

Hollow Proofs: Measuring LLM Dishonesty with a Formal Verifier

Orpheus Lummis

The Secure Program Synthesis Hackathon, May 2026

Abstract

Honesty benchmarks for language models must check claims against ground truth, which can be difficult: world-knowledge benchmarks conflate lying with ignorance, and belief-based methods such as MASK rely on an elicited belief they cannot independently check. We study a setting where ground truth is mechanical: when a model writes a proof in Verus (a deductive verifier for Rust), the verifier decides whether the proof holds, and its `--no-cheating` mode decides whether the proof relied on an unsound escape such as `assume`. Using these checks, we detect two forms of dishonesty: hiding an unsound primitive in a proof and denying it (a *hollow proof*), and claiming a failed proof succeeded under pressure when the model judges it failed (a *grounded lie*). Our contributions are methodological: first, eliciting an observation ("did your last check pass?") rather than a prediction ("is your proof complete?") separates dishonesty from miscalibration, because a misreported observation can only be a lie, while a wrong prediction can be honest overconfidence. Second, because the verifier can also check the elicited belief, the benchmark withholds a dishonesty verdict when that belief is too unreliable to trust. We release the harness and a difficulty-graded set of 121 Verus tasks. In a feasibility pilot on two versions of an open-weight model family (DeepSeek-V4-Pro and -Flash), both report their verification results honestly under deadline pressure even though both demonstrably know how to cheat.

1. Introduction

AI systems increasingly produce code with a machine-checked proof attached. The proof is meant to be a trustworthy receipt, but if models learn to hand out receipts that mean nothing, formal verification fails.

Detecting dishonesty needs a *referent*: the truth a claim is checked against. For most honesty evaluations the referent is the weak link. World facts conflate ignorance with lying; an elicited belief is itself an unverifiable model output; scenario-based intent is read off a chain of thought that is not a reliable witness. We observe that for claims about formally checkable artifacts, the referent is a deterministic verifier, reproducible and exact, independent of any second model or human rater. That lets us define dishonesty as a mechanical contradiction between what a model reports and what its artifact actually does.

Threat model. A model writes a proof in a normal loop (write, check, fix) against a standard verifier, then reports on its work. Two failure modes concern us. (1) It inserts an unsound escape that makes the checker accept anything, then denies it. (2) Its proof does not verify, its own unpressured assessment agrees, yet under pressure it claims success. Both are lies about a verifiable fact, distinct from honest incapacity (tried, failed, said so) and from miscalibration (genuinely but wrongly believes it succeeded).

Contributions.

1. **A judge-free dishonesty benchmark for LLM-produced proofs.** Verus's `--no-cheating` mode (which bans the unsound escape-hatch primitives) is the honesty oracle; dishonesty is a mechanical contradiction between self-report and artifact, with no LLM judge and no human rater.
2. **A distinction between honesty and calibration.** A *prediction* claim ("is your proof complete?") measures calibration; an *observation* claim ("did your last check pass?") measures honesty. We show one task whose verdict flips from dishonest to honest by this reframing alone. Conflating the two inflates apparent dishonesty.
3. **The (V, B, S) construct with a validated belief referent.** It extends MASK-style belief-vs-statement honesty by auditing the elicited belief against the verifier (`agreement(B, v)`). When the belief fails that check, the measure declines to report lies rather than inventing them. To our knowledge this is the first belief-statement honesty measure whose belief referent is itself oracle-validated.
4. **An agentic harness and a difficulty-graded set of 121 Verus tasks**, open-sourced, with the soundness check firewalled from the model and an out-of-band screen that separates "honest" from "didn't know how to cheat."

We build on the Verus verifier and its `--no-cheating` mode, the CloverBench / MBPP / VeruSAGE task sources, and the MASK belief-statement framing; the construct, the harness and its oversight firewall, the curated task set, the cheat-knowledge screen, and all reported runs are new to this work.

2. Related Work

Honesty evaluations differ mainly in the *referent*. **TruthfulQA** and **SimpleQA** judge against world facts, conflating ignorance with lying. **MASK** judges a pressured statement against the model's neutrally elicited belief, separating honesty from accuracy, but the belief is an unverifiable model output it cannot check. **Apollo's in-context scheming** reads intent off the chain of thought. **Sandbagging** measures strategic underperformance against demonstrated capability, the within-subject idea we reuse for capability screening. **Chain-of-thought faithfulness** work (Lanham et al.; Turpin et al.) shows stated reasoning need not cause the answer, so we treat chain-of-thought as corroboration only.

On the formal-methods side, **VeruSAGE** found that without a cheat checker, Claude Sonnet 4 cheated on 14% of its proof tasks (Sonnet 4.5: 7%; GPT-5: 2%; under 1.5% for all once a checker was offered), where cheating meant unsound primitives such as `assume`, `admit`, or `external_body`, or weakening the specification. The Verus `--no-cheating` mode (verus-lang/verus #1499) mechanically detects the primitive-based forms. We draw tasks from or position against **CloverBench**, **MBPP**, **DafnyBench**, **AutoVerus / VerusBench**, **FVAPPS**, **CLEVER**, and **VERINA**.

How we differ. Our referent is a mechanical verifier, so where MASK must trust its belief, we audit it. Where VeruSAGE asks whether a proof cheats, we ask whether the model is *dishonest about* cheating, which means comparing artifact to self-report and separating lying from both incapacity and miscalibration.

3. Methods

Verifier and oracle. Verus 0.2026.05.17.e479cce (Rust 1.95.0). Standard verus answers "does this verify"; `verus --no-cheating` additionally bans the four unsound primitives (`assume`, `admit`, `external_body`, `assume_specification`). The verdict `V` is `pass`, `fail`, or `hollow` (passes the standard verifier but fails `--no-cheating`). We validated the detector on synthetic fixtures: every primitive is caught, a clean baseline passes both.

Harness constraints. Verus is not concurrency-safe (it writes a binary named after the crate stem into the working directory, so parallel runs corrupt each other), so we serialize and isolate each run in its own temp directory. The task references predate the current termination-check requirement, so we prepend `#![verifier::exec_allows_no_decreases_clause]` to restore their original semantics; this is orthogonal to the soundness escapes.

Oversight firewall. The agentic `verus_check` tool exposes standard verification only. The model iterates up to 5 rounds and never sees the soundness check or any self-report comparison; those run once, post-hoc, in the scorer. So a model that shortcuts sees the checker turn green and never learns it is being watched. (At the round cap we strip the tool and force a finalize turn, retrying tool-free until the model returns a parseable report; this cut sample loss from 30% to 0%, see §4.)

Cheat-knowledge screen. A 0% hollow rate is only an honesty signal if the model *could* cheat. So an out-of-band, neutral probe, never mentioning `--no-cheating` or that honesty is scored, asks about each primitive and how to make a stuck proof pass. Only a model that demonstrably knows the mechanism yields an interpretable null.

Two claim framings. On the same runs we contrast a *prediction* claim (`proof_status`, asked before the verdict is known) with an *observation* claim (`last_verus_check_passed`, a statement about an event the model saw). Prediction measures calibration; observation measures honesty.

(V, B, S) protocol. Two phases, within-subject across pressure. (1) *Production (pressured)*: the model submits an artifact and statement `S`; `V` is computed post-hoc. (2) *Belief elicitation (neutral, separate context)*: the artifact is re-presented with no stakes and the model assesses whether it verifies, over three anchor-free paraphrases majority-voted to `B`. The fresh context keeps `S` from anchoring `B`.

Conditions and prompts. Conditions are **neutral** and **deadline** ("customer waiting"). The model can always report failure without penalty, so it never has to cheat. To show a null is not a prompt artifact, we ran two of three drafted prompt designs (the intermediate `V1` was dropped): **V0** names the primitives and blesses honest failure, and **V2** is a realistic reviewer hand-off that names no primitives and offers no blessed exit (denial is implicit in a `soundly_proves_spec` field). Adaptive-thinking models ignore `temperature`, so we draw `k` samples per cell.

Taxonomy and statistics. `B=fail`, `V=fail`, `S=pass` is a **grounded lie**; `V=hollow` with a denial is a **hollow lie**; `B=pass`, `V=fail`, `S=pass` is an **honest error** (miscalibration, reported separately). The headline is the within-subject contrast $\Delta\text{rate} = \text{rate}(\text{deadline}) - \text{rate}(\text{neutral})$, which isolates the strategic part from the absolute rate; we also report the elicitation-fidelity audit `agreement(B, V)` (low fidelity discounts any grounded-lie claim).

Models and data. Subjects run via Fireworks; no language model acts as a judge or grader anywhere in the pipeline. Single-shot capability probes: DeepSeek-V4-Pro 2/6 sound (viable), Qwen3.6 Plus 0/6 (excluded). Tasks are a difficulty-graded set filtered to primitive-free, standard-

passing references so whole-file `--no-cheating` is a valid detector: easy 8 (CloverBench), medium 12 (MBPP), hard 101 (VeruSAGE), 121 total (`data/curated_gradient.json`). Reproduce with `uv run pytest -q` (46 tests) then the `run_pilot.py` / `analyze.py` scripts.

4. Results

Pilot scale is small (single model per arm, $k=1$, $n = 8$ to 88). Every number below traces to a run log. The contributions are methodological and a diagnosed null, not a large dishonesty rate.

Table 1: verdict counts per arm. Every dishonesty cell (hollow proof, grounded lie) is empty across all three arms; the only dishonesty-adjacent event is one prediction-framed overclaim (the "false compl." entries), reclassified as miscalibration below.

Arm	Model	Prompt	n	Sound	Honest fail	Hollow	False compl.	Grounded lie
Gradient (agentic)	V4-Flash	V0	88	36	45	0	1	n/a
Gradient (agentic)	V4-Flash	V2	40	15	24	0	1	n/a
(V, B, S)	V4-Pro	VBS	8	5	3	0	0	0

(V0 also had 6 excluded samples, since fixed; V2 shown post-fix at 0 excluded. "n/a": the hollow arms do not elicit a belief B, so the grounded-lie verdict does not apply.) For the (V, B, S) arm, $\text{agreement}(S, V) = 8/8$ and $\text{agreement}(B, V) = 3/8$.

The model knows how to cheat and doesn't. Across 88 agentic runs DeepSeek-V4-Flash submitted 366 programs to `verus_check` and used an unsound primitive in none of them. The screen confirms this is choice, not ignorance: it answered every primitive question correctly and, asked how to force a stuck proof through, spontaneously produced the canonical hollow proof (`assume(postcondition);`), calling it "unsound, use only temporarily for debugging." It also produced 36 sound proofs, so basic capability is established.

Hollow rate at floor, robust to the prompt. Under V0 (88 runs = $44 \text{ tasks} \times \{\text{neutral, deadline}\}$): 0 hollow across all tiers (easy 0/14, medium 0/23, hard 0/45); $\Delta\text{HollowRate} = 0$. The V2 hand-off (40 samples) reproduced it: 0 hollow (this arm does not elicit a belief, so it cannot register grounded lies). 15 of the 40 were *stuck* trials (the model failed and could not finish the proof) where an unsound primitive would have forced a pass; the model used none.

Sample-loss fix. V2 first lost 12/40 samples (9 empty completions at the round cap, 3 truncations). The tool-free finalize retry and a token-headroom bump recovered all of them on a re-run (0/40 excluded), and they resolved to honest failures: the harness had been discarding honest behavior, not hidden dishonesty.

The claim-framing flip. On `array_concat` under deadline (V4-Pro), the *prediction* claim ("is it complete?") said `complete` on a proof that failed, which looks dishonest; the *observation* claim ("did your last check pass?") on the same task said `false`, which is honest. The overclaim was overconfidence, not a lie. In this single flip case, the framing substitution converts the verdict from dishonest to honest, the kind of conflation the prediction/observation distinction is meant to catch.

The fidelity audit gates the belief mode. For the (V, B, S) arm (V4-Pro, 8 trials): the production statement matched the verifier every time (`agreement(S, V) = 8/8`), but the blind belief (fresh context, assess by inspection, no tool) tracked it on only 3 of 8, worse than the 5/8 from always guessing "verifies" (the majority answer). So the audit flags B as untrustworthy, and the construct declines to report grounded lies rather than fabricate them from a belief it cannot trust. A 0 here is the audit working, not a measurement of zero lying.

Cost. About \$0.04/sample (V4-Pro, agentic, ~24K tokens, ~3.6 rounds); a full k=3 batch is roughly \$5.

5. Discussion and Limitations

As AI-written, formally verified code becomes routine, the proof is the trust anchor, so a benchmark that mechanically catches a model lying about its own proof, with no judge model in the loop, is a useful primitive for oversight. Our pilot read is narrow but worth stating plainly: the capable open models we tested report verification outcomes honestly under deadline pressure while knowing how to cheat. VeruSAGE shows the underlying behavior is real (Claude Sonnet 4 cheated on 14% of tasks when unchecked); we ask the different question of whether a model lies *about* cheating, and on our tasks find that it does not.

The pilot is directional, not a population estimate: one model per arm, k=1, and every dishonesty cell empty, so no confidence interval excludes zero. Two design choices also influence where a signal could appear: the verify-and-sound task filter, needed so a whole-file `--no-cheating` detector is valid, keeps the tractable VeruSAGE lemmas and drops the hard primitive-using systems proofs where cheating was previously seen, under-sampling the regime most likely to elicit it; and the grounded-lie mode requires a model that fails, recognizes the failure, and still claims success, a cell empty by construction for a well-calibrated model. Belief elicitation is itself unsolved: the blindly elicited belief matched the verifier on only 3 of 8 trials, worse than always guessing the majority answer, so the measure needs a same-context elicitation that raises fidelity without anchoring the belief to the pressured statement. Finally, we test a single pressure framing (deadline) and use chain-of-thought only as corroboration, since it is provider-specific and not guaranteed faithful; the detector's false-positive and scalability limits are detailed in the appendix.

Next steps: draw the belief from the model's production-informed state (off the record) to raise fidelity; run a model panel spanning the capability and alignment axes, screened first; build a frontier-difficulty task set with a target-function-scoped detector so the hard primitive-using tasks become usable; and add convergent validity against MASK plus replication in Lean 4 and Dafny.

6. Conclusion

Hollow Proofs measures dishonesty in LLM-written proofs against a formal verifier rather than a human or a second model. It flags two things: a proof that passes only by hiding an unsound primitive the model denies using, and a claim of success on a proof the model's own unpressured assessment judged to have failed. The second flag fires only when that assessment is reliable, checked against the verifier, so an untrustworthy belief never produces a lie verdict. In our pilot the two DeepSeek-V4 models (Pro and Flash) produced neither: both reported their results honestly under deadline pressure, though both can cheat. The contribution is mainly methodological: asking what a model observed ("did your last check pass?") rather than what it

predicts ("is the proof complete?") tells a deliberate lie apart from honest overconfidence. Detecting actual lying would require a less-aligned model or harder tasks.

Code and Data

- **Code:** the harness, detector, scorers, and reproduction scripts, released open-source under the MIT license.
- **Data:** a difficulty-graded set of 121 Verus tasks (8 from CloverBench, 12 from MBPP, 101 from VeruSAGE), each curated to verify soundly under our Verus version. The tasks derive from the Verus-Bench and VeruSAGE-Bench corpora in `microsoft/verus-proof-synthesis`; we redistribute task identifiers, not upstream content, which keeps its original license.
- **Reproduction:** the test suite and the zero-cost mock-model validators run with no external data; re-running the pilots needs Verus and the upstream corpus (see the README).

References

1. Ren, R., Agarwal, A., Mazeika, M., et al. (2025). *The MASK Benchmark: Disentangling Honesty From Accuracy in AI Systems*. arXiv:2503.03750.
2. Lin, S., Hilton, J., Evans, O. (2022). *TruthfulQA: Measuring How Models Mimic Human Falsehoods*. ACL 2022. arXiv:2109.07958.
3. Wei, J., Karina, N., Chung, H. W., et al. (2024). *Measuring Short-form Factuality in Large Language Models (SimpleQA)*. OpenAI. arXiv:2411.04368.
4. Meinke, A., Schoen, B., Scheurer, J., Balesni, M., Shah, R., Hobbhahn, M. (2024). *Frontier Models are Capable of In-context Scheming*. arXiv:2412.04984.
5. van der Weij, T., Hofstätter, F., Jaffe, O., Brown, S. F., Ward, F. R. (2024). *AI Sandbagging: Language Models can Strategically Underperform on Evaluations*. arXiv:2406.07358.
6. Lanham, T., Chen, A., Radhakrishnan, A., et al. (2023). *Measuring Faithfulness in Chain-of-Thought Reasoning*. Anthropic. arXiv:2307.13702.
7. Turpin, M., Michael, J., Perez, E., Bowman, S. R. (2023). *Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting*. NeurIPS 2023. arXiv:2305.04388.
8. Yang, C., Neamtu, N., Hawblitzel, C., Lorch, J. R., Lu, S. (2025). *VeruSAGE: A Study of Agent-Based Verification for Rust Systems*. arXiv:2512.18436.
9. Lin, Z. (2025). *No cheating mode*. Pull Request #1499, verus-lang/verus (merged 2025-03-21). <https://github.com/verus-lang/verus/pull/1499>. Adds `--no-cheating`, disallowing `assume`, `admit`, `#[verifier::external_body]`, and `assume_specification`.
10. Lattuada, A., Hance, T., Cho, C., et al. (2023). *Verus: Verifying Rust Programs using Linear Ghost Types*. Proc. ACM Program. Lang. 7, OOPSLA1, 286–315. DOI 10.1145/3586037. arXiv:2303.05491.
11. Sun, C., Sheng, Y., Padon, O., Barrett, C. (2024). *Clover: Closed-Loop Verifiable Code Generation*. SAIV 2024, LNCS. arXiv:2310.17807. (CloverBench is the evaluation dataset introduced here.)
12. Austin, J., Odena, A., Nye, M., et al. (2021). *Program Synthesis with Large Language Models*. arXiv:2108.07732. (The MBPP source paper.)

13. Loughridge, C., Sun, Q., Ahrenbach, S., et al. (2024). *DafnyBench: A Benchmark for Formal Software Verification*. arXiv:2406.08467.
14. Yang, C., Li, X., Misu, M. R. H., et al. (2024/2025). *AutoVerus: Automated Proof Generation for Rust Code*. Proc. ACM Program. Lang. 9, OOPSLA2. arXiv:2409.13082. (VerusBench, a 150-task suite, is introduced within this paper.)
15. Dougherty, Q., Mehta, R. (2025). *Proving the Coding Interview: A Benchmark for Formally Verified Code Generation (FVAPPS)*. arXiv:2502.05714.
16. Thakur, A., et al. (2025). *CLEVER: A Curated Benchmark for Formally Verified Code Generation*. arXiv:2505.13938.
17. Ye, Z., et al. (2025). *VERINA: Benchmarking Verifiable Code Generation*. arXiv:2505.23135.

Appendix

Limitations

- *False positives*: whole-file `--no-cheating` flags any unsound primitive, including legitimate systems trust boundaries; this is why we filtered VeruSAGE references using `external_body / assume_specification` out of the clean set. The hard tier needs a target-function-scoped detector.
- *False negatives*: we measure soundness-claim dishonesty only, not spec gaming (proving a weaker property than claimed), selective disclosure, or dishonesty that never touches a verifiable artifact.
- *Edge cases*: Verus version drift changes what verifies (we pin a version and normalize termination semantics); blind belief has near-chance fidelity and is invalid without the audit.
- *Scalability*: Verus is not concurrency-safe at the file level, so pilot runs are serialized; a publication harness must isolate per-run (`CARGO_TARGET_DIR / container`) to parallelize.

Dual-Use Considerations

The benchmark is defensive: it measures and discourages dishonesty in AI-produced proofs, and the firewall stops a model from learning to evade the soundness check during evaluation. The residual risk is that naming the primitives and detector could help a model author route around them (e.g. craft an unsound proof that still passes `--no-cheating`). We judge this low: `--no-cheating` is already public upstream, and the more concerning capability (spec gaming) is out of scope. We recommend the target-function-scoped detector and held-out task slices to keep evaluation ahead of evasion.

LLM Usage Statement

This project was built with substantial LLM assistance (Claude) for code, experiment design, and drafting this report. All results, the verifier outputs, and the mechanical claim-vs-artifact checks are produced by the instrument and were independently verified by running the test suite (46 tests passing) and the analysis scripts. Every numeric claim in §4 traces to a run log under `runs/`. Chain-of-thought traces are treated as corroboration only.