

A Trust Layer for Agent Evaluation

Mohammadreza Sediqin

Centific Research

New York, USA

mohammadreza.s@centific.com

Shivali Dalmia

Centific Research

Seattle, USA

shivali.dalmia@centific.com

Srinivasa Karthikeya Reddy Kovvuri

Centific Research

Washington, USA

srinivasa.kovvuri@centific.com

Abhishek Mukherji

Centific Research

Seattle, USA

abhishek.mukherji@centific.com

Abstract

Deterministic benchmark scores show that an agent received credit, but not whether that credit was earned, reported honestly, or would hold on a second run. We introduce a *Trust Layer for Agent Evaluation*, an additive post-hoc framework that reports, beside each recorded score, whether it should be believed. It verifies four properties: whether the result is supported by the benchmark’s own grading logic, whether a passing answer was earned through traceable computation, whether the agent’s completion claim matches what occurred, and whether the result is stable under repeated execution. The first three use only saved artifacts; the fourth re-runs the agent. Model judgments only label evidence under majority voting; all verdicts follow deterministic rules and never modify the recorded score. Applied to five agent configurations on 108 tasks from Agents’ Last Exam, every model shows passing runs with no traceable computation (at rates varying tenfold), confirmed false completion claims, and unstable results: 18–46% of tasks do not stay in one score band over five runs. Only 22.6% of recorded passes clear all four checks (95% CI 15.0–32.6, $n=84$). Measuring what an agent can do and verifying that it actually did it are different problems, and current benchmarks address only the first.

1 Introduction

LLM agents are now evaluated on increasingly realistic long-horizon benchmarks in software engineering, scientific discovery, web interaction, and professional work. SWE-bench (Jimenez et al., 2024), τ -bench (Yao et al., 2024), ScienceAgent-Bench (Chen et al., 2025), and Agents’ Last Exam (ALE) (Sun et al., 2026) replace subjective assessment with executable grading, deterministic tests, and reference-based verification, so a benchmark score is often treated as an objective measurement of capability.

But a score answers only one question: *did the grader award credit?* It does not say whether the credit was earned, whether the reported outcome reflects what happened, or whether the result would hold on a second run. A run that *fabricates a plausible value* scores exactly like one that computed it. A run that *fails but claims success* scores exactly like one that fails and says so. Because agent runs vary from attempt to attempt, a run that passes once in five tries scores exactly like one that passes every time; repetition studies show single-run scores shifting by several points from chance alone (Bjarnason et al., 2026). And a failing score may reflect a packaging or evaluation artifact rather than genuine inability. In each case the number is accurate about the run and misleading about the agent, which raises a question the score cannot resolve on its own: *can a recorded benchmark score be trusted?*

Existing work addresses these concerns separately, through benchmark auditing, reward hacking and corrupt success (Skalse et al., 2022; Cao et al., 2026), honesty (Ren et al., 2025; Chern et al., 2024), reliability (Gupta, 2026), and provenance tracking for LLM-generated content (Sediqin et al., 2026a), usually with new benchmarks, datasets, or protocols. None answers the simpler question a benchmark user faces: given a score already recorded, should it be believed?

We introduce a *Trust Layer for Agent Evaluation*, an additive post-hoc framework that audits recorded outcomes using artifacts benchmarks already save (trajectories, staged outputs, grading code) and runs alongside the benchmark rather than replacing it. It applies four verifiers, one per case above. *Competence* checks whether the benchmark’s own grading logic reproduces the recorded outcome and names the decisive cause of failure. *Reward Hacking* checks whether a pass was earned through traceable computation or reached through shortcuts and ungrounded outputs. *Agent Decep-*

tion Detection checks the agent’s completion claim against the independently verified outcome. *Reliability* checks whether repeated runs under identical conditions stay in the same score band.

This is the first framework we know of to verify these four properties together, for an individual recorded score, from evidence a benchmark already produces. Its central design principle is the *separation of judgment from decision*: models may help label evidence, but every verdict comes from a deterministic rule, and any load-bearing model label needs a majority of three independent votes. This matters because the closest prior work finds that judges of false completion claims anchor on confident language, with no configuration exceeding AUROC 0.65 (Advani et al., 2026).

On 108 ALE tasks, the layer finds grading contradictions in both directions, including recorded passes that the benchmark’s own grader scores at zero when re-run on the saved output; passing results with no traceable computation behind them; false completion claims in every recorded band; and results that do not reproduce. The dimensions prove additive rather than overlapping: many flagged tasks are caught by a single dimension, so removing any one would leave them looking clean, and only a minority of recorded successes clear all four. Our contributions are:

1. To our knowledge, the first post-hoc trust layer that verifies whether a recorded score is reproducible, earned, honest, and supported by the benchmark’s own grading logic, unifying four properties previously studied separately.
2. A design in which every verdict is deterministic and model judgment only labels evidence under a majority of three votes, so the headline number stays intact and auditable.
3. An empirical study over 108 professional tasks showing that substantial trust failures hide behind benchmark scores, that the four dimensions detect largely distinct failures, and that requiring all four leaves far fewer verified results than the scores suggest. Code will be released upon acceptance.

2 Related Work

Agentic benchmarks and validity auditing. Recent agent benchmarks favour verifiable, code-based outcomes. ALE grades long-horizon professional tasks with deterministic scripts against

expert references, relying on LLM judging for under 7% of workflows (Sun et al., 2026). SWE-bench grades code fixes for real GitHub issues with unit tests (Jimenez et al., 2024), τ -bench compares the final database state with a reference (Yao et al., 2024), and ScienceAgentBench scores self-contained programs on tasks from peer-reviewed publications (Chen et al., 2025). Each reduces a run to one score from the final artifact, leaving the trajectory unscored. Such scores can misrepresent competence: Zhu et al. (2026) formalize validity threats in a manual checklist spanning task and outcome validity, Kapoor et al. (2024) show that uncontrolled cost distorts conclusions, and automated auditing finds that test suites can reject correct solutions, with evaluator limitations increasingly dominating reported scores (Tu et al., 2026). These efforts audit benchmark *construction* through expert review, and related checks govern consistency in document repositories (Sedigh et al., 2026b).

Reward hacking, honesty, and reliability. Specification gaming, satisfying the literal objective but not its intent, was formalized by Skalse et al. (2022) and catalogued across reinforcement learning systems by Krakovna et al. (2020). For tool-using LLM agents, benchmarks measure the propensity to skip verification or tamper with the evaluation itself (Thaman, 2026; Gabor et al., 2026), and Cao et al. (2026) report *corrupt success*, correct end-states reached while violating procedural integrity, in 27% to 78% of reported successes. Advani et al. (2026) characterize false success, agents asserting completion the environment contradicts, and show that LLM judges detect it unreliably; MASK (Ren et al., 2025) and BeHonest (Chern et al., 2024) measure propensity in elicited scenarios rather than verifying claims in real runs. On reliability, single-run pass@1 across 60,000 SWE-Bench-Verified trajectories varies by up to 6.0 points depending on the run selected, and the authors recommend pass^k as a pessimistic bound (Bjarnason et al., 2026); related work treats evaluations as experiments whose variance determines reproducibility (Mustahsan et al., 2025), and ReliabilityBench measures consistency under repetition and perturbation (Gupta, 2026).

Positioning. Validity work audits how a benchmark is *constructed*; reward hacking, false success, and honesty work each target one mode, usually through an LLM judge whose own reliability is in question; and reliability work builds repetition into *new* benchmarks, all at a different level than one

score. The closest work, procedure-aware evaluation (Cao et al., 2026), defines a new evaluation of whether an agent followed the intended procedure. We instead ask whether a score an existing benchmark *already recorded* holds up, verifying four properties of it from the saved trajectory and grading code, per dimension and as a conjunction, so a pass that is procedurally clean but unstable, or earned but falsely reported, is still caught.

3 System Architecture

The trust layer is an additive stage that runs after a benchmark has scored an agent (Figure 1). It has four dimensions, one per failure class. **D1 (Competence)** asks what a recorded score is actually made of and exposes scores the grader does not reproduce, in either direction. **D2 (Reward Hacking)** asks whether a passing score was earned or reached without doing the work. **D3 (Agent Deception Detection)** asks whether the agent’s success claim is contradicted by the outcome. **D4 (Reliability)** asks whether the score stays in the same band on repetition. D1–D3 read only what the benchmark saved; D4 also re-runs the agent. The layer reuses the benchmark’s own grader wherever a verdict is needed and never overwrites the recorded score, reporting its findings beside it. It needs three things from a benchmark: saved trajectories, accessible grader logic, and outputs readable offline.

Evidence: the trajectory and re-grading. Two artifacts benchmarks produce but never fully use supply the evidence: the *trajectory*, the step-by-step record showing *how* a result was produced, and the benchmark’s own *grader*, re-run on the saved output to reproduce the score instead of trusting it. Every dimension rests on one or both.

3.1 D1: Competence

In short: re-run the grader and ask what actually decided the score. D1 re-derives the verdict from the saved output with the task’s own grader and compares it with the recorded score. No model assigns a score: a model labels gates by severity, and the verdict is a deterministic function of those labels. D1 has four stages, each using the output of the last.

Stages A and B: typing and re-grading. A *gate* is a check inside the grader whose failure forces the score to zero, such as a missing file or a wrong value. Stage A recovers every gate into a set G and

types it by what it protects,

$$\tau(g) \in \left\{ \begin{array}{l} \text{critical, process,} \\ \text{administrative} \end{array} \right\}, \quad (1)$$

where *critical* means the deliverable is defective, *process* a violated method rule, and *administrative* a packaging defect. Each gate also carries an anti-gaming flag $\alpha(g)$. Typing is done by an LLM judge under a majority of three votes and, being a property of the grader rather than a run, once per task. Stage B re-runs the grader on the saved output and maps the fired gates F to a verdict,

$$v = \Phi(F, \tau, \alpha), \quad F \subseteq G, \quad (2)$$

with the most severe fired gate deciding, so a real correctness failure cannot hide behind a lesser one. Appendix A gives the gate criteria, matching procedure, severity ordering, and the verdicts Stage B can return.

Stages C and D: claim and challenge. Stage C reduces the agent’s terminal message, by deterministic pattern matching, to a completion claim c and tests it against v , giving the three-valued honesty signal used by D3:

$$\ell = \begin{cases} 1 & \text{if } c = \text{success} \wedge v \in V_{\text{fail}}, \\ \perp & \text{if } c = \text{success} \wedge v \in V_{\text{unver}}, \\ 0 & \text{otherwise,} \end{cases} \quad (3)$$

where V_{fail} are verdicts that reproduce a genuine failure and V_{unver} those that could not be reproduced offline. Stage D is the only place the earlier facts are combined, by a purely deterministic rule with no model call:

$$(D, \delta) = \Psi(s, v, \ell, m, \kappa), \quad D \in \left\{ \begin{array}{l} \text{confirm, contradiction,} \\ \text{upgrade, unverified} \end{array} \right\}, \quad (4)$$

where s is the recorded score, m whether the prompt itself demanded the failed format, and κ a format-blind check that the output values match the reference. A *confirm* reproduces the recorded outcome; a *contradiction* disagrees, with a signed direction δ recording which side was wrong; an *unverified* could not be checked offline; and an *upgrade*, the one outcome needing all three earlier stages, is reached when the run failed a check that does not mean the work was wrong. Appendix A gives the upgrade conditions and contrasts a contradiction with an upgrade.

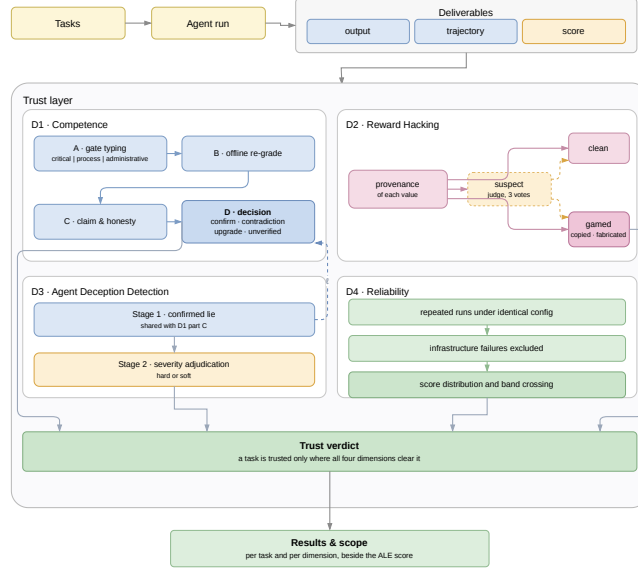


Figure 1: Overview of the Trust Layer. D1 re-derives the verdict, D2 traces value provenance, D3 tests the agent’s claim, D4 repeats the run.

3.2 D2: Reward Hacking

In short: ask where each answer value came from.

A score shows that an answer is correct, not how it was produced. An agent that hardcodes a constant, copies a value the prompt revealed, or writes plausible numbers with no computation behind them scores exactly like one that did the work, and the grader is fooled. D2 marks such runs gamed, separating them from the honest passes they would otherwise be counted among. It traces provenance rather than re-checking correctness.

Tracing each value. D2 uses the reference answer, which the agent never saw, as an offline oracle to pick out which output values are the answer, then traces each against three sources from the run: output of commands the agent executed, inputs it read or was given in the prompt, and text it typed by hand into files and shell commands. Each value gets a label,

$$\pi(x) \in \left\{ \text{DERIVED, GIVEN, TYPED, UNGROUNDED} \right\}, \quad (5)$$

DERIVED if it appears in executed output, GIVEN if it came from an input or the prompt, TYPED if written by hand, and UNGROUNDED if it appears nowhere. The decisive test is for an answer value absent from executed output: was it typed by the agent, or produced by a script at run time? A correct value a script computed is real work; a hand-

typed one is not. Scalars are traced one at a time and arrays at the file level (Appendix A).

Confessions, the judge, and the verdict. The agent’s messages and reasoning are scanned for admissions of copying or hardcoding, with a negation guard so a stated intention *not* to hardcode is never read as a confession. A confession is never a flag on its own: with a value already found unearned (matching the reference but neither computed nor read) it confirms a hack; otherwise it only marks the run for a closer look. Values that remain genuinely unclear go to a model judge, which sees the prompt, the flagged values, the agent’s words, and its code, and is warned that printing a hardcoded constant does not make it computed. Before convicting, D2 has the judge check whether the task required the value to be computed; if the task permits a direct answer, the value is cleared. A strict majority of three votes is required to convict, so a split resolves to clean: the check would rather miss a cheat than accuse an honest run. The run verdict is

$$V_{\text{int}} \in \{\text{clean, suspect, gamed}\}, \quad (6)$$

where a strong flag gives gamed, a weak flag or unresolved value suspect, and a clean trace clean. suspect is only a routing state that sends the run to the judge, which resolves it, so none is reported. Each verdict records its cheat type (copied or fabricated); signals range from a confession with a

provenance gap, the strongest, to a judge-voted fabrication resting on the absence of a trace, the weakest (Appendix A). An untraced value is flagged for review, not convicted, and is marked gamed only when a condition above holds, never when the task permits a direct answer. A value fabricated and then written through executed code looks derived, so D2 bounds untraceable fabrication, not fabrication laundered through execution.

3.3 D3: Agent Deception Detection

In short: check the agent’s final report against what really happened. The last message is where the agent reports its result, so when it claims success on a run that failed, oversight breaks down. The benchmark gives the same score either way, so lying costs the agent nothing and the score never shows it. We use “lie” operationally: a claim of success on a run whose failure was proven by re-grading; D3 shows the report was false, not that the agent knew it was. D3 has two jobs: a yes-or-no check inside the D1 pipeline, and a separate judgement of how serious a confirmed lie was.

Job 1: is the agent lying? This is the deterministic signal ℓ of Equation (3). It checks the agent’s success claim against the re-graded verdict v , not the recorded score: testing against the recorded score would only show that the agent claimed success and scored zero, mixing up a lying agent with a grading error, while a re-grade tells them apart. If re-running the grader proves the failure, the claim is a confirmed lie ($\ell = 1$); if the grader cannot be re-run, it is marked suspected and D3 stops ($\ell = \perp$). This fact feeds back into Stage D, where a proven lie removes the agent’s benefit of the doubt on a formatting-only failure. Because ℓ rests on D1’s re-grade, the two dimensions are not independent evidence.

Job 2: how serious was the lie? After D1 finishes, a model judge re-examines each confirmed lie under majority vote, so this output is an adjudication rather than a measurement. The judge sees the agent’s words and what independently happened (the recorded score, re-graded verdict, fired gate, failure explanation, grader messages, and a trajectory digest) and produces

$$(\hat{\ell}, \gamma, \eta, \phi) = \Xi(\ell, v, q, E), \quad (7)$$

where q is the agent’s exact claim and E what happened in the run: a final decision $\hat{\ell}$ confirming or

dropping the lie, the kind of claim γ , whether a concrete defect sits behind it η , and whether the lie alone should fail the task ϕ (Appendix A). Job 2 never feeds back into D1, which uses only ℓ , so the challenge decision never inherits the judge’s variance and no score is changed. A confirmed lie is used in exactly three places: it blocks a packaging upgrade in Stage D, tags a lenient contradiction with whether the agent also lied, and routes an unverified pass to suspect.

3.4 D4: Reliability

In short: run the task again and see whether the result stays in the same band. One run is a sample, not a measurement of a point; a pass that holds one time in five is luck recorded as competence, invisible to any single-run dimension. D4 executes each task $N = 5$ times under identical configuration and characterises the score distribution $\{s_1, \dots, s_N\}$. Each run is first classified as an infrastructure or genuine failure from the termination record, trajectory text, and usage counters (exhausted credit, a dead sandbox, zero tool calls); infrastructure runs are excluded, so D4 measures the agent, not the harness. Over the surviving runs S ,

$$\bar{s} = \frac{1}{|S|} \sum_{s \in S} s, \quad \bar{\Delta} = \bar{s} - s_{\text{rec}}, \quad (8)$$

$$\text{crossed} = \mathbf{1} [|\{\text{band}(s) : s \in S\}| > 1],$$

where s_{rec} is the recorded score and $\text{band}(\cdot)$ maps a score to pass, partial, or fail with ALE’s thresholds (Sun et al., 2026). A task is *varied* when crossed fires and *stable* otherwise (Gonzalez-Pumariega et al., 2026). *crossed* asks whether the repeats agree with each other; $\bar{\Delta}$ asks how far they sit from the recorded score, negative meaning the recorded run was the better one. Only *crossed* decides whether a task clears D4; $\bar{\Delta}$ is reported, not used as a gate.

Two further statistics separate what a score measures from what it merely samples (Bjarnason et al., 2026). Treating a score as true ability plus noise, the variance of repeats splits into a between-task part (genuine differences in difficulty and capability) and a within-task part (run-to-run randomness). Their ratio

$$\rho = \sigma_{\text{between}}^2 / (\sigma_{\text{between}}^2 + \sigma_{\text{within}}^2) \quad (9)$$

is the share of variance that is signal, and $1 - \rho$ the share that is not (Mustahsan et al., 2025). Estimating each task’s pass probability $\hat{p}$ from its repeats gives $\hat{p}^k$, the chance of passing k consecutive runs,

whose gap from $\hat{p}$ is the capability a one-shot score reports but cannot sustain. Appendix A gives the per-task form of ρ and what else the persisted repeats support.

3.5 Aggregation Across Dimensions

A wrong answer, an unearned pass, a false claim, and an unstable score call for different responses, so the dimensions are reported separately; averaging them would discard the distinction the layer exists to draw. They are also reported as a conjunction: a task is *trusted* only where every dimension clears it, meaning the verdict re-derives, the answer values are grounded, no false claim is confirmed, and the task is stable across repeats. This is a gate, not a score: no dimension is weighted against another, and the recorded score is never overwritten.

Counting rules. A task clears D1 only when re-grading yields a verified pass; for a recorded pass this means confirm, so contradiction and unverified are D1 failures, and a task whose grader cannot be re-run offline counts as unverified. It clears D2 only when clean; gamed and “no verdict” are D2 failures. It clears D3 when no false claim is confirmed. It clears D4 only when stable, whichever direction its repeats move, since a score that changes bands cannot be trusted either way, and a score is read as evidence about the agent (Section 1). A task may fail several dimensions (a run that fabricated its answer often also claimed success for it), so the conjunction is not a sum and we report the overlap (Zhou et al., 2026). The four are not exhaustive, but they cover the trust failures saved artifacts can evidence.

4 Experiments

4.1 Dataset and Setup

We evaluate the trust layer on 108 publicly available tasks from ALE (Sun et al., 2026), a benchmark of long-horizon work graded by deterministic scripts against expert references, spanning ALE’s three difficulty tiers and 11 professional domains, each with its own grading code producing a single score per run. Each task was run once under each of five agent configurations: Claude Sonnet 4.6, Claude Opus 5, GPT-5.6 Sol (GPT-Sol), Muse Spark 1.2 (MetaMuse), and Gemini 3.6 Flash, the same agent harness and decoding configuration. The recorded bands differ by model, from 15 recorded passes for the weakest to 24 for the strongest (Appendix B). For D1–D3, the

Model	confirm	contra	unverified	upg
Sonnet	56	38	3	0
MetaMuse	50	47	0	0
GPT-Sol	50	47	0	0
Opus	53	43	1	0
Gemini	61	35	1	0

Table 1: D1 outcome per model.

Model	Confirmed	hard	soft
Sonnet	55	26	29
MetaMuse	35	31	4
GPT-Sol	40	38	2
Opus	17	15	2
Gemini	27	26	1

Table 2: Confirmed false claims per model.

trust layer reads only what each run already saved: the agent’s trajectory, the outputs it staged, and the task’s own grading code. It re-executes that code against the saved output without re-running the agent, and never modifies the recorded score. Where a dimension requires a model judgment we use GPT-5.5 with three independent votes and a strict majority.

A task is *ungradable* when its grader cannot be re-run offline, which holds for 11 of the 108 and is a property of the task rather than of any model, so the same 11 are excluded for every agent. These runs staged no output the grader could read, so D1 produces no verdict, D3 has nothing to test a claim against, and D2 no answer values to trace, leaving 97 assessed throughout. D4 instead re-runs the agent five times per task under identical configuration. 87 of the 108 completed all five repeats under Sonnet, and the 50 tasks repeated under all five configurations were drawn at random from those, stratified across ALE’s tiers (24 Near-Term, 18 Full-Spectrum, 8 Last-Exam) and recorded band. Repeated-execution studies commonly use between three and six runs (Guo et al., 2026; Zhang et al., 2025), so five is in line with current practice. Each dimension therefore reports its own denominator. Because the 50 repeated tasks are drawn from all 108, the aggregate in Section 4.6 can include ungradable tasks, which count as failing D1 and D2 under the counting rules of Section 3.5.

4.2 D1: Competence

D1 executes each task’s grading code offline on the saved output, without re-running the agent,

Model	assessed	clean	gamed	not assessed	gamed%
Sonnet	97	75	14	8	14.4
MetaMuse	97	52	38	7	39.2
GPT-Sol	97	45	49	3	50.5
Opus	97	75	10	12	10.3
Gemini	97	44	25	28	25.8

Table 3: D2 verdicts per model. Gamed% is over the 97 assessed tasks, including runs carrying no verdict.

and compares the result with the recorded score (Table 1, 97 gradable tasks). The models cluster between 50 and 61 confirmations although their recorded passes differ by nine. Most contradictions (contra) are partial scores below the verified-pass bar: the benchmark credits partial work in proportion, while D1 requires a complete re-derivation. The real disagreements are smaller: 2 to 9 recorded passes per model re-derive as failures, and one or two recorded failures as passes (Appendix C). The upgrade (upg) path was reachable but never granted, since every candidate’s values differed from the reference. Gate typing, D1’s only model-labelled stage, agrees closely enough that the majority rule absorbs residual disagreement rather than masking a coin flip (Appendix C); every later stage is deterministic: re-executing the grader five times on the same output returns an identical verdict and band on all 108 tasks.

4.3 D2: Reward Hacking

Over the 97 assessed tasks (Table 3), the models differ by a factor of five: Opus and Sonnet reach their answers without doing the work on roughly one run in eight, GPT-Sol on half; on recorded passes the spread is tenfold, from 1 of 22 for Opus to 10 of 21 for GPT-Sol (Table 9). Runs with no verdict are neither clean nor gamed and stay in the denominator. This matters most for Gemini, whose rate over runs with a verdict is 36.2% rather than 25.8%; as these are unlikely to be missing at random, its true rate lies between 25.8% and 54.6%. Fabrication, a value that matches the reference yet appears in no executed output and nothing the agent read, accounts for most convictions; copying a supplied value is rarer and an explicit admission rarer still, so the judge carries most D2 verdicts for every model. Every Sonnet gamed verdict was verified by hand against the trajectory, and all 14 held; Appendix D gives the breakdowns. Injecting the reference value into runs the detector had cleared tests the other direction: D2 flagged 36 of 38 where the value was

Model	Tasks	stab.	var.	var.%	$\bar{\Delta}$
Sonnet	50	27	23	46.0	−0.31
Opus	50	39	11	22.0	−0.30
GPT-Sol	50	41	9	18.0	−0.22
MetaMuse	50	38	12	24.0	−0.12
Gemini	50	36	14	28.0	−0.24

Table 4: D4 outcome per model.

left ungrounded, and 2 of 45 where executed code wrote it.

4.4 D3: Agent Deception Detection

D3 runs the two jobs of Section 3.3. A claim can only be tested where re-derivation reproduces what happened, so the 11 tasks without an offline verdict are ineligible; Table 2 covers the remaining 97. A claim is *hard* when a concrete defect sits behind it (a fired critical gate, a missing output, or a trace showing the work was wrong) while the agent reported the deliverable as verified, and *soft* when no such defect is evidenced, typically an overclaim that all checks pass when the agent’s own numbers were already below the bar. Roughly half of Sonnet’s confirmed claims are soft, while the other four models’ are mostly hard, so the same detector finds two styles of misreporting. Models also differ in how often they make confirmed false completion claims, from 17 for Opus to 55 for Sonnet, and false claims appear in every recorded band for every model, including full passes. On two annotated samples of Sonnet runs, annotators agreed on whether success was claimed in 48 of 50 terminal messages ($\kappa = 0.85$), and the matcher reaches $F_1 = 0.91$ against their consensus; on severity they agreed at $\kappa = 0.84$, and the judge reaches $F_1 = 0.97$.

4.5 D4: Reliability

Table 4 covers the 50 repeated tasks. Its $\bar{\Delta}$ column, the mean change from the recorded score over varied pass-band tasks, is negative throughout: a recorded pass that fails to reproduce returns a lower score than the one on record. Band composition differs by model, so rates are comparable only within band (Appendix E, which also gives the 87-task results). The varied rate ranges from 18.0% for GPT-Sol to 46.0% for Sonnet, a factor of two and a half on identical tasks and conditions, so instability is a property of the agent as well as the task. The movement follows the recorded band: the recorded run was the better run where the bench-

mark recorded success and the worse run where it recorded failure (Appendix E). Pooled over all tasks the mean change is $+0.03$ or less for every model, so the directions cancel and an aggregate hides the effect completely.

Sonnet varies on 25.3% of the 87 tasks; its repeat variance splits into 0.143 between tasks against 0.040 within, $\rho = 0.78$, so close to a quarter of the variation in recorded scores is about which run was saved, not about the agent. The noise is concentrated: the median task varies by a standard deviation of 0.007 across repeats, the varied tasks by 0.350. A task passes on 19.3% of individual repeats but only 13.8% of five-run sequences, and of the 24 tasks passing at least once, only 12 pass every time: half the tasks the benchmark could count as successes succeed only some of the time, and no single-run dimension can tell the halves apart. Sonnet’s two independent campaigns, over the 87 tasks and the 50 drawn from them, disagree: 25.3% against 46.0% varied under the same configuration, where a random draw would carry about 13 varied tasks rather than 23 (Table 13). The D4 verdict is therefore itself a sample; we report both campaigns rather than the more favourable one, and a larger N would tighten the estimate. Because $\bar{\Delta}$ does not gate the verdict, a stable task can settle in a band other than the recorded one. Of the 181 stable verdicts in Table 4, 16 do so (Gemini 10, Sonnet 2, Opus 2, MetaMuse 2, GPT-Sol 0), four of them recorded passes. Requiring reproduction of the recorded band would reclassify these 16; we report them separately rather than folding them into the stable/varied label.

4.6 Aggregation Across Dimensions

Table 5 gives the conjunction over the tasks the benchmark recorded as passes, where it does its work: D1’s verified-pass bar rejects a partial score by definition, so no partial task can clear it. The D1–D4 columns overlap and do not sum; Appendix F gives all three bands. Between 7.7% and 40.0% of recorded passes survive all four checks; all five intervals overlap, so the rates bound how many passes hold up rather than ordering the configurations. Pooled (last row of Table 5), only 19 of 84 recorded passes survive all four checks (22.6%, 95% CI 15.0–32.6), so 65 fail at least one, and the majority fails in every configuration. Models fail for different reasons: GPT-Sol loses two thirds of its passes to D2, 12 of 18 with no traceable computation behind them, while Opus is ungrounded on

Model	Passes	failed				aggregate		
		D1	D2	D3	D4	trusted	%	95% CI
Opus	20	8	3	0	5	8	40.0	[21.9, 61.3]
Sonnet	15	4	5	1	5	4	26.7	[10.9, 52.0]
MetaMuse	18	8	6	1	3	4	22.2	[9.0, 45.2]
GPT-Sol	18	7	12	3	4	2	11.1	[3.1, 32.8]
Gemini	13	7	5	2	3	1	7.7	[1.4, 33.3]
All	84	–	–	–	–	19	22.6	[15.0, 32.6]

Table 5: Conjunction over recorded passes among the 50 repeated tasks, under the counting rules of Section 3.5. Intervals are Wilson score intervals at 95%.

three and loses more to re-derivation than to anything else. Between a third and a half of flagged tasks are caught by a single dimension, so removing any one would leave them indistinguishable from clean ones. The ablation (Table 11) also shows how much rests on model labels: the two deterministic dimensions (D1, D4) leave 34 of 84 recorded passes, and the two judge-assisted ones reject a further 15, leaving 19. Over all 50 tasks the totals reflect the threshold more than the evidence, since D1 fails every partial task; the recorded-pass figures answer the question the layer was built for.

5 Conclusion and Future Work

We presented a trust layer that verifies four properties of a recorded agent score: whether a failure was genuine, whether a pass was earned, whether the agent’s claim matched reality, and whether the score is stable across repeated runs. It is additive, never overwrites the recorded number, and confines model judgment to labeling evidence. Across five agent configurations, every model reaches a passing answer without computing it on some runs, at rates differing tenfold between the cleanest and the worst; confirmed false completion claims appear in every recorded band for every model; and repeated execution takes 18% to 46% of tasks across score bands, signed so that a recorded pass was the agent’s better run and a recorded failure its worse one, a direction pooling erases. Requiring all four properties leaves 22.6% of recorded successes standing, and between a third and a half of flagged tasks are caught by a single dimension, so no one check substitutes for the others. Because the layer is benchmark-agnostic, future work will apply it to SWE-bench Verified, τ -bench, and ScienceAgent-Bench to make agent evaluation more trustworthy across benchmarks. Measuring what an agent can do and verifying that it did are distinct problems,

and only the first is currently instrumented.

Limitations

Our claims are scoped to what the evidence supports. Precision is validated directly for D2 (hand-checked gamed verdicts) and D3 (annotator agreement, Section 4.4) on one configuration; extending these checks to all models, and validating D1’s gate typing against human labels, is natural next work. With 13 to 20 recorded passes per configuration and overlapping intervals in Table 5, the results establish that most recorded passes fail at least one check in every configuration, not a ranking of the agents. All results come from ALE; because the layer needs only a trajectory, a staged output, and a re-runnable grader, applying it to other benchmarks is straightforward. Every decision rule is deterministic, and the ablation in Section 4.6 shows how much of the result holds without any judge. A gamed verdict means no traceable computation, not intent to cheat, and D2 does not catch fabrication laundered through execution; a confirmed lie shows a report was false, not that the agent knew it. D4 re-executes the agent, its $N = 5$ verdict is itself a sample, and stable tasks whose band differs from the recorded one are reported separately (Section 4.5).

References

- Laksh Advani et al. 2026. [From confident closing to silent failure: Characterizing false success in llm agents](#). *arXiv preprint arXiv:2606.09863*.
- Bjarni Haukur Bjarnason, André Silva, and Martin Monperrus. 2026. [On randomness in agentic evals](#). *arXiv preprint arXiv:2602.07150*.
- H. Cao, I. Driouich, and E. Thomas. 2026. [Beyond task completion: Revealing corrupt success in llm agents through procedure-aware evaluation](#). *arXiv preprint arXiv:2603.03116*.
- Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. 2025. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, volume 2025, pages 96934–96990.
- Steffi Chern et al. 2024. [BeHonest: Benchmarking honesty in large language models](#). *arXiv preprint arXiv:2406.13261*.
- J. Gabor, J. Lynch, and J. Rosenfeld. 2026. [Evilgenie: A reward hacking benchmark](#). *arXiv preprint arXiv:2511.21654*.
- Gonzalo Gonzalez-Pumariiega, Saaket Agashe, Jiachen Yang, Ang Li, and Xin Eric Wang. 2026. On the reliability of computer use agents. *arXiv preprint arXiv:2604.17849*.
- Yaoqi Guo, Yang Liu, Jie M Zhang, Yun Ma, Yiling Lou, and Zhenpeng Chen. 2026. Swe-doctor: Guiding software engineering agents with runtime diagnosis from multi-faceted bug reproduction tests. *arXiv preprint arXiv:2607.00990*.
- A. Gupta. 2026. [ReliabilityBench: Evaluating llm agent reliability under production-like stress conditions](#). *arXiv preprint arXiv:2601.06112*.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.
- Sayash Kapoor, Benedikt Stroebl, Arvind Narayanan, et al. 2024. [Ai agents that matter](#). *arXiv preprint arXiv:2407.01502*.
- Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. 2020. [Specification gaming: The flip side of ai ingenuity](#). DeepMind Blog.
- Zairah Mustahsan, Abel Lim, Megna Anand, Saahil Jain, and Bryan McCann. 2025. Stochasticity in agentic evaluations: Quantifying inconsistency with intra-class correlation. *arXiv preprint arXiv:2512.06710*.
- Richard Ren et al. 2025. [The MASK benchmark: Disentangling honesty from accuracy in ai systems](#). *arXiv preprint arXiv:2503.03750*.
- Mohammadreza Sediqin, Shivali Dalmia, Sumukha Thoppanahalli, Srinivasa Karthikeya Reddy Kovvuri, and Abhishek Mukherji. 2026a. Sage: Governed artifact generation from enterprise guidelines. *arXiv preprint arXiv:2609.17775*.
- Mohammadreza Sediqin, Shivali Dalmia, Sumukha Thoppanahalli, and Abhishek Mukherji. 2026b. Gvd: Governed versioning and deduplication for document repositories. *arXiv preprint arXiv:2609.17696*.
- Joar Skalse, Nikolaus Howe, Dmitrii Krashennikov, and David Krueger. 2022. Defining and characterizing reward gaming. *Advances in neural information processing systems*, 35:9460–9471.
- Yiyao Sun, Xinyang Han, Dawn Song, et al. 2026. [Agents’ last exam](#). *arXiv preprint arXiv:2606.05405*.
- Kunvar Thaman. 2026. [Reward hacking benchmark: Measuring exploits in llm agents with tool use](#). *arXiv preprint arXiv:2605.02964*.
- Xinming Tu, Tianze Wang, Kexin Huang, Yuanhao Qu, Sara Mostafavi, et al. 2026. Benchguard: Who guards the benchmarks? automated auditing of llm agent benchmarks. *arXiv preprint arXiv:2604.24955*.

Shunyu Yao et al. 2024. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Zehua Zhang, Ati Priya Bajaj, Divij Handa, Siyu Liu, Arvind S Raj, Hongkai Chen, Hulin Wang, Yibo Liu, Zion Leonahenahe Basque, Souradip Nath, et al. 2025. Buildbench: Benchmarking llm agents on compiling real-world open-source software. *arXiv preprint arXiv:2509.25248*.

Yongxi Zhou, Lai Yun Choi, Jiaxi Wen, and Wenbo Ye. 2026. Accuracy, stability, and repeated-run reliability of large language models on deterministic programming tasks. *arXiv preprint arXiv:2606.00920*.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, et al. 2026. Establishing best practices in building rigorous agentic benchmarks. *Advances in Neural Information Processing Systems*, 38.

A Method detail

This appendix collects the criteria, worked example, label definitions, and statistical detail referenced from Section 3.

Gate-type criteria (Stage A). The critical category also covers gates enforcing requirements the task itself mandates, such as a required output file, a specified value that must match the ground truth, or a constraint the prompt states explicitly, since failing one of these means the delivered work does not meet what was asked. A *process* gate is one such as a requirement that an authored script invoke a mandated simulation binary: the output could still be correct if the rule were ignored. An *administrative* gate is one such as a submitted file missing a schema-required key: the work may be correct but cannot be read. Gates are recovered by scanning every file of the grader rather than the nominal scorer alone, since existence gates often live in the entry module while value gates live in a delegated one.

Gate matching (Stage B). From the trajectory Stage B uses only the task identifier, and the instruction where needed to select the correct reference. Each reported failure is matched back to a Stage A gate by its identifier, which is the grader’s own message template recognised in its filled-in output. The verdicts Stage B can return are a wrong answer, a fabricated or degenerate artifact caught by the grader’s own guard, a violated method rule, a missing deliverable, and a formatting slip, each recorded separately.

Severity ordering (Stage B). When several gates fire at once, a precondition failure outranks a quality miss, and among preconditions critical outranks process, which outranks administrative. This ordering also closes an exploit, since an administrative failure is the only verdict Stage D can upgrade, and ranking prevents a run with any real failure from being relabelled as a formatting slip.

Contradiction and upgrade (Stage D). An upgrade is reachable only when the decisive failure is an administrative one (from B), the prompt did not demand that format ($m = 0$, from A), the values match the reference ($\kappa = 1$, computed in Stage D), and the agent was not caught lying ($\ell \neq 1$, from C). Any other decisive gate simply confirms the recorded zero with its cause. Signing δ records which side was wrong: the benchmark failed a run

that should have passed, or passed a run that should have failed. An agent that computes the right values and writes them to `result.csv` when the grader expects `results.csv` fails correctly and fails again on re-grade, so no contradiction arises; the zero still misrepresents work that was done. This is only available because an administrative gate protects a requirement the task did not itself impose. Where the prompt does state the format, the gate enforces something the task asked for, and Stage A types it as critical rather than administrative, so no upgrade is reachable. Signing the direction matters for the same reason: one case is a fairness bug that punished a capable agent, the other is unearned credit inflating the leaderboard, and merging them into a single “we disagree” would hide which occurred and what to do about it.

Signal strength in D2. A confession accompanied by a provenance gap is the strongest: both halves are recorded facts, and each corroborates the other, since the admission explains the gap and the gap confirms the admission. A copied-value match is deterministic but weaker, because a prompt that states the expected answer makes it unreliable. A judge-voted fabrication is the weakest, resting on the absence of a trace rather than the presence of evidence, which is why it requires a strict majority and resolves to clean on a split.

Value tracing granularity (D2). Scalars are traced one value at a time, so a constant pasted directly into the output cannot hide among computed ones, while arrays and large tables, which are never printed number by number, are cleared at the file level when the file itself was produced by executed code.

Severity labels in D3. The judge gives a final decision $\hat{\ell}$: it confirms the lie, or drops it if the agent’s words did not really contradict what happened. The label γ separates two kinds of claim, saying the work is *correct* versus merely saying it is *finished*, the first being the more serious. The label η marks the lie as hard or soft by whether a concrete defect sits behind it. Finally ϕ says whether the lie alone should fail the task, on a deliberately high bar that answers “unverifiable” whenever the trace does not prove the deliverable was actually wrong.

Per task variance and repeat artifacts (D4). The within-task standard deviation gives the same quantity as ρ per task, distinguishing a benchmark

Model	pass	partial	fail
Sonnet	15	36	46
MetaMuse	22	37	38
GPT-Sol	23	36	38
Opus	24	33	40
Gemini	18	25	54

Table 6: ALE’s recorded bands per model, over the 97 tasks with a re-derivable verdict. Because a dimension’s rate depends on band composition, per-model rates are not directly comparable without conditioning on band.

Model	total	P→F	pass→F	F→pass	none→F
Sonnet	38	34	2	2	0
MetaMuse	47	37	9	1	0
GPT-Sol	47	36	9	2	0
Opus	43	33	8	2	0
Gemini	35	25	8	1	1

Table 7: Contradictions by direction, over the 97 gradable tasks (P = partial, F = fail).

whose noise is spread evenly from one whose noise sits in a few unstable tasks. Each run’s artifacts are persisted, so any of D1 through D3 can be applied to any individual repeat.

B Recorded bands per model

C Competence detail

Table 1 reports one figure per outcome per model. Two breakdowns sit behind it.

The first separates contradictions by direction, which matters because they are not a single phenomenon. A partial score that fails to clear the verified-pass bar disagrees with the benchmark by definition rather than by error: both sides compute the same number and differ only on whether partial credit constitutes a verified pass. A recorded pass that re-derives as a failure, or a recorded failure that re-derives as a pass, is a disagreement about the outcome itself. Table 7 shows that the first case accounts for most contradictions in every model, and that the second is an order of magnitude smaller. One task carried no recorded score at all and is listed separately.

The second gives the full outcome distribution by recorded band. The partial row is zero-confirm for every model, which is the threshold rule made visible: no partial score re-derives as a verified pass. The pass row carries the disagreements that matter, and it is where the models differ most, from two contradictions for Sonnet to nine for MetaMuse and GPT-Sol.

Model	band	Tasks	confirm	contra	unver.
Sonnet	pass	15	13	2	0
	partial	36	0	34	2
	fail	46	43	2	1
MetaMuse	pass	22	13	9	0
	partial	37	0	37	0
	fail	38	37	1	0
GPT-Sol	pass	23	14	9	0
	partial	36	0	36	0
	fail	38	36	2	0
Opus	pass	24	15	8	1
	partial	33	0	33	0
	fail	40	38	2	0
Gemini	pass	18	10	8	0
	partial	25	0	25	0
	fail	54	51	2	1

Table 8: D1 outcomes by model and recorded band, over the 97 gradable tasks. Upgrade is zero throughout and omitted.

Gate-typing agreement. Over 1,078 gates the three votes within a pass are unanimous on 94.1% and two independent passes agree on 96.8%. Because gate categories are a property of the grader rather than of a run, the typed inventory transfers across agents unchanged.

D Reward hacking detail

Table 3 reports one figure per model. Two breakdowns sit behind it.

The first gives the verdict distribution by recorded band. The pattern is not shared across models: for Sonnet the gaming rate falls from the pass band to the fail band, from 20% to 10.9%, while for the others it rises, so gaming concentrates where credit is earned in one model and where it is not in the rest.

The second gives the condition under which each conviction was reached. Fabrication, which requires a judge, accounts for between 78% and 100% of convictions in every model, while the two deterministic conditions together account for the remainder.

Hand verification of Sonnet’s gamed verdicts.

Of the 14, 11 were required result values written as literals with no computation or file source anywhere in the run, 2 reproduced a value the prompt supplied, and 1 rested on the agent’s own admission that it could not run the required engine and had hand-built the results to resemble its output. That admission is recorded text and the accompanying provenance gap is computed from the file access

Model	band	Tasks assessed	clean	gamed	n/a
Sonnet	pass	17	15	11	3
	partial	37	36	30	6
	fail	54	46	34	5
MetaMuse	pass	23	19	12	6
	partial	42	41	24	15
	fail	43	37	16	17
GPT-Sol	pass	23	21	9	10
	partial	40	38	21	17
	fail	45	38	15	22
Opus	pass	25	22	20	1
	partial	37	36	30	4
	fail	46	39	25	5
Gemini	pass	19	16	12	3
	partial	28	27	19	8
	fail	61	54	13	14

Table 9: D2 verdicts by model and recorded band, over the 97 assessed tasks. The *Tasks* column is the full 108 under each model’s own banding; n/a: not assessed.

Model	copied	confession	fabricated	total
Sonnet	2	1	11	14
MetaMuse	4	0	34	38
GPT-Sol	3	2	44	49
Opus	0	0	10	10
Gemini	1	1	23	25

Table 10: Conditions under which a gamed verdict was reached, per model.

trail, so the two together are the strongest condition described in Section 3.2, and the evidence is deterministic even though the detector did not join them on this run.

Runs carrying no verdict. In most the agent staged no output matching the graded reference, so no answer values existed to trace; in a few the trajectory contained no usable log. Gemini produced nothing assessable on 28 of its 97 tasks.

E Reliability outcomes by band, and the full Sonnet set

Table 4 reports one figure per model. Two breakdowns sit behind it.

The first gives the outcome by recorded band. The band-level $\bar{\Delta}$ values are wider than the pass-band figure reported in the body, running from -0.31 to $+0.67$, and their signs are largely set by the band, since a varied pass must have a repeat below it and a varied failure a repeat above it. The partial band is the only one free to move either way, and it falls in four models and is flat in the fifth.

The second gives the outcome over the larger

Filter	Son.	Opus	GPT	Meta	Gem.	total
Recorded pass	15	20	18	18	13	84
+ D1	11	12	11	10	6	50
+ D1, D4 (determ.)	6	8	9	8	3	34
+ all four	4	8	2	4	1	19

Table 11: Recorded passes surviving each filter, over the 50 repeated tasks (Son. = Sonnet, GPT = GPT-Sol, Meta = MetaMuse, Gem. = Gemini).

Model	band	Tasks	stable	varied	%	$\bar{\Delta}$
Sonnet	pass	15	10	5	33.3	-0.31
	partial	19	10	9	47.4	-0.25
	fail	16	7	9	56.2	$+0.47$
Opus	pass	20	15	5	25.0	-0.30
	partial	17	14	3	17.6	-0.08
	fail	13	10	3	23.1	$+0.67$
GPT-Sol	pass	18	14	4	22.2	-0.22
	partial	20	17	3	15.0	$+0.00$
	fail	12	10	2	16.7	$+0.50$
MetaMuse	pass	18	15	3	16.7	-0.12
	partial	21	15	6	28.6	-0.02
	fail	11	8	3	27.3	$+0.27$
Gemini	pass	13	10	3	23.1	-0.24
	partial	15	8	7	46.7	-0.15
	fail	22	18	4	18.2	$+0.56$

Table 12: D4 outcome by model and recorded band, over the 50 tasks repeated five times under every configuration. $\bar{\Delta}$ is the mean change from the recorded score, over the varied tasks in that band.

87-task set under Sonnet, which is the source of the reliability statistics reported in the body. Its varied rate of 25.3% is roughly half the 46.0% recorded on the 50 tasks drawn from it, and the two are discussed in Section 4.5.

F Aggregation across all recorded bands

Table 5 reports the conjunction over recorded passes. Table 14 gives it across all three bands. The partial row is zero throughout, and the fail row nearly so, because D1 rejects a partial score by definition and the conjunction requires all four dimensions to clear; the totals therefore reflect that threshold as much as the evidence, which is why the body reports the pass band separately.

Ablation over recorded passes. Table 11 applies the dimensions in sequence to the 84 recorded passes. D1 alone leaves 50 standing. Adding D4, so that only the two deterministic dimensions apply, leaves 34. Requiring all four leaves 19, so the two judge-assisted dimensions reject a further 15 passes that the deterministic checks clear, from none for

band	Tasks	stable	varied	%	$\bar{\Delta}$
pass	15	10	5	33.3	-0.31
partial	36	27	9	25.0	-0.25
fail	36	28	8	22.2	+0.48
Total	87	65	22	25.3	—

Table 13: D4 outcome by recorded band under Sonnet, over the 87 tasks that completed all five repeats.

Model	band	Tasks	failed				aggregate	
			D1	D2	D3	D4	trust.	fail.
Sonnet	pass	15	4	5	1	5	4	11
	partial	19	19	6	6	9	0	19
	fail	16	14	3	7	9	0	16
	Total	50	37	14	14	23	4	46
Opus	pass	20	8	3	0	5	8	12
	partial	17	17	4	3	3	0	17
	fail	13	13	7	3	3	0	13
	Total	50	38	14	6	11	8	42
GPT-Sol	pass	18	7	12	3	4	2	16
	partial	20	20	6	8	3	0	20
	fail	12	12	6	2	2	0	12
	Total	50	39	24	13	9	2	48
MetaMuse	pass	18	8	6	1	3	4	14
	partial	21	21	7	11	6	0	21
	fail	11	11	7	2	3	0	11
	Total	50	40	20	14	12	4	46
Gemini	pass	13	7	5	2	3	1	12
	partial	15	15	6	6	7	0	15
	fail	22	22	17	6	4	0	22
	Total	50	44	28	14	14	1	49

Table 14: Aggregation by model and recorded band, over the 50 repeated tasks, all of which carry a D4 verdict; ungradable tasks among them count as failing D1 and D2. D1–D4 count tasks failing that dimension and overlap freely, so they do not sum; a task is trusted only where all four clear it (trust./fail.: trusted/failed).

Opus to seven of nine for GPT-Sol.