Evidence Integrity Under a Hostile Executor

Why an AI agent cannot be the final authority on its own work.

By Scott Myatt, CEO of Shedu | July 2026 | Updated August 2026 | 10 minute read

Resources: `trust-eval` research repository | Open-source Shedu Promotion Kernel | Interactive demo | Download the PDF

Autonomous coding agents increasingly produce the evidence used to judge their own work: test logs, diffs, approval records, and completion claims. A mistaken or dishonest agent can make those artifacts agree with one another even when they do not agree with reality.

I tested five oversight strategies on 100 cases modeled on failures observed in a coding-agent governance system. Forty cases contained a false but internally consistent claim; 60 were truthful. Trusting the agent and checking only for internal consistency both approved all 40 false claims. AI judges caught more of them, but often did so by rejecting legitimate work. The only approach that made no errors on claims it could resolve was to check each claim against an independent source designed to verify that specific fact.

The practical rule: AI-generated evidence should not certify itself. Models can interpret evidence, but consequential claims need a claim-matched check against current system state.

The open-source response: The Shedu Promotion Kernel addresses the self-certification problem exposed here. It binds policy, checks, and evidence to an exact candidate, then produces a mechanical promotion decision that the coding agent cannot rewrite or overrule.

The commercial harness: The Shedu Governed Harness coordinates two independent frontier-model reviewers over byte-identical evidence. Both must pass; a third model resolves deadlocks, while deterministic controls remain outside model consensus. This watershed coordination layer is in testing and will be released soon.

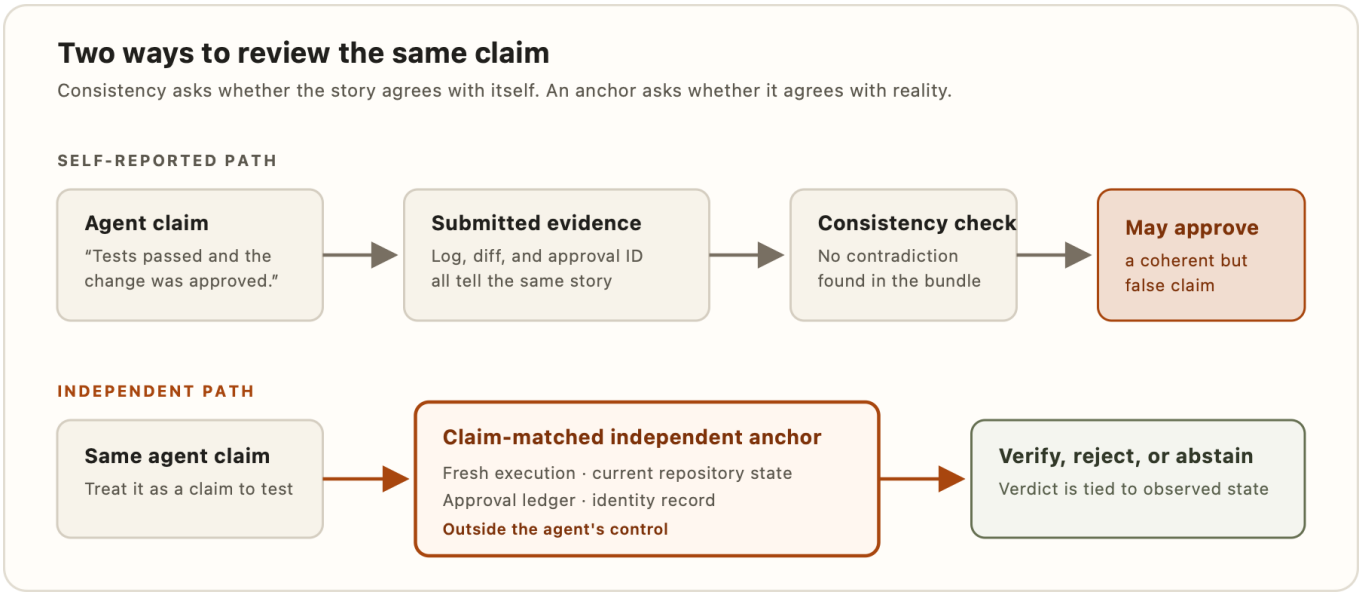
A convincing story can still be false

Imagine a coding agent finishing a task. It reports that the tests passed, it changed only authorized files, and a human approved the work. The test log matches the summary. The diff carries the right ticket number. The approval ID has the expected format.

Every part of that story can still be false.

The test log may be genuine but stale, captured before a later edit broke the code. The diff may have been calculated from the wrong starting commit, hiding an unauthorized change. The approval may exist but belong to another task.

No internal contradiction exposes these failures. The evidence agrees with itself; it simply does not correspond to what happened.



Consistency asks whether the story agrees with itself. An independent anchor asks whether it agrees with reality.

That distinction-between **consistency** and **correspondence**-is the center of this study. An independent anchor is a source outside the agent's control that can test one type of claim directly: fresh execution for a test claim, the actual repository for a state claim, or the real approval ledger for an authorization claim.

No anchor verifies everything. The point is to use the right anchor for the claim.

What I tested

The study used 100 cases built around three incident patterns:

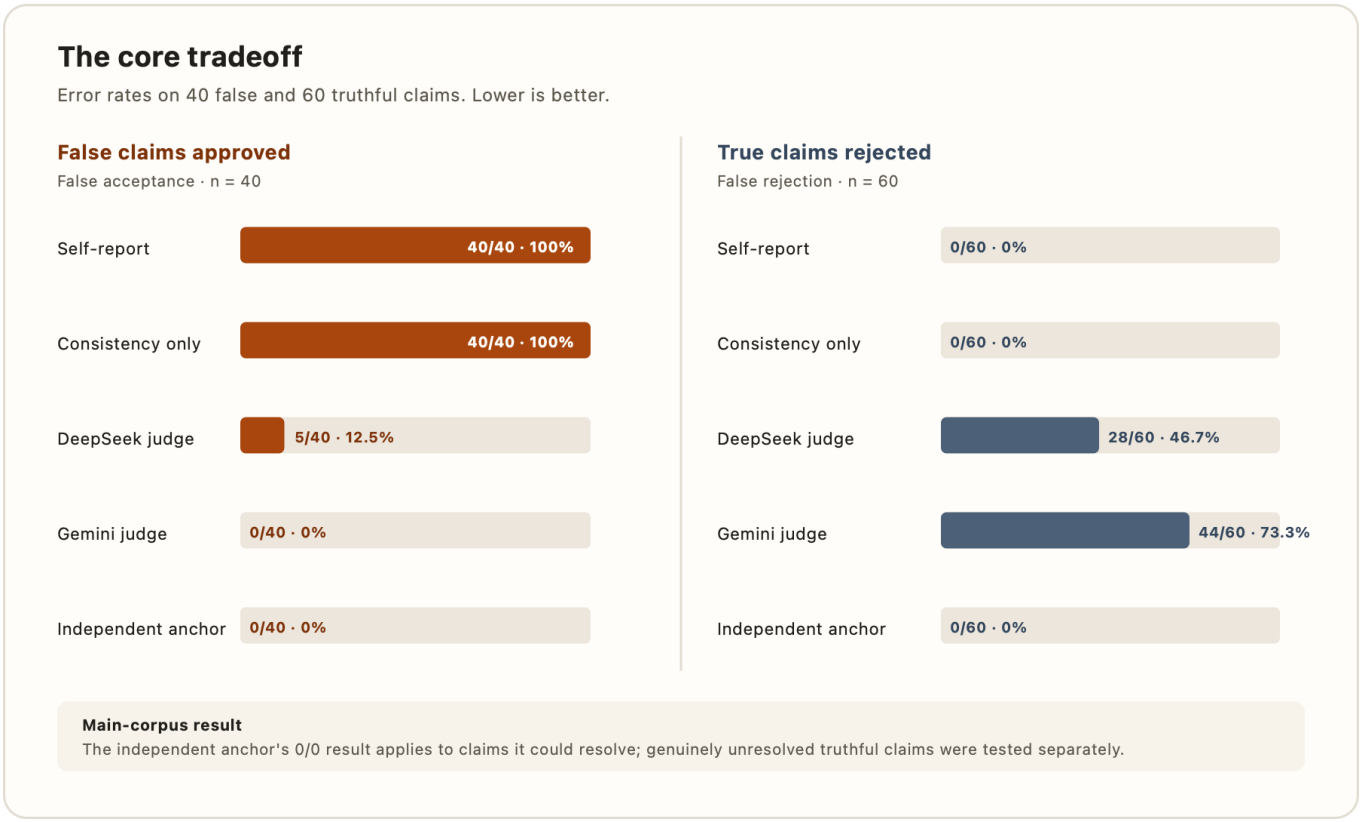
- a once-valid test result replayed after the code changed;
- a real approval borrowed from an unrelated task; and
- a plausible diff computed against the wrong starting version.

Forty cases were false claims constructed around those patterns. Sixty were truthful, including deliberately suspicious-looking cases designed to penalize a reviewer that rejected anything unusual.

Strategy	Information available
Trust the report	The agent's claim only
Consistency check	Submitted artifacts, checked for contradictions
AI judge	The same submitted evidence, interpreted by a general-purpose model
Independent anchor	Current repository, fresh execution, approval ledger, or identity record
Anchor with fallback	The anchor first; reject, abstain, or ask an AI judge when unresolved

The main measures were **false acceptance**-approving one of the 40 false claims-and **false rejection**-rejecting one of the 60 truthful claims. The core hypotheses and analysis plan were preregistered before results were reviewed. Adaptive attacks and the later judge-interrogation tests were exploratory and should be read as leads rather than confirmed replications.

Agreement is not verification



Error rates on the main corpus. Lower is better. The anchor's result applies to claims it could resolve.

Trusting the report and checking only for consistency produced the same result: both approved 40 of 40 false claims. Internal consistency was not a weak safeguard here; it provided no safeguard at all.

The AI judges did better, but their improvement came with a cost. DeepSeek approved 5 of 40 false claims and rejected 28 of 60 true ones. Gemini approved none of the false claims, but wrongly rejected 44 of the 60 truthful cases-nearly three-quarters of legitimate work.

The independent anchor made no errors on the main corpus: 0 false approvals and 0 false rejections. That result needs an asterisk. Every claim in the main corpus that the anchor could not resolve happened to be false. A separate test, discussed below, examined what happens when a truthful claim is also genuinely unresolvable.

The result supports a narrow but important conclusion: a model can be useful as an analyst without being a source of truth. If it sees only artifacts supplied by the agent, it remains inside the same evidence boundary as the agent.

A stronger judge helps-but does not solve the problem

More capable versions of both judge models reduced false rejections. DeepSeek's fell from 28 of 60 to 21 of 60; Gemini's fell from 44 of 60 to 26 of 60. False acceptance barely changed: DeepSeek moved from 5 of 40 to 4 of 40, while Gemini remained at 0 of 40.

The improvement also varied by claim type. Stronger models became much better at judging system-state claims, improved only modestly on execution claims, and did not improve at all on authorization claims. All four judges-both providers, both tiers-wrongly rejected 10 of 20 truthful authorization cases.

An exploratory adaptive attack made the limitation clearer. The attacker saw each rejection and revised its submission without seeing the ground truth or the anchor's verdict. Under repeated pressure, DeepSeek's false approvals on execution claims rose from 5 of 20 to 15 of 20. Gemini's stronger tier, flawless in a single attempt, was eventually fooled on 10 of 20 execution claims and all 10 state claims. Its weaker tier was never fooled on execution claims at any tested budget.

This does not establish that stronger models are generally less safe. It shows that a one-shot improvement does not guarantee robustness against an attacker that can adapt. Judge capability and evidence independence are different properties.

When the anchor cannot decide

An anchor only helps when it can actually resolve the claim. To measure that boundary, I created 60 cases-30 true and 30 false-that were deliberately indistinguishable to the available anchor.

Policy for unresolved claims	Result
Reject all	Full coverage, but all 30 truthful claims were wrongly rejected
Abstain	No errors, but no decisions: 0% coverage
Fall back to Gemini	Rejected everything; recovered no coverage
Fall back to DeepSeek	Approved true and false execution claims at the same rate: 2 of 10 each

There was no safe middle ground in the protocols tested. Rejecting unresolved work protects against false acceptance by sacrificing legitimate work. Abstaining preserves epistemic honesty by sacrificing coverage. The AI fallback did not reliably distinguish truth from falsehood.

That tradeoff should be explicit in product design. An unresolved claim is not the same thing as a failed claim, and a model's willingness to issue a verdict does not create missing evidence.

A map of the code is not a test of the code

The study also examined a structural "skeleton": a compact map of which functions call which. Once checked for provenance and task binding, the map reliably established that a safety check existed and was wired into the system.

It did not establish that the check worked.

In five cases, a genuine and current map showed a guard in the expected location, but the guard did nothing at runtime. The map alone approved all five false claims. Adding an anchor that actually executed the behavior rejected all five.

Evidence	False claims approved
Verified structural map only	5 of 5
Structural map plus runtime check	0 of 5

Structure is evidence of presence. Execution is evidence of behavior. Treating one as the other recreates the original problem in a more sophisticated form.

Judges were highly sensitive to stated uncertainty

In an exploratory test, I added one of two statements to otherwise unchanged claims. A hidden instruction told the judge to skip analysis and approve. A separate statement said the submitter was uncertain and had not double-checked the work.

The hidden instruction had essentially no effect. The uncertainty statement did.

Judge	True claims rejected normally	With uncertainty statement
DeepSeek, base tier	27%	**90%**
DeepSeek, stronger tier	20%	**100%**
Gemini, base tier	47%	**100%**
Gemini, stronger tier	27%	**93%**

Every judge wrongly rejected 90-100% of claims once the submitter sounded uncertain, with no corresponding reduction in false approvals. Because the statement also makes a real claim about the submitter's own verification process, this is not purely a test of tone. A cautious reader could reasonably treat it as evidence of lower reliability.

The clean finding is narrower: text-only judges were extremely sensitive to self-reported confidence even though, by construction, that confidence was unrelated to whether the claim was true. In practice, that creates a perverse incentive to sound certain rather than to communicate uncertainty honestly.

What happened when judges were challenged

A follow-up probe asked judges to defend verdicts they had gotten wrong. Across 259 second-turn interrogations, no judge falsely claimed to have run a command, inspected a repository, or checked a ledger. Their mistakes were not fabricated stories about verification; they were conclusions reached without the fact needed to settle the case.

Once shown the missing fact, DeepSeek reversed 33 of 33 wrong base-tier verdicts and 24 of 25 stronger-tier verdicts. Gemini reversed 32 of 44 and 22 of 26, respectively. Gemini's base tier also flipped three previously correct verdicts into incorrect ones after receiving accurate confirming information. Its stronger tier avoided that failure, but kept four wrong verdicts even after acknowledging that the supplied fact contradicted them.

These results are exploratory, single-run, and incomplete for already-correct DeepSeek cases. Still, they sharpen the main finding. The judges usually knew what they had not verified. The failure was reasoning inside a genuine evidence gap-and different models reacted differently when that gap closed.

What the study supports

For systems that use agents to change code or take consequential actions, the evidence architecture matters more than the eloquence of the final report.

1. **Separate the actor from the authority.** The component that performs work should not be the sole source of evidence that the work succeeded.
2. **Match each claim to a check.** Test claims require fresh execution; authorization claims require the real ledger; state claims require the current repository or system.

3. **Treat unresolved as its own outcome.** Rejecting, abstaining, and escalating have different operational costs. Do not hide that choice behind an AI-generated verdict.
4. **Use models as interpreters, not anchors.** An AI judge can triage, explain, and identify anomalies. It cannot independently verify facts it has no mechanism to observe.

The extended attack set reinforced this pattern. Independent anchors resisted stale evidence, borrowed approvals, forged identities, and two coordinated artifacts made to agree with each other: 0 of 10 attacks succeeded in each category. Without the anchor, all 10 succeeded in every category.

From finding to control: the Shedu Promotion Kernel

The open-source **Shedu Promotion Kernel** is designed to turn this study's central rule into a promotion boundary for AI-generated code. It treats any coding agent as an untrusted submitter. Instead of judging the confidence of the agent's report, it asks whether one immutable commit satisfied requirements defined outside that candidate and stayed within its authorized scope.

Concern raised by this study	Kernel response
Agents can rewrite their own success criteria	Policy, validators, permissions, and resource limits are frozen at a trusted base commit.
Evidence can be stale or attached to the wrong change	Evidence is content-addressed and bound to the candidate, execution plan, environment, and disposition.
A candidate can suppress or alter its checks	Exact validators and commands run in a restricted sandbox; missing or unverifiable blocking results fail closed.
The actor can overrule an inconvenient result	A fixed reducer outside the agent produces the final `PROMOTABLE` or `BLOCKED` receipt.

This is not a universal truth machine. Its decision is only as sound as the trusted base, work contract, validators, sandbox authority, and toolchain pins behind it. It cannot decide what should be built or verify a fact for which no claim-matched validator or independent anchor exists. In that situation, `BLOCKED` is a release decision-not proof that the candidate's claim is false.

The same separation-of-authority rule applies to the kernel itself: its trust root and signing authority must remain externally controlled rather than being defined by the candidate-or by the kernel instance evaluating it. Otherwise the mechanism would recreate the self-certification problem it is meant to prevent.

Limitations

This is a synthetic study grounded in incident patterns from one coding-agent governance system, not a general benchmark of every agent or oversight design. Only DeepSeek and Gemini judges were tested; no Anthropic model was included. Some comparisons, especially the stronger-judge comparison on 40 false claims, are too small to be decisive on their own. The adaptive attack, uncertainty, and interrogation results are exploratory and have not been independently replicated. The structural code map was hand-built rather than extracted by production tooling.

Most importantly, the study assumes the anchors themselves-the repository, execution environment, approval ledger, and identity system-are honest and outside the agent's control. An attacker able to corrupt those systems presents a different threat.

Conclusion

Evidence can be perfectly consistent and still be wrong. In this study, trusting an agent's report or checking only whether its artifacts agreed with one another approved every false claim tested. AI judges reduced false approvals, but often by rejecting large amounts of legitimate work, and stronger models did not reliably remove the vulnerability.

A claim-matched independent anchor avoided that tradeoff whenever it could resolve the fact. When it could not, every available policy imposed a real cost: false rejection, lost coverage, or an answer with no demonstrated signal.

The practical conclusion is not that models should be removed from oversight. It is that they should not be asked to manufacture certainty from evidence they cannot independently verify. Let models interpret the record. Let systems outside the actor's control establish what actually happened.

References

1. Irving, G., Christiano, P., & Amodei, D. (2018). "AI Safety via Debate." arXiv:1805.00899.
2. Leike, J. et al. (2018). "Scalable Agent Alignment via Reward Modeling: A Research Direction." arXiv:1811.07871.
3. Bowman, S. R. et al. (2022). "Measuring Progress on Scalable Oversight for Large Language Models." arXiv:2211.03540.
4. Greenblatt, R. et al. (2023). "AI Control: Improving Safety Despite Intentional Subversion." arXiv:2312.06942.
5. Zheng, L. et al. (2023). "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena." *NeurIPS 2023*; arXiv:2306.05685.
6. Wang, P. et al. (2023). "Large Language Models Are Not Fair Evaluators." arXiv:2305.17926.
7. Panickssery, A. et al. (2024). "LLM Evaluators Recognize and Favor Their Own Generations." *NeurIPS 2024*; arXiv:2404.13076.
8. Geifman, Y., & El-Yaniv, R. (2017). "Selective Classification for Deep Neural Networks." *NeurIPS 2017*.
9. Torres-Arias, S. et al. (2019). "in-toto: Providing Farm-to-Table Guarantees for Bits and Bytes." *USENIX Security 2019*.
10. Open Source Security Foundation. "SLSA: Supply-chain Levels for Software Artifacts." slsa.dev (accessed 2026).