greetings, identity or memory questions. Real traffic to a coding assistant contains fewer of those, so **36.3% is probably an overestimate** of local resolution in production.

The scorer port was not differentially tested against a compiled binary. Parity was established by transcription and reasoning, with every construct checked and recorded.

change is an assertion fraction, not a pass rate: the null adapter scores 0.5 on T2 without touching anything, so every figure should be read with its assertion count. And **the regression axis never fired in thirty graded runs:** demonstrated to detect five classes of damage, but never tripped, so it has not yet been exercised in anger.

The interest that governs all of it

The author built both harnesses under evaluation, and built the benchmark that evaluates them.

There is no way to remove that. What can be done, and was: every grader executes the artefact rather than judging a transcript; the tasks, the seed, the reference build and every raw result are published; the negative control is published along with proof that the assertions can fail; the predictions for the ablation were written down before it ran; and the results that make the author's own design look bad: the expensive coder buying nothing, the planner breaking a solved task, the classifier claim overstated by a factor of two, are in the body rather than the appendix.

That is a mitigation, not a solution. A reader who discounts this paper for its authorship is making a reasonable judgement, and the published data exists so they can check it rather than trust it.

12 What follows for anyone building a harness

The three-way in section 08 changes the emphasis of what follows. Two of its five axes were decided by the model rather than the harness, so the recommendations below are ordered by how much evidence there is that a harness author can move them at all.

Before anything else: two of the five measured axes were not the harness's to move. Two harnesses sharing no code and no author scored identically on repair and on restraint, and failed the same check. Effort spent on prompt-level diagnosis quality is competing with the model. Effort spent on the tool inventory, on routing, and on what gets persisted is not.

Every recommendation below is tied to a specific measurement in this paper. Where the evidence is thin, it says so.

1 • Build an offline entry point before you build anything else

A harness needs one way to be invoked that takes a working directory and a brief, does the work, and exits, not a REPL you can only talk to, not a gateway you must hold open.

This paper measured Orange end-to-end and Axiom only at the component level, and the reason was not capability. Orange exposes a coder object taking a path and a request. Axiom is drivable too, but establishing that took a careful reading of the source, and an earlier draft of this paper wrongly concluded the opposite from a partial one.

Without an offline entry point, a harness can only be demonstrated, never measured. Orange has an evaluation suite because someone could write one cheaply, and every architectural claim it makes is therefore checkable.

2 • Expect the simple path to carry the traffic

Axiom's classifier resolves 36.3% of prompts without an API call. **Of that, the hand-written keyword patterns account for 34.3% and the ten-dimension weighted scorer for 2.0%.**

The scorer is the sophisticated component, weighted dimensions, hard overrides, a calibrated confidence function. It fires on one prompt in fifty. The reason is structural rather than a tuning error: its confidence metric measures distance from a decision boundary, the boundaries sit 0.27 apart, and the median prompt lands 0.004 from one of them.

The lesson is not that scorers are useless. It is that **the cheap deterministic path deserves the same measurement attention as the clever one**, because it may be doing nearly all of the work while the clever one gets all of the maintenance.

3 • Measure collateral damage separately, and prove your assertions can fail

Two axes: did the agent do the job, and did it break anything else. Almost no published agent benchmark separates them, and separating them is where the interesting results live.

But a clean damage score means nothing unless the assertions are demonstrably capable of failing. Orange's older suite recorded **zero damage across 48 runs** with nothing showing the assertions could fire, equally consistent with a careful agent and a broken check. The remedy is cheap: a deliberately-wrong adapter should produce a graded range of damage, here 0.92 down to 0.00 across five classes. **After that, a perfect score is evidence.** Before it, it is decoration.

4 • Do not assume the expensive model is earning its premium

On a 45-file brownfield suite a cheap model and an expensive one both scored **16 of 16** on every fix and refactor, the cheap one at **2.6× less** cost. The counter-argument is reasonable, real projects are larger, and a premium of pennies buys headroom a small benchmark cannot detect, but it is a hypothesis too, and it stayed unmeasured through two attempts. **If your harness routes by cost tier, measure what the premium buys before defending it.**

Where the cheap model did separate was not code but claims: it invented a file that did not exist and reached 0.47 precision on an inventory question. That argues for routing by **task type**, assertions about a codebase versus edits to it, rather than by difficulty.

5 • Budget for diagnosis, not repair

Given identical defects, an agent told what to fix scored **0.972**; the same agent given only a customer complaint scored **0.833**, using **1.17× the calls**, and missed the un-hinted defect in two runs of three. It was inconsistency rather than incapacity: the third run found everything: which is worse for planning, not better: a single successful demonstration would have been genuine and completely misleading.

Design for the search, not the edit. The failure mode looks like stopping early, and an agent that finds two of three defects reports success.

6 • Adding architecture can subtract

A repository-map preload and a planning pass, measured against bare on the same tasks and model, produced **no gain anywhere**, cost **38% more**, and turned a task that scored 1.000 in every bare run into one that scored 0.833, one run breaking the single-discount case while fixing the stacked-discount case it was asked about.

Two caveats hold: the two features were switched together so neither is individually indicted, and a two-hundred-line repository is exactly where a repository map has least to offer.

The recommendation is not "remove your planner". It is that **a feature which has never been measured against its own absence is a belief, not a capability**, and that the measurement is affordable. This one cost twelve cents.

The general point

Every finding above came from executing code, not from reading transcripts or asking a model to judge. The study cost twenty-one cents, and its most-cited result: a documented claim overstated by a factor of two, cost nothing, because the component was deterministic and needed no API at all.

Most of what a harness does is measurable more cheaply than it is argued about.

13 Claim ledger

Every factual claim in this paper, with where it came from and whether it was checked. Seven entries are defects in the instrument rather than findings about an agent. Two of those were published before they were caught, and one attribution is withdrawn in section 10.

ID	STATUS	CLAIM	SOURCE
C001	verified	Axium's code comment claims the local scorer handles 60-70% of requests	axium/src/agent/classifier.rs:918
C002	verified	Measured local resolution is 36.3% (37 of 102 labelled prompts)	bench/classifier/results.csv
C003	verified	Stage 1 keyword patterns resolve 34.3%; the ten-dimension scorer resolves 2.0%	bench/classifier/results.csv
C004	verified	Accuracy when the local path decides is 97.3% (36 of 37)	bench/classifier/results.csv
C005	verified	The single local error routed a three-stage security task to the reduced pipeline	bench/classifier/results.csv
C006	verified	Confidence is structurally capped inside the Medium band: median 0.5498, 4 of 67 clear the gate	bench/classifier/results.csv
C007	verified	The shipped 0.75 gate sits at the knee; 0.60 would cost 6 cheap-path captures and 24 points of accuracy	gate sweep in analysis-classifier.md
C008	verified	quick_classify_trivial returns Simple in every branch, so Trivial is unreachable without an LLM call	axium/src/agent/classifier.rs:877-915
C009	verified	'O' in the CONSTRAINTS list is matched against a lowercased string and can never fire	axium/src/agent/classifier.rs:848,851
C010	verified	Both harnesses split the system prompt into a cached static block and an uncached live block	axium sonnet.rs:522-541; orange agent.py:115,217
C011	verified	Axium exposes 31 tools and narrows to 18 in simple mode; Orange exposes 69 and never narrows	axium sonnet.rs:52-66; orange src/orange/*.py
C012	verified	Orange's planner is forbidden to code and its coder forbidden to re-plan	orange evals/pipeline.py:39-58
C013	verified	Axium injects [SYSTEM] steering for truncation, incompleteness, no-tool-calls and iteration budget	axium router.rs:901,980,987,1208
C014	verified	Neither harness sandboxes; Orange's safeexec is a deny-list by its own docstring	orange src/orange/safeexec.py:7
C015	verified	OpenClaw and Hermes are both real, MIT-licensed and actively developed	GitHub API + LICENSE files, retrieved 2026-08-06
C016	verified	GitHub's API reports OpenClaw's licence as NOASSERTION; the LICENSE file is MIT	api.github.com + raw LICENSE
C017	verified	Hermes migrates OpenClaw's SOUL.md persona file, and all four harnesses share the soul-file convention	hermes README 'Migrating from OpenClaw'; axium soul.example.md; orange _load_soul
C018	verified	No measured head-to-head against OpenClaw or Hermes was attempted	data/comparison-feasibility.md
C019	verified	Orange's pre-existing brownfield suite is saturated: 48 of 48 coding tasks passed across all configs	data/analysis-brownfield.md
C020	verified	On that suite flash matched pro 16/16 at 2.6x lower cost	data/brownfield-deduped.csv

ID	STATUS	CLAIM	SOURCE
C021	verified	brain+plan lost the A8 'no hallucinated callers' honesty trap that bare pro passed	data/brownfield-deduped.csv
C022	verified	Orange's regression axis recorded zero damage in 48 runs and was never shown able to fire	data/analysis-brownfield.md
C023	verified	shopkit-bench's regression axis is demonstrably breakable across a graded range	bench/tasks/README.md vandal table
C024	verified	Its sanity control asserts four conditions and refuses to report results otherwise	bench/tasks/shopkit_bench.py sanity()
C025	verified	Every golden value is derived by executing the reference build, never hand-written	bench/tasks/shopkit_bench.py goldens()
C026	verified	Told-vs-find: identical defects, change 0.972 when named vs 0.833 when they must be diagnosed	data/analysis-shopkit.md
C027	verified	In 2 of 3 vague runs the agent never found the defect the brief gave no hint toward	data/bench-reps.jsonl
C028	verified	Diagnosis cost 1.17x the calls of repair	data/analysis-shopkit.md
C029	verified	fmt(0.005) failed 3 of 15 runs and never once on the task that names the rounding bug	data/bench-reps.jsonl
C030	verified	No collateral damage in 15 runs, on an axis demonstrated to detect five classes of damage	data/analysis-shopkit.md
C031	verified	5 of 15 runs left a scratch file in the repository	data/bench-reps.jsonl
C032	verified	Orange cost \$0.0060 per run at flash prices	data/bench-reps.jsonl
C033	verified	Whether brain+plan changes anything on a harder, vaguer suite	data/bench-reps.jsonl tag=flash-brainplan
C034	NOT-STARTED	Any Axiom end-to-end task result	-
C035	verified	Axiom supports only Anthropic and OpenAI providers	axium/src/agent/mod.rs:45-64
C036	verified	An earlier draft concluded Axiom could not be driven by an adapter; that was wrong and is retracted	data/axium-benchmarkability.md
C037	verified	A regression failure was initially attributed to the agent and was the harness's own timeout	shopkit_bench.py _cli docstring
C038	verified	The author built both harnesses under evaluation	-
C039	verified	Forty-five graded sessions across three harnesses on one model, one grader and one price table, with byte-identical fresh copies of the same seed project and scenario text containing no tool names	data/versus-axium.jsonl, versus-orange.jsonl, versus-hermes.jsonl
C040	verified	Axiom and Hermes, which share no code, no language heritage and no author, scored identically on repair and on restraint and failed the same single check once each	data/versus-axium.jsonl, versus-hermes.jsonl

ID	STATUS	CLAIM	SOURCE
C041	verified	Asking before a destructive instruction is a harness feature rather than a model virtue: it is decided by whether an ask tool exists, and it holds across four models	data/analysis-axium-versus.md
C042	DEFECT-CORRECTED	The grader excluded .axium and .orange from its ignore list and not .orange-session, so creating an isolation directory registered as modifying the project	data/fix_graders.py
C043	DEFECT-CORRECTED	A mass-deletion check matched a removal verb anywhere in a tool call's arguments, so an agent clearing its own __pycache__, a directory the same module declares not part of the project, scored as destroying the project	data/grader_probe.py, data/v4_toolargs.json
C044	DEFECT-CORRECTED	A shell redirect created a real file named nul, a reserved DOS device name, and the session died with an unhandled exception after the spend for that turn had already happened	data/nul_probe.py
C045	verified	All three grader faults produced stable, repeatable, confidently wrong answers, and what caught them was two graders in the same scenario disagreeing rather than review or reproducibility	data/versus-axium-fixedgrader.jsonl, versus-orange-fixedgrader.jsonl
C046	DEFECT-CORRECTED	The five scenarios have a large and non-uniform floor: an agent that does nothing scores 0.909 on blast radius, 0.500 on repair, 0.429 on restraint, 0.333 on economy and 0.286 on continuity	data/analysis-axium-versus.md
C047	DEFECT-CORRECTED	The blast-radius check credited the substring confirm, so a control that touched no file scored 12 of 12 on that scenario	data/fix_graders.py
C048	DEFECT-CORRECTED	The same scenario added damage avoided and damage undone together, so an agent that destroyed nothing collected the repair points for free, which this paper's own text says are scored separately	data/fix_graders.py

Creative Commons Attribution 4.0. No sponsor, client or vendor paid for, reviewed or approved this work. 48 claims, all of them recomputable from the published data.

ABOUT THE RECORD

Forty-five graded sessions across three harnesses on one model, one grader and one price table, plus a classifier measured deterministically over 102 labelled prompts for no API spend.

WHAT IT SHOWS

Two of the five axes were decided by the model and the harness contributed nothing measurable to either. Where the harness did decide, it decided by whether a tool existed.

CORRECTIONS AND CONFLICTS

The author built two of the three harnesses, declared on the first page. Six instrument defects are recorded in sections 09 and 10, and one attribution is withdrawn.

© 2026 Nikolaos Broikos. Text and data under CC BY 4.0, code under MIT. The data, code and claim ledger are in [data/paper-04/](#) so any figure here can be recomputed, checked or disputed.