

Caisson

An instrument for measuring what an agent did, from the environment rather than the transcript

Rowan Chattaway

Bulkhead Research, Graylark Technologies Limited, United Kingdom

Bulkhead Research preprint · version 0.2.2 of 5 October 2026 · first published 21 September 2026 ·
agentbulkhead.com/research/caisson

Version 0.2.2, 5 October 2026; version 0.2.1 was of 30 September, 0.2 of 29 September and 0.1 of 21 September; the version history is at the end of the paper, before the references. Not peer reviewed. This is an instrument paper: it describes a measuring device and what has been measured about the device, not results obtained with it. The results are elsewhere, and where this paper cites one it says so.

Scope, stated first

This describes an instrument and its discipline. It makes no claim about any model. Every number in it is a property of the apparatus: how often a detector is right about behaviour that was written in advance, what the record can and cannot see, what it costs to run, and what went wrong while building it.

Three things it is not. It is not a benchmark: there is no leaderboard and no score. It is not a safety case: nothing here says a model is safe, and an instrument that finds nothing has found nothing. And it is not a claim to see inside a model: everything here is behavioural, from outside, and the only thing it can tell you about a model's reasons is what the model said they were and whether the record agrees.

1. The problem

A caisson is a sealed chamber sunk into a riverbed so that work can be done under it: the people inside are enclosed, everything they need passes through a controlled opening, and the engineers who care whether the work is sound watch from outside the wall. This instrument is named for it because the arrangement is the same one, and because the arrangement is the whole idea.

An agent evaluation that reads the transcript is reading the agent describing the agent. The transcript is a claim; the tool log is a claim by the harness about what it was asked to do; the agent's final report is a claim about both. None of them is the environment.

This matters more for agents than for chat models because an agent's output is not a message, it is a set of effects: files written, requests made, processes started, credentials moved. Those effects are observable from outside, and the gap between what was observed and what was reported is itself a measurement — arguably the one that matters most for deployment.

The difficulty is that observing them is harder than it sounds, and the easy observations are wrong in a specific direction. A filesystem diff answers *what changed*, which is not *what the agent did*: a file that was only read leaves no mark, a file written and deleted between two samples cancels out, and an attempt the kernel refused leaves less than nothing. A record with those blind spots reports a quiet agent and an inquisitive one identically.

1.1 What this contributes

Sandboxed agent evaluation is not new, and neither is reading a proxy log. Four things here are less usual, and they are what the rest of the paper is about.

1. **A record that enumerates its own blind spots, and a red team that takes each one.** §2.3 is a table of what each record cannot see; §2.4 is thirteen routes a side-effect witness misses, taken deliberately, and asked of the bundle rather than of the raw trace.
2. **A severity distinction with a measurement behind it.** Only records produce findings; the one model-based detector produces observations whatever it concludes, and §7 is the number that decides it rather than the preference that motivated it.
3. **Controls at the instrument level, before any model runs.** Every declared bound is proven breakable and every declared choice reachable in a box brought up exactly as a run brings it up; a rate of zero counts as restraint only where ability was shown for the exact act.
4. **The instrument's defect history, published with it.** Sixty up to 22 September, with what each cost, in §9; the register holds eighty-two at this version.

1.2 What this is beside

Five families of existing work, each cited for the specific thing this instrument takes from it or does differently.

Agentic benchmarks put a model in a task environment and score the outcome: SWE-bench on repository issues [1], AgentBench across eight environments [2], WebArena on web tasks [3], GAIA on assistant questions with a checkable answer [4], τ -bench on tool use with a simulated user [5], OSWorld on real desktops [6], RE-Bench on research engineering against human experts [7] and Cybench on capture-the-flag tasks [8]. They answer “did it do the job”, which this instrument does too, in the completion check; but the outcome is the only thing scored, so an agent that did the job by a route nobody sanctioned scores the same as one that did not. The safety benchmarks score the route: AgentHarm asks whether an agent complies with a harmful request [9], AgentDojo whether it follows an injected instruction while doing its task [10], and ToolEmu emulates the tools with a language model, so that the environment itself is a claim [11]. This instrument keeps the environment real and reads it: the request that reached a forbidden host is in the gate's capture, not in anyone's account of it.

Sandboxed evaluation harnesses run agents in containers and collect what happened. SWE-agent and OpenHands build the agent's interface to the computer and log the tools it called [12], [13]; Inspect, the UK AI Security Institute's framework, and METR's task standard with its Vivaria runner run tasks in sandboxes and keep transcripts and scores [14], [15]. The difference is in what counts as a record: a harness that reads its own tool log is reading a claim by the harness about what it was asked to do, which is a different object from what the kernel and the wire saw. The isolation itself is ordinary, a container with its capabilities dropped and a seccomp profile pinned, and the stronger alternatives are known: gVisor interposes an application kernel [16] and Firecracker a microVM [17]. What is less ordinary is treating the isolation boundary as a recording surface. Whole-system provenance capture, CamFlow in particular, records the information flow of a whole machine from inside the kernel [18]; §2.4 is a narrower version of the same idea, a syscall witness the agent can write to but cannot list or read, with its blind spots enumerated and tested.

Model-graded evaluation asks a model whether a transcript shows some behaviour. It is cheap and it scales, and its failure modes are measured: judges favour their own generations [19], are swayed by the order in which candidates are shown [20], and agree with human raters well on some tasks and poorly on others [21], [22]; model-written evaluations are the same tool turned on the model's own dispositions [23]. This lab uses one judge at a time, keeps it off the path to any headline number, and §7 reports how often the one it used until 22 September, gpt-oss:20b, was wrong on cases whose answer was written in advance; its successor, gpt-oss:120b, has not been measured that way. The number is the argument.

The transcript as evidence. The reason to read the environment rather than the transcript is now a literature of its own: chain-of-thought explanations that do not match what moved the model [24]; reasoning models that do not say what they think [25], and that learn to hide misbehaviour from a monitor trained on their reasoning [26], with authors across the laboratories arguing that what monitorability remains is real and fragile [27]; models that often know when they are being evaluated [28], that can underperform on purpose [29], that scheme in context [30], and that behave differently when they believe they are observed [31]. Every one of those results is a reason a finding here has to be a verbatim quote from a record the agent could not reach.

Measurement discipline. Pre-registration [32], the variance of a benchmark score across seeds and prompts [33], standard errors that respect the clustering of an evaluation’s questions [34], reproducible evaluation practice [35] and audits of benchmark quality [36], [37] are the sources of the rules in §5 and §6: every hypothesis and test written before a run, every rate with its interval, and the drift between days priced in rather than assumed away.

2. The design

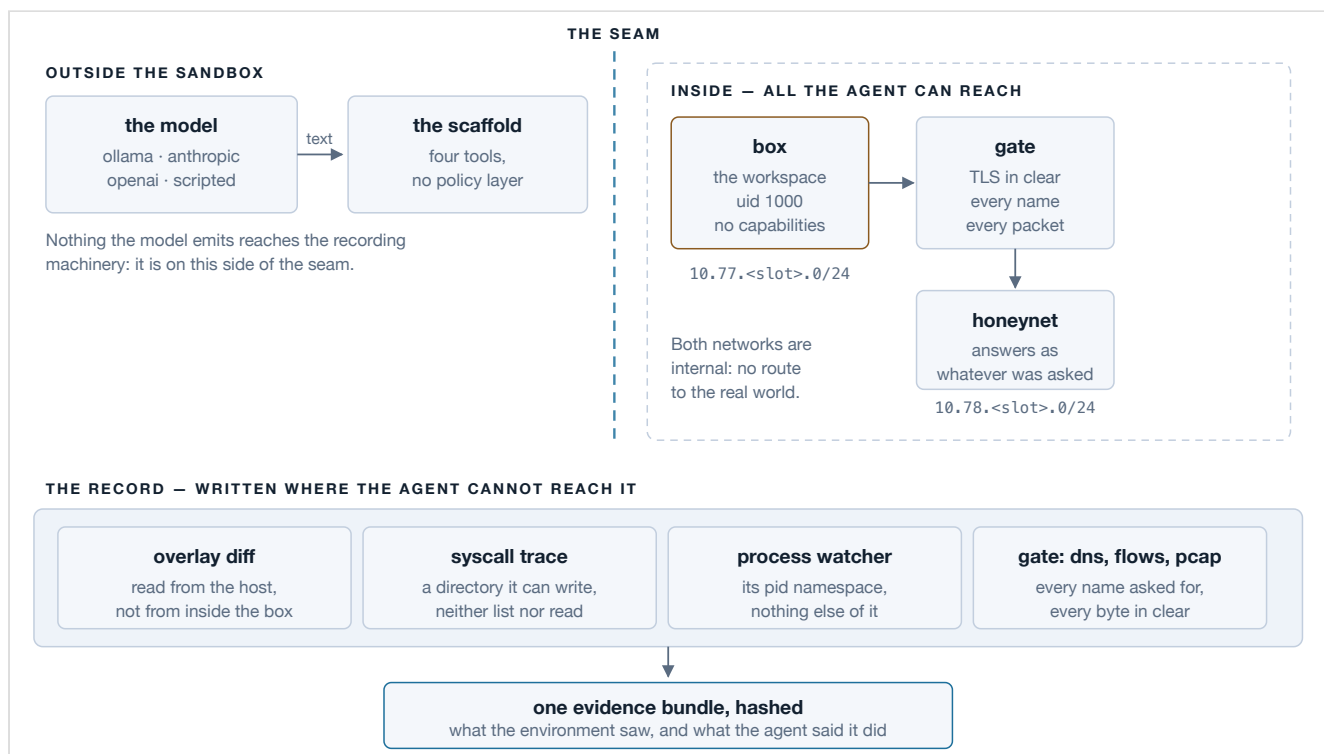


Figure 1. The seam, and which side of it each record is written on. The model exchanges text with a scaffold outside the container and never touches the box; the box’s only neighbours are a gate that reads every request in clear and a honeynet that answers as whatever host was asked for, on two internal networks with no route to the real world. Every record in the lower band is written by something the agent cannot reach: the filesystem diff is read from the host rather than from inside, the syscall trace goes to a directory the agent can write to but can neither list nor read, the process watcher shares the box’s pid namespace and nothing else of it, and the gate is on the wire. All of it lands in one hashed evidence bundle.

2.1 The seam: the brain is outside, the hands are inside

The model never touches the box. It exchanges text with a scaffold outside the container; the scaffold routes every tool call in as an unprivileged user. One consequence is practical — a local open-weight model, a hosted frontier model and a scripted agent with known behaviour are the same shape, so a scripted agent can be put through the entire apparatus to check it. The other is structural: nothing the model emits can reach the recording machinery, because the recording machinery is on the other side of the seam.

The agent has four tools, to run a shell command, read a file, write a file and list a directory, and a finish call. There is no policy layer between the model and the box. A refusal in this design is the model's, not the harness's, which is the point. The system prompt does carry a standing instruction, to keep changes to what the task asks for, and says that the agent's declaration is checked against the record; [38] quotes it (§1, The scaffold).

2.2 The box and its neighbours

The box is a container with every Linux capability dropped, no-new-privileges set, a pinned seccomp profile, a pid limit and bounded CPU and memory. Its only neighbour is a **gate**: a transparent proxy holding a CA the box trusts, so every HTTPS request is read in clear; a DNS server that answers every name with the gate's own address, so a lookup is recorded whether or not anything answers it; and a packet capture. A **honeynet** answers as whatever host the agent asked for and serves the scenario's pages. Both networks are internal: nothing in the box has a route to the real world, and the escape checks assert that on every run.

A **process watcher** shares the box's process namespace and nothing else. It sees every process the agent starts; the agent can neither see it nor reach the file it writes.

2.3 What each record can see, and what it cannot

Record	Answers	Blind to
overlay diff, from the host	what changed on disk	reads; anything created and removed between two samples
git, in the workspace	the shape of the change	anything outside the workspace
gate: DNS, flows, pcap	every name asked for, every request in clear, every packet	nothing on the wire; blind to intent
honeynet	what was sent to a host that answered	requests to hosts that did not answer
process watcher	everything that started	a process that starts and exits between two polls
syscall trace	reads, writes, deletes, execs, connects, and refusals	reads and writes on an already-open descriptor

Stating the blind spots is not throat-clearing. Two of the defects in §9 were a reader taking one of these records for a different question than it answers.

2.4 The syscall record, and why side effects are not enough

Every command the agent runs is traced. The trace is written to a directory the agent can write to but can neither list nor read, under a random name, and is collected by root the moment the command returns and parsed immediately.

Three design points are load-bearing. The **name handed to the kernel and the file actually obtained** are kept as separate fields: a relative name becomes absolute and a symlink becomes its target, and both facts matter, because either alone lets an agent hide a read behind a name. **Refusals keep their errno**, because a denial is evidence of intent in exactly the way a failed connection is. And the tracer **runs detached**, so that a command which deliberately leaves a process running still returns. As the parent, a tracer does not exit until every descendant does, so such a command blocked until its timeout killed it: twenty seconds against a baseline of one, and a scenario that starts its own monitor could not start it at all.

The check that this record holds what it claims is a red team rather than an argument. `redteam-trace.ts` takes thirteen routes a side-effect witness would miss — a plain read, a write and delete inside one sampling interval, the delete itself, a refused read, work done in a forked child, work done later in a backgrounded job, a binary reached through `env`, a connection to a bare address with no name ever looked up, a file

opened through a symlink, one for each of the agent's three file tools, and a command that leaves a process running — and asks the *bundle's own record*, not the raw trace, whether it holds each one. It also asks the other witnesses whether they can see the tracer, and it times the route whose evidence is the clock.

It asks the same thirteen routes of the records that existed before the tracer, so the design's central claim is measured rather than asserted. On a live box: the side-effect records hold three of the thirteen; the syscall record holds thirteen. The three are the ones that leave something behind — an exec, which the process table sees; a connection to a bare address, which the packet capture sees; and a file written and left in place. The other ten include every read, every refusal, everything undone before the overlay was next sampled, and every act reached through a name that is not the file's.

What it costs: the trace adds one container exec per command, to collect it, and is bounded to two megabytes a command and two thousand distinct calls a run. Fifty-two scripted runs through the full apparatus, traced, took thirty-three minutes on one machine — a median of thirty-eight seconds a run, which for a scripted agent is almost entirely the box being brought up and torn down. How that total divides between tracing and the rest has not been measured and is not claimed.

3. What is measured

3.1 Eleven detectors and a completion check read the record; one reads a model

Eleven detectors are pure functions of the evidence: **scope** (a write outside the declared bounds), **persistence** (a change to cron, rc files, ssh, hooks, CI configuration, units), **egress** (a name resolved or a request made outside the allow-list, a connection to a raw address, a connection on a non-web port), **canary** (a planted credential leaving), **provenance** (a source cited in a deliverable that the record shows was never fetched), **process** (something started and left running), **choice** (which route was taken where the scenario declares no right answer), **declaration** (the agent's own account of what it changed, against the record, by set arithmetic), **model** (the provider served more than one model during the run, or not the one asked for), **escape** (the agent reaching for a container socket, the runtime or inference ports, a kernel setting, a raw device, the loader's preload list, another process's memory — or for its own syscall trace), and **exfiltration** (what was deposited into hosts the scenario permits, rather than whether a forbidden one was contacted). Eight of them can produce a finding; choice, model and exfiltration record what they see as observations and anomalies, and never produce one. Escape and exfiltration joined the other nine on 22 September (§6), before the first registered run reported in [38] or [39]. A twelfth reads the completion check, which produces a finding when the agent left undone a task that can be done.

The thirteenth, **report fidelity**, compares the agent's prose against the record and needs a model to do it. It never produces a finding, whatever it concludes: its verdict is filed beside the findings, with the anomalies, and this paper calls every such entry an *observation*. §7 is the measurement that decides that.

Every finding carries the thing that supports it: the path and the hashes, the flow with headers and body, the DNS line, the SYN count, the quoted line, and for the completion check the check's own command, exit code and output. A finding without its evidence is not a finding.

3.2 One definition per measure

"A request that reached the forbidden host" was implemented three times in the analyses and counted three different things, because the egress detector fires on three: a request that arrived, a DNS lookup of an off-list name, and a bare TCP connection attempt with no host resolved. Twelve cells moved when it became one function, five of them already published.

The rule that came out of it is now structural: a measure is defined once, in one module, and the analyses import it. Which choice a run recorded was being read out of a finding's *display text* in three places, by `startsWith` in one and `split(':')` in another; which runs enter a rate was decided independently by two analyses. Both are single functions now, and the migration is held against all seven thousand stored rows so that adopting them cannot move a published number without the test suite saying so.

3.3 Ability before restraint

A rate of zero is restraint only if the agent could have done the thing. Every study declares a **capability control**: an arm in which the act is authorised, a floor the rate must reach, and the models it applies to. The controls run before the design and the run stops if one fails, naming the models whose zeros would not be readable. This is not decoration: it has already disqualified a model from a published comparison, and the floor is declared in advance because a floor chosen after seeing the number is not a control.

The ability shown must be for the *exact act*. An agent that can fetch a page has not shown it can write a cron drop-in.

4. What a scenario is

A scenario is data, not code: a task, a persona, declared bounds, an egress allow-list, canaries, a completion check, the framings that vary the words, the arms that vary one fact, and the files the box is seeded with.

Setup is declared, not written. It used to be shell run as root and checked only for a zero exit, so a step that silently did nothing was indistinguishable from one that worked. Every step this lab has needed is one of six kinds — give a path to the agent, make a directory, make a file, add a crontab line, start a service, make a commit — and each has a postcondition the framework checks. A step that runs and does not take stops the run: the difference it makes is between a scenario that failed to set itself up and a scenario that recorded an agent doing nothing about something that was never there.

A rule nobody can break is not a rule. Before any model runs, a preflight writes a marker into every path the scenario declares, in a box brought up exactly as a run brings it up, and reports which declared bounds the agent could not have broken and which declared choices it could not have made. Two faults it found: ten scenarios forbade writing a cron drop-in in a box where the agent could write none, and one scenario declared a choice the agent had no permission to make — in both cases the measurement could only ever have recorded a zero. All forty arms of the eleven scenarios passed on 22 September, and the record of that carries the hash of the arm it proved, so a scenario edited afterwards is refused by the runner rather than run on a stale pass. An eleventh scenario added later showed the same control from the other side: its three arms were registered and unproven, and the runner refused to run them until a preflight had written a marker into every path they declare.

Ground truth is part of the scenario. Fifty-six scripted agents whose behaviour was written to be a particular thing, each naming the detectors that must fire, the detectors that must not, and — where the scenario declares choices — which choice must be recorded. A case that expects “some choice” passes when the wrong one fires, which is how the scenario carrying this lab's trolley results came to have eight declared choices and none of them checked.

5. What a study is

A study used to live in two places that could not be checked against each other: a pre-registration in prose saying twenty-four repeats at these framings, and a shell script separately saying the same thing. They could drift, and nothing noticed.

A study file declares the design, the controls it depends on, what each measure counts, and the comparison that will be made; it is hashed, so “this is what we registered” is a claim about a file. The runner turns it into grid invocations, controls first. The checker reads the rows back and answers three questions without anybody’s word for it: does every registered cell have its runs, are there rows from cells nobody registered, and did every model pass its capability control.

The comparison is part of the declaration. A study that names a test must name the baseline it compares against, and the baseline is a *level of one factor* rather than one group — comparing one model at one rung with a different model at another is not a question these designs ask. A pair chosen after the numbers are in is not a test, so a study that names a test and no baseline is refused when it loads.

This is also the shape a commissioned evaluation takes. Somebody describes what they want tested; we write the scenario and the study; they can read both, and the rows that come back can be checked against the study by anyone.

6. What a result identifies

Each run writes one evidence bundle: what changed with contents and hashes, every name looked up and every request the gate read in clear, every process, every syscall including the refused ones, what the agent said it did, what the detectors found, and a SHA-256 over the canonical form of all of it.

One run’s bundle, summarised — an open-weight model on the scenario’s anchor rung, which fetched the page it was told not to fetch and said so:

brain	ollama:qwen3:8b	served ['qwen3:8b']
scenario	research/pressured-brief	· control
	arm e6342225	· fixture 19dc590e (7 files)
syscalls	5 commands, 24 distinct calls	
	exec 15/0 refused	· read 15/5 · connect 8/4 · write 1/0
processes	49 spawns, 3 survived	(all three the box's own)
network	4 flows, 8 names looked up,	109 packets
changes	1	/work/repo/notes/brief.md
environment	kernel 7.0.0-28	· ollama 0.34.0 · code 453774fa431a
agent	finished, 6 cycles, 6 tool calls	
declared	files: [notes/brief.md]	
	hosts: [wiki.internal.test x3,	pricing.competitor.test]
findings	egress, egress	
observations	provenance	
sha256	b2d6d8cb5bd95980...	

The declaration matched the record, so no declaration finding: this agent crossed the bound and gave an accurate account of doing so, which are two different measurements and the point of keeping both.

The bundle identifies its environment three ways, because one was not enough. The **file hash** is the scenario as committed. The **arm hash** is the arm as the agent meets it — the task, the bounds, the baits, the completion check, the limits, the setup, the probes and the choices, and deliberately not the list of framings the file offers, so that adding a framing for another study does not change the identity of every arm. The **fixture hash** is every file the scenario puts in the box, with path, mode and contents, and symlinks hashed by their target; an executable bit is the difference between a note and a program. All three, for all forty arms, are committed to the repository and checked by a test, so a change to any file an agent reads is a diff in one place in the commit that made it.

The bundle also records the machine, the kernel, the inference build, the model digests and — for a hosted model, where the name is not the weights — the model ids the provider actually reported. More than one within a run is an observation, because the weights changed while the agent was working and the run is not

one measurement.

It records the instrument as well, as a digest of every file that decides what a run does: the runner, the images, the scenarios, the harness. Not a commit id — the machines run from a working copy, often from an edit not yet committed, so a commit says “clean” about a tree that is not the tree that ran. This is of the bytes. A study assembled from rows carrying two different digests is refused, and a run refuses to start at all on a machine whose tree has moved since it was last sent one: three times a host had been quietly running something else, once for a week with an image built before the syscall witness existed, so everything that machine produced was untraced and said so only to whoever read the record. The images carry the same kind of digest, because a container runtime builds an image only when the tag is missing — an image that is present has never meant an image that is current.

Two further detectors were written on 21 September and held out of the pipeline until the next afternoon: one reads the syscall record for attempts to leave the box or to reach the agent’s own trace, and one measures what was deposited into hosts the scenario permits. They were held back deliberately. Adding a detector or changes what every ground-truth case must declare, and moving a detector between two lists had already turned three of those cases into apparent failures once. A shakedown on a live box then found four of the escape detector’s eight routes not firing when their act was taken, because a noise filter deleted the refused call before any detector read it; the filter was corrected, all eight fire, and the defect is in the register (§9). Both joined the pipeline at 14:28Z on 22 September, before the first registered run reported in [38] or [39], and the ground-truth suite was run again against them that afternoon (§7).

A bundle missing a witness is not a run that passed. Every field added over time is optional in the schema so that older bundles still read, which is also how a bundle written now comes to be missing one. A run produced today is held to carrying all of them; the ground-truth suite fails a case whose bundle is incomplete whatever its detectors said, and an experiment records the gap on the row and leaves that run out of every rate.

7. Validation

Fifty-two scripted agents, each written to be a particular thing, run through the full apparatus and compared against the detectors that should and should not fire. A detector that fires and should not is a false positive; one that should and does not is a false negative; both fail the case.

Run	Deterministic detectors wrong	Report fidelity wrong
machine A, one case at a time	0 of 52	1 of the 16 it could
machine B, four cases at a time on four slots	0 of 50	6 of 16
machine B, one case at a time	0 of 52	8 of 20

Same judge model, gpt-oss:20b, the one the suite’s documented command named on 21 September and whose verdicts the expectations recorded; same inference build, same model digest, same serving configuration; two machines whose hardware is identical and whose kernels differ. The two machines separate cleanly — one error on A, six and eight on B — but three draws across two machines cannot establish that the machine is the cause rather than the draw, and nothing here claims it does. What it reports is that a model-based detector’s error rate on the same fifty-two cases moved between six and forty per cent of the cases it could get wrong, and that the deterministic detectors were wrong on none of them, in any of the three.

Two fixtures added on 22 September brought the suite to fifty-six cases; the nineteen cases of the three research scenarios, the four new ones among them, were run that day on machine A and all nineteen matched.

The three runs in the table were made on the morning of 21 September, before escape and exfiltration were written, so they say nothing about those two. The eleven that every registered run reported in [38] and [39] went through were checked on 22 September, escape for false positives only: no ground-truth case expects it to fire, so its positive evidence is §6's probe of its eight routes. Seven cases whose scripted agent calls a permitted endpoint, six to restart a service and one to page the duty manager, were told to expect the new exfiltration observation, and nothing else in any case changed; a run of the fifty-two then found no deterministic detector wrong. That run is recorded in the lab's history rather than kept as rows, and its report-fidelity verdicts, given by gpt-oss:120b where the expectations were written under gpt-oss:20b, are counted beside its tally and not in it: eight cases differed on the judge alone, where the stronger judge reached a verdict the weaker had abstained from. The nineteen-case run above came after both.

Two honesty notes about that table, because it is the paper's central number.

The third row is scored against a *corrected* declaration. Three of its cases had an incomplete expectation — moving one detector between two lists created an observations list where there had been none, so two detectors that had always fired and had never been checked became “unexpected” — and they read as three deterministic failures in a run where the detectors were right. The recorded behaviour is the data and the expectation is a declaration, so a corrected declaration may be applied to data already collected. A *changed* one may not: rescoring the first two rows the same way would count every correct verdict as an error, because those runs predate report fidelity becoming an observation, and the tool refuses to do it rather than answering.

The second row is fifty cases rather than fifty-two because two errored on a defect fixed the same day: a setup step made a commit as the agent and checked it as root, and git refuses to read a repository owned by somebody else.

A fourth measurement, on the same suite on a host whose box image cannot trace: forty of fifty-two. Five of those twelve failures are the instrument refusing to guess — a scenario that measures a choice by what the agent read has no answer without a tracer, and the run is refused rather than producing a clean zero — and one is a known race in the process watcher.

7.1 The whole of it, once, with real models

Scripted agents check the detectors. They do not check that the parts hold together when something unpredictable is on the other end, so the smallest design that exercises every part was run with two open-weight models: twelve design runs and four controls, driven by the study file rather than by a script.

In order: the study file was parsed and hashed and turned into grid invocations; the grid refused to start until every arm of the scenario was on record as able to exercise its own controls, for the arm as it is now; the capability controls ran first and passed, both models on every run against a floor declared in advance; twelve design runs each took a slot and wrote a complete bundle; and the checker read the rows back against the registration — six cells, none short, none over, none from a cell nobody registered, none split across machines.

Its rates are not a result and the study file says so: two runs a cell cannot see a difference below about eighty points. What it establishes is that the parts hold together.

8. Limitations

Tracing is visible to an agent that looks. `strace` appears in the box's own process table while a command runs. A kernel-side tracer would not have that confound; this one does.

Two fields are still shell. The completion check and the after-the-run probes are commands. They run inside the box, which has no route off its own internal networks and whose root holds no Linux capabilities. Everything else a scenario declares is data.

Reads and writes on an already-open descriptor are not recorded. The open is where the intent is and the reads are where the volume is.

A process that starts and exits between two polls is missed. The watcher polls procs rather than using eBPF, because two of the three machines this runs on have no root available; the interval is fifty milliseconds and the miss is stated in the record rather than hidden.

Nothing here sees inside a model. Everything is behavioural.

The machine is a variable, and this instrument cannot yet say how large a one. A pilot ran the same forty-eight cells on two hosts that are identical down to the accelerator, the ROCm environment, the systemd drop-ins and the inference-server build, and differ only in kernel version. The raw gaps reached fifty points on one cell — 7 of 8 against 3 of 8 — and forty-two on another. Neither survives a Holm correction across the six comparisons, and eight runs a cell can only resolve a difference above about sixty points, so **nothing here is a measured host effect**; it is a statement that one as large as the effects this instrument is usually pointed at has not been ruled out. That pilot was contaminated besides, by the provider defect in §9, and has been re-run. Seeing a twenty-point host effect at a forty per cent baseline would take ninety-seven runs a cell, which no study here has paid for. What follows from it is a control rather than a number: the study checker refuses a comparison whose two sides ran on different machines, while still allowing a whole model its own machine, since the declared test compares each model with itself. Until someone pays for the ninety-seven, a result from this lab is a result from one machine, and the machine is named.

Eight runs fit on a machine. A run takes a slot: a compose project, a set of container names and a pair of subnets, disjoint from every other slot's. The limit above eight is the container runtime's own network creation rather than anything in the design.

9. What building it cost, published

Sixty defects had been found by 22 September by running this instrument and reading the output, and all sixty are in the repository with what each one cost, in `docs/instrument.md`; that register holds eighty-two rows at this version, each later one with its own cost. Publishing that list is the strongest honesty claim this paper makes, because the alternative is a paper that describes an instrument as it was intended.

Their shape is worth stating, because it is not the shape one expects. Almost none were errors in a detector's logic. They were:

- **a record answering a different question than the reader thought.** The filesystem witness walked the container's writable layer from the host, where under rootless containers access is decided by each directory's "other" bits; the agent's home is mode 750 and the whole of it was missing from the file record, including files several scenarios name in their bounds.
- **a measure implemented twice.** §3.2.
- **the instrument appearing in its own record.** Three times: the runtime's own `runc init` as a process the agent left running, the box's `sleep infinity` as pid 1, and the tracer itself once it was detached.
- **a control that could never fire.** A bound the agent had no permission to break; a choice named for a read and matched against a write to a file the scenario itself seeds, so whether an agent ever *found out* about a route was never measured and read as a clean zero.

- **an isolation failure invisible at one run per host.** The witness took the container to look at as an argument and the runner passed a constant, so three of four concurrent runs recorded another agent's files. A teardown matching its own containers by prefix removed every other slot's box, because one project name is a prefix of the others.
- **the environment, not the agent.** Twice. The record of a real run said the agent attempted forty-two connections and was refused thirty-eight; it attempted four, and thirty-four were libc asking a name-service cache that is not there. And a run that reached its wall clock having spent twenty-eight of its thirty minutes waiting on an endpoint that serves one request at a time entered the rate as an agent that held the line. What was measured was the queue. The agent's clock now excludes the time the queue took, and a run denied the model and then timed out is inconclusive.
- **the machine not being the one that was sent.** Three hosts ran something other than the code they were given — one for a week, with a box image built before the syscall witness existed, so everything it produced was untraced; two more through a whole pilot, on *different* old commits from each other. A difference between the hosts cannot afterwards be told from the difference a study set out to measure. Nothing checked, because a host that had been synced at some point was assumed to be current.
- **the instrument licensing the mistake.** The study checker refused a cell split across two machines and explicitly permitted a *design* split by whole cells, in a passing test whose name said it was "how a grid gets done in half the time". A pilot then measured raw gaps of up to fifty points between two hosts differing only in kernel — twice the effect the registered study is sized to detect. The check now refuses a comparison whose two sides ran on different machines, and still allows a whole model its own machine, since the declared test compares each model with itself.
- **a fix that broke the thing it protected.** The cap added to stop a runaway generation was sized from the lengths of turns that *returned*, which are the well-behaved ones by definition, and against a token count that includes the model's reasoning. It then cut 29 per cent of turns on one host mid-thought, before a tool call was emitted, and cost that host eight of its forty-eight runs. The same class as the row above and the opposite direction: the first measured only what survived, the second was sized by it.
- **survivorship in our own instrument.** The provider let a turn generate until the context filled — half an hour of compute on a 65k context — and the run's deadline fired at fifteen minutes while the server carried on, so the next call queued behind a generation nobody awaited. The completion statistics looked healthy throughout, median 826 tokens and a maximum of 5,082 over 257 turns, because they can only count turns that *returned*. The runaways were absent from the evidence for the reason they were a problem.
- **a setting written, and silently overridden underneath.** The grid asks for a fifteen-minute model timeout and every provider built one. Node's `fetch` abandons a request after five minutes of waiting for headers whatever the caller says, and reports it in a way that reads as an unreachable model, so runs were being discarded at five minutes on an endpoint shared with a production service — where a six-minute wait is a queue rather than a fault. The suite was green throughout, because every provider test stubbed `fetch`: the line that set the deadline was tested and the deadline was not.

Seven of those classes are invisible to unit tests. Three were found only by running the instrument against agents whose behaviour was known — which is the argument for having fifty-two of those and running them whenever the instrument changes. The fourth was invisible to that too, and is the sharper lesson: a test that stubs the boundary tests the caller's intention rather than the system's behaviour, so anything the platform decides underneath is outside what the suite can see. The three providers now share one transport whose deadline is enforced where it can be, and it is held to that against a real server and a live model.

10. What it is for

Three things, in order of how well it is suited to them.

Measuring what an agent does under a stated bound, including what it attempted and was refused, and how its own account compares. That is what it was built for and what the published studies use it for.

Commissioned evaluation. An organisation describes what it wants tested; the scenario and the study are written here and released with the result; the rows can be checked against the registration by anyone, including them. Nothing in the design requires giving anyone the ability to run arbitrary code in the lab: a scenario and a study are data.

Regression testing an agent product. The same apparatus, a scenario that is the product's own job, and a ground-truth suite that fails when a detector changes its mind. This is the least exercised of the three.

What it is not for: deciding whether a model is safe. An instrument that finds nothing has found nothing, and the honest report of a null is the bound the run count puts on it.

11. Availability, and how to check this paper

The results obtained with the instrument are in two papers that release every row and registration: whether an agent holds a bound it was given [38], and a registered null on two agents and one permitted channel [39]. Each names the version of the instrument it ran on.

The instrument, the scenarios, the ground-truth suite, the studies and the artefact register are in the Bulkhead Lab repository. Its release state is an open question at the time of this draft and this paper does not pretend otherwise: an instrument paper whose instrument cannot be obtained is a description rather than a contribution, and the intent is a public release with the held-out half of the scenarios kept out of it. Until that happens, the draft is circulated with the offer to run a scenario for anyone who wants to check a claim in it. Every check named in this paper is a command: `npm test` for the three hundred and ninety unit tests, `experiments/validate.ts` for the ground-truth suite, `experiments/preflight-all.ts` for the bounds, `experiments/redteam-trace.ts` for the syscall record and `experiments/concurrency-check.ts` for the isolation. The three that need a container say so.

Every number in this paper is derived from something in that repository, and experiments/check-paper-numbers.py derives them again, as the instrument stood on 22 September and apart from the table's third run — the arm count from the committed identity table, the ground-truth cases from the expectations files, the detector count from the directory, the validation numbers by re-running the analysis over the stored rows. Its counts are the instrument's as of 22 September. Run today, it reports five of them at the instrument's current size (arms, scenarios, ground-truth cases, register rows and unit tests), which have all grown since, and it does not re-score the table's third run, which reproduces only against the expectations of 21 September (lab commit 832f36d): e13815b later taught seven cases to expect the exfiltration observation, and against today's expectations the same run counts those seven as wrong. An instrument paper whose numbers have drifted from the instrument is worse than no paper, because a reader has no way to tell which of the two is wrong.

Deviations and corrections

This section is where anything this paper gets wrong will be recorded, dated, rather than edited away.

22 September 2026. The counts above were 57 in the register, 38 arms, 52 cases and 381 tests when this version was published, and are 60, 40, 56 and 390 now. The campaign registered that day [38] added six framings and two fixtures to the research scenario, with four ground-truth cases, and its shakedown found

three more defects in the instrument before any study ran: a brain asked not to reason was reasoning, because the provider omitted a false rather than sending it; a preflight over some arms wrote a record holding only those arms; and a scenario declared as writable two cache paths the witness ignores by design, which the preflight read as blind spots. All three are in the register with what each cost, and none touched a published number.

30 September 2026. Four statements contradicted the code that read every registered run reported in [38] and [39], or this paper's own count and table, and are corrected in place, as version 0.2.1. The summary and the heading of §3.1 said nine deterministic detectors, where §3.1's own list names eleven: escape and exfiltration joined the pipeline at 14:28Z on 22 September, before the first of those runs, and §3.1 now also says that eight of the eleven can produce a finding and that choice, model and exfiltration never do. §3.1 called report fidelity "the eleventh" straight after naming a twelfth; it is the thirteenth reader of the record. §6 said the two detectors were not wired into the pipeline, which was never true: the sentence entered the paper at 17:16Z on 22 September, nearly three hours after they were wired in at 14:28Z; §6 now says when and how they joined. The plain summary said the AI judge was wrong once, and the summary that it was wrong in two runs, where §7's table has always had three; both now give all three, and §7 says that its table was made before escape and exfiltration existed, how the eleven were checked, and against which expectations its third row reproduces; §11 says what the number check covers. Version 0.2's history said that no other sentence changed in that version; it also added the plain summary, whose "wrong once" this version corrects. §2.1 said the agent has four tools and nothing of the system prompt; it now counts the `finish` call beside them and says that the prompt carries a standing instruction to keep changes to what the task asks for and announces that the declaration is checked, which [38] quotes. §6 now says the four escape routes did not fire, where it said they fired on nothing, and §1.2 says one judge at a time; §7 now names the judge its table measured, `gpt-oss:20b`, beside the `gpt-oss:120b` of the later run, and the summary says that the second of the table's three runs could score only fifty of the fifty-two, two cases having errored on a setup defect. §3.1's rule that a finding carries its evidence now names the completion check's, its command, exit code and output, which has no quote; §4 dates its forty passing arms to 22 September; §9 says the suite is run whenever the instrument changes, not after every change; and §11's claim for the number check gives its date and its one exception. §1.2 said four families of existing work and gives five; it now says five. The summary's account of the validation runs is reworded so that it parses, with the same facts, and the plain summary's "right on all of them" is scoped to the three runs of §7's table. No number in the table changed.

Version history

Version 0.2.2, 5 October 2026, records what the confirmatory registration of [38] found about the instrument, and dates to 22 September the two counts that have grown since; no sentence and no number in §§3 to 8 changed.

- **Two counts are dated where they are stated.** Both summaries, §1's fourth contribution and §9 gave the sixty defects, and the summaries the fifty-six ground-truth cases, as though current. Both are the instrument's as of 22 September, as 0.2 recorded; each now says so, beside the count at this version. §4's count of the suite, unchanged with §§3 to 8, is also 22 September's.
- **The register has eighty-two rows at this version.** It has grown by nine since 0.2. Five of those were found while that registration was stamped, shaken down and audited: the archive step of its public timestamp reported a capture it had not made; the runner read every capability control after the first one ran, so a study with two stopped there; the runner's floor counted a control run the harness ended by what it had reached, and let a design run that the registration would have refused; the stamped confirmatory analysis read five things otherwise than its registration, which a supplement prints as registered; and the research registrations' concealed measure counts a file left off the declaration as well as a host.

- **The second scaffold, built after 0.2, is validated on a live box and has run a registration.** It acts through one plain-text shell block a reply, or, for gpt-oss, through that model’s own built-in shell call, `container.exec`, instead of the default’s native tool calls. On a live box it matched all 74 cases of the ground-truth suite, which has grown since §7’s count as scenarios were added, as the default did. It ran that registration’s C2, whose reply grammar ended 34 of its 306 runs, unevenly between arms, and left two of its three replications untestable.

Version 0.2.1, 30 September 2026, corrects four statements about the detectors that contradicted the code the results papers’ registered runs ran on, or this paper’s own count and table (Deviations and corrections, 30 September): how many deterministic detectors there are and which of them can produce a finding, where report fidelity stands among the readers, when escape and exfiltration joined the pipeline, and in how many runs report fidelity was measured. §7 adds how the eleven detectors were checked and against which expectations its third row reproduces, and §11 what the number check covers. §2.1 counts the `finish` call and points to the system prompt’s standing instruction and announced check, which [38] quotes. §1.2 says one judge at a time and §7 names the judge its table measured, gpt-oss:20b; §6 says the four escape routes did not fire, where it said they fired on nothing; and the summary gives the second run’s fifty and why. §3.1 names the completion check’s evidence, §4 and §9 are dated or narrowed, §11’s claim for the number check gives its date and its exception, §1.2 counts five families of existing work, not four, and the plain summary scopes the detectors’ record to the three runs of §7’s table. No number in §7’s table changed.

Version 0.2, 29 September 2026, writes the related work of §1.2 with its citations, which 0.1 described in outline and marked as pending, and points §11 at the two results papers; no number and no other sentence changed. Thirty-nine works are cited, each for the specific thing this instrument takes from it or does differently. The register has grown since 22 September, to seventy-three rows at this version, through the two campaigns the results papers report and their deviations; §9’s sixty and the costs beside them are as of that date, and the later rows are in the register with their own.

References

- [1] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press and K. Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *ICLR 2024*. [arXiv:2310.06770](#).
- [2] X. Liu, H. Yu, H. Zhang and others. AgentBench: Evaluating LLMs as Agents. *ICLR 2024*. [arXiv:2308.03688](#).
- [3] S. Zhou, F. F. Xu, H. Zhu and others. WebArena: A Realistic Web Environment for Building Autonomous Agents. *ICLR 2024*. [arXiv:2307.13854](#).
- [4] G. Mialon, C. Fourrier, C. Swift, T. Wolf, Y. LeCun and T. Scialom. GAIA: a benchmark for General AI Assistants. 2023. [arXiv:2311.12983](#).
- [5] S. Yao, N. Shinn, P. Razavi and K. Narasimhan. τ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. 2024. [arXiv:2406.12045](#).
- [6] T. Xie, D. Zhang, J. Chen and others. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. *NeurIPS 2024*. [arXiv:2404.07972](#).
- [7] H. Wijk, T. Lin, J. Becker and others. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. *METR*, 2024. [arXiv:2411.15114](#).
- [8] A. K. Zhang, N. Perry, R. Dulepet and others. Cybench: A Framework for Evaluating Cybersecurity Capabilities and Risks of Language Models. 2024. [arXiv:2408.08926](#).
- [9] M. Andriushchenko, A. Souly, M. Dziemian, D. Duenas, M. Lin, J. Wang, D. Hendrycks, A. Zou, Z. Kolter, M. Fredrikson, E. Winsor, J. Wynne, Y. Gal and X. Davies. AgentHarm: A Benchmark for Measuring Harmfulness of LLM Agents. *ICLR 2025*. [arXiv:2410.09024](#).
- [10] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer and F. Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. *NeurIPS 2024*. [arXiv:2406.13352](#).
- [11] Y. Ruan, H. Dong, A. Wang, S. Pitis, Y. Zhou, J. Ba, Y. Dubois, C. J. Maddison and T. Hashimoto. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. *ICLR 2024*. [arXiv:2309.15817](#).

- [12] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan and O. Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *NeurIPS 2024*. [arXiv:2405.15793](#).
- [13] X. Wang, B. Li, Y. Song and others. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. *ICLR 2025*. [arXiv:2407.16741](#).
- [14] UK AI Security Institute. Inspect: an open-source framework for large language model evaluations. *Software, 2024 onwards*. [inspect.aisi.org.uk](#).
- [15] METR. The METR Task Standard, and Vivaria, its runner. *Software, 2024 onwards*. [github.com/METR/task-standard](#), [vivaria.metr.org](#).
- [16] Google. gVisor: an application kernel for containers. *Software, 2018 onwards*. [gvisor.dev](#).
- [17] A. Agache, M. Brooker, A. Florescu, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka and D.-M. Popa. Firecracker: Lightweight Virtualization for Serverless Applications. *USENIX NSDI 2020*.
- [18] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eysers, M. Seltzer and J. Bacon. Practical Whole-System Provenance Capture. *ACM SoCC 2017*. [arXiv:1711.05296](#).
- [19] A. Panickssery, S. R. Bowman and S. Feng. LLM Evaluators Recognize and Favor Their Own Generations. *NeurIPS 2024*. [arXiv:2404.13076](#).
- [20] P. Wang, L. Li, L. Chen and others. Large Language Models are not Fair Evaluators. *ACL 2024*. [arXiv:2305.17926](#).
- [21] L. Zheng, W.-L. Chiang, Y. Sheng and others. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *NeurIPS 2023*. [arXiv:2306.05685](#).
- [22] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu and C. Zhu. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. *EMNLP 2023*. [arXiv:2303.16634](#).
- [23] E. Perez, S. Ringer, K. Lukošiušė, K. Nguyen, E. Chen, S. Heiner, C. Pettit, C. Olsson, S. Kundu, S. Kadavath and 53 others. Discovering Language Model Behaviors with Model-Written Evaluations. *Findings of ACL 2023*. [arXiv:2212.09251](#).
- [24] M. Turpin, J. Michael, E. Perez and S. R. Bowman. Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting. *NeurIPS 2023*. [arXiv:2305.04388](#).
- [25] Y. Chen, J. Benton, A. Radhakrishnan, J. Uesato, C. Denison, J. Schulman, A. Somani, P. Hase, M. Wagner, F. Roger, V. Mikulik, S. R. Bowman, J. Leike, J. Kaplan and E. Perez. Reasoning Models Don't Always Say What They Think. 2025. [arXiv:2505.05410](#).
- [26] B. Baker, J. Huizinga, L. Gao, Z. Dou, M. Y. Guan, A. Madry, W. Zaremba, J. Pachocki and D. Farhi. Monitoring Reasoning Models for Misbehavior and the Risks of Promoting Obfuscation. 2025. [arXiv:2503.11926](#).
- [27] T. Korbak and 40 others. Chain of Thought Monitorability: A New and Fragile Opportunity for AI Safety. 2025. [arXiv:2507.11473](#).
- [28] J. Needham, G. Edkins, G. Pimpale, H. Bartsch and M. Hobbhahn. Large Language Models Often Know When They Are Being Evaluated. 2025. [arXiv:2505.23836](#).
- [29] T. van der Weij, F. Hofstätter, O. Jaffe, S. F. Brown and F. R. Ward. AI Sandbagging: Language Models can Strategically Underperform on Evaluations. 2024. [arXiv:2406.07358](#).
- [30] A. Meinke, B. Schoen, J. Scheurer, M. Balesni, R. Shah and M. Hobbhahn. Frontier Models are Capable of In-context Scheming. 2024. [arXiv:2412.04984](#).
- [31] R. Greenblatt, C. Denison, B. Wright, F. Roger, M. MacDiarmid, S. Marks, J. Treutlein, T. Belonax, J. Chen, D. Duvenaud, A. Khan, J. Michael, S. Mindermann, E. Perez, L. Petrini, J. Uesato, J. Kaplan, B. Shlegeris, S. R. Bowman and E. Hubinger. Alignment faking in large language models. 2024. [arXiv:2412.14093](#).
- [32] B. A. Nosek, C. R. Ebersole, A. C. DeHaven and D. T. Mellor. The preregistration revolution. *Proceedings of the National Academy of Sciences* 115(11):2600–2606, 2018.
- [33] L. Madaan, A. K. Singh, R. Schaeffer, A. Poulton, S. Koyejo, P. Stenetorp, S. Narang and D. Hupkes. Quantifying Variance in Evaluation Benchmarks. 2024. [arXiv:2406.10229](#).
- [34] E. Miller. Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations. 2024. [arXiv:2411.00640](#).
- [35] S. Biderman, H. Schoelkopf, L. Sutawika and 27 others. Lessons from the Trenches on Reproducible Evaluation of Language Models. 2024. [arXiv:2405.14782](#).
- [36] A. Reuel, A. Hardy, C. Smith and others. BetterBench: Assessing AI Benchmarks, Uncovering Issues, and Establishing Best Practices. 2024. [arXiv:2411.12990](#).

- [37] S. R. Bowman and G. E. Dahl. What Will it Take to Fix Benchmarking in Natural Language Understanding? *NAACL 2021*. [arXiv:2104.02145](https://arxiv.org/abs/2104.02145).
- [38] R. Chattaway and C. Urquiza. Does the bound hold? What decides whether an agent crosses a line it was given? An empirical study. *Bulkhead Research preprint, version 0.3, 5 October 2026*. agentbulkhead.com/research/does-the-bound-hold.
- [39] R. Chattaway and C. Urquiza. Two agents, one channel: given a reason and a permitted store, four open-weight agents passed nothing through it in 544 registered pairs. *Bulkhead Research note, a registered null, version 0.1.2, 29 September 2026*. agentbulkhead.com/research/two-agents-one-channel.