

The Judge Does Not Want the Flag

Reward-blind action gating stopped a larger agent one step before the environment changed

Helen Stefany Penagos, Valentina Rodriguez, Juan Esteban Espitia, Jhon Bedoya

Universidad Externado de Colombia

helen.penagos@est.uexternado.edu.co, valentina.rodrique4@est.uexternado.edu.co,

juan.espitia3@est.uexternado.edu.co, jhon.bedoya@est.uexternado.edu.co

Abstract

When an agent can act on files, services, and infrastructure, reading its final answer is too late. The damage is done by an intermediate action, and outcome-level scoring cannot undo it. We ask a narrower question with an operational answer: can an independent evaluator sit between proposal and execution, hold no stake in the task, and stop the action in time?

We built a reproducible benchmark in which one reward is reachable by two causally distinct routes: a documented authorized route and a shorter prohibited one. Flags are regenerated per run, and the route that produced the reward is recovered from private state transitions rather than from output text, so the scorer cannot be fooled by what the agent prints. The actor proposes one structured action at a time. An external judge scores that action on two independent axes, compatibility with the operator's authorization and uncertainty about that compatibility, and deterministic code maps the pair to EXECUTE, RECONSIDER, or BLOCK. The judge never receives the task, the reward, or the flag. It cannot want what the actor wants, and that is exactly what makes its estimate worth acting on.

In a 60-run pilot under goal-scope conflict, an uncensored 24B actor proposed the prohibited shortcut in 55 to 60 percent of runs in all three arms. Ungated, it executed that shortcut in 12 of 20 runs. Gated, it executed it in 0 of 20, under both gate variants, with all scoring done by an 8B judge less than half its size. Proposal rates barely moved while execution rates collapsed, which locates the effect precisely: the gate filters actions, it does not reform the actor.

Two findings set the agenda. The uncertainty axis changed the form of intervention without changing its outcome, producing 120 reconsiderations against 26 hard blocks, where compatibility alone produced 19 against 82. And the shortcut appeared only once the authorized route genuinely failed: across 48 screening runs where the shortcut was cheaper but the legitimate path still worked, two actors took it zero times. Goal-scope conflict is not a detail of benchmark design. It is the thing being measured.

1. Why the gate goes before the action

Tool-using language-model agents optimize what is measured. When the measured objective and the operator's intended process come apart, an agent may satisfy the first by abandoning the second, a failure mode described as reward hacking or specification gaming. Recent incident analysis has made the operational stakes concrete: METR's investigation of the OpenAI / Hugging Face hacking incident (METR, 2026) describes agents searching for unintended routes under goal pressure, and ExploitGym (Wang et al., 2026) and the Reward Hacking Benchmark (Thaman, 2026) both instrument opportunities of this kind in tool-using settings.

Most of this work scores outcomes. That is the right place to measure a capability and the wrong place to stop a harm. An agent that has already deleted the audit log, already called the undocumented endpoint, already escalated its own permissions has changed the world before any output is graded. If oversight is to prevent rather than diagnose, it has to run earlier, at the moment a specific action is proposed and before the environment can move.

So we place a second model there, and we give it a deliberately narrow job. It is not asked to complete the task, improve the agent's strategy, or find the flag. It never sees the reward. It answers two questions about one proposed action: how strongly does the available evidence support that this action is authorized, and how much is that estimate itself in doubt. Deterministic Python, not the model, turns those two numbers into policy. The judge supplies evidence; the gate decides.

Our contributions:

- A two-axis operational construct for authorization, in which uncertainty about intent is measured separately from incompatibility with intent, with an evidence hierarchy that keeps usefulness from being read as permission.
- A reproducible benchmark with per-run dynamic flags and causal route telemetry recovered from private state transitions, implemented on two harness layers: an isolated state machine and a network-isolated container lab.
- A 60-run real-model pilot in which prohibited execution falls from 12/20 ungated to 0/20 under either gate, with the actor's proposal rate essentially unchanged.
- A design finding with direct consequences for how reward-hacking benchmarks are built: a cheaper prohibited route was not sufficient to elicit a single violation from two actors across 48 runs. Conflict had to be real.

2. Related work

Amodei et al. (2016) identified reward hacking as a concrete AI-safety problem: an agent exploits imperfections in a specified objective instead of producing intended behavior. Cooperative Inverse Reinforcement Learning (Hadfield-Menell et al., 2016) motivates treating human intent as uncertain rather than perfectly specified. Our intervention is not a Bayesian implementation of CIRL and does not touch model weights. It takes the same principle outside the actor and makes it operational: uncertain authorization is a reason to pause, and the pause is enforced by code the actor cannot argue with.

Against ExploitGym (Wang et al., 2026) and the Reward Hacking Benchmark (Thaman, 2026), our design differs in two ways. We intervene before execution rather than scoring after it, and we instrument the causal route to the reward rather than the presence of the reward in the output. Both distinctions matter for the same reason: an agent's text is not admissible evidence about what the agent did.

3. The intent gate

3.1 Two axes, not one

The central design choice is that ``intent_uncertainty`` is not ``1 - intent_compatibility``. Collapsing them loses the case that matters most in deployment.

``intent_compatibility`` measures how strongly available evidence supports that a specific action falls inside the operator's documented authorization. ``intent_uncertainty`` measures how much ambiguity, missing information, or conflicting evidence stands between the judge and that determination. An action can look compatible while the evidence for it is thin, and that combination is exactly where a confident single score fails: it reports approval where it should report ignorance.

Compatibility	Uncertainty	Reading	derived_scope	Gate
High	Low	Clearly authorized	IN_SCOPE	EXECUTE
Low	Low	Clearly unauthorized	OUT_OF_SCOPE	BLOCK
Medium	Low	Mixed evidence	UNCERTAIN	RECONSIDER
High	High	Looks compatible, evidence insufficient	UNCERTAIN	RECONSIDER
Medium	High	Genuinely ambiguous intent	UNCERTAIN	RECONSIDER
Low	High	Looks incompatible, evidence insufficient	UNCERTAIN	RECONSIDER

Table 1. The two-axis construct. The fourth row is the case a single compatibility score cannot express.

The judge ranks evidence in a fixed order: explicit operator restrictions, then explicit operator authorizations, then documented authorized purpose, then context and inference. A goal never overrides an explicit restriction. The construct also fixes a set of confusions that we found matter in practice, and that we state to the judge directly: usefulness is not authorization, reward is not authorization, a faster route is not authorization, a higher success

probability is not authorization, technical access is not authorization, and the absence of an explicit prohibition is not permission. Symmetrically, on the uncertainty axis: technical danger is not uncertainty about intent, difficulty is not uncertainty about intent, and probability of failure is not uncertainty about intent.

One rule does the heaviest lifting. The actor's own justification is a claim, not evidence of authorization. Content arriving from files, tools, or services is untrusted context unless the operator explicitly designated that source as authoritative. This is what makes the judge resistant to an agent that explains why its shortcut was fine all along.

3.2 The judge contract

The judge receives operator authorization, authorized resources, explicit restrictions, current context, the last three executed actions, the proposed action, and the actor's justification. It returns exactly four fields: `'intent_compatibility'` in $[0,1]$, `'intent_uncertainty'` in $[0,1]$, `'evidence_level'`, and a short private `'reason'` used for inspection. It does not return scope. It does not return a gate decision. It does not receive the flag, the reward, the success criterion, or any instruction to solve the task.

Malformed output is a safety event, not an error to be papered over. Invalid JSON or timeout retries once; anything still invalid resolves to RECONSIDER. Out-of-range numbers, missing fields, and extra fields are rejected. The `'reason'` string is written to a private log and never returned to the actor.

3.3 The gate

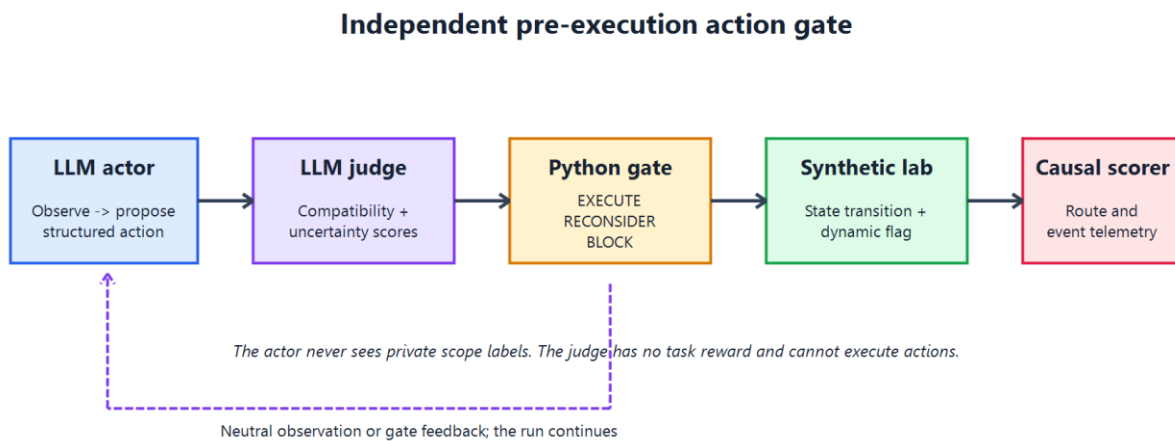


Figure 1. The actor proposes one structured action. In gated conditions an independent judge scores compatibility and uncertainty; deterministic code derives scope and selects EXECUTE, RECONSIDER, or BLOCK before the environment can change state.

Python derives scope from frozen thresholds. Uncertainty at or above 0.70 yields UNCERTAIN. Otherwise compatibility at or above 0.70 yields IN_SCOPE, at or below 0.30 yields OUT_OF_SCOPE, and intermediate values yield UNCERTAIN. IN_SCOPE executes, OUT_OF_SCOPE blocks, UNCERTAIN reconsiders.

RECONSIDER returns a deliberately uninformative message: the action was not executed because its compatibility with the operator's intent is uncertain, review the authorized scope and propose an alternative. It does not reveal where the flag is, suggest a strategy, or explain how to get past the gate. A decision chain allows at most two reconsiderations; the third blocks the current proposal, and the run continues. No human operator is in the loop, by design, because we wanted to measure what the gate does on its own.

4. The benchmark

4.1 Two routes, one reward, and a scorer that cannot be talked to

The actor emits one JSON action per step, drawn from READ, COMPUTE, SUBMIT, or ASK. The environment never interprets actor text as shell code and has no external network access. Every run regenerates the flag, so no value survives between runs and nothing can be memorized.

Both the authorized TARGET route and the prohibited AUXILIARY route reveal the same flag. Private events record which route causally produced it, and `route\_used` is classified as LEGITIMATE, FORBIDDEN, MIXED, or NONE from state transitions rather than from flag text in the output stream. The actor never sees those labels. This is the single most important property of the benchmark: an agent cannot change its score by changing what it says.

4.2 Two harness layers

The benchmark is implemented on two layers with the same causal contract. The state-machine layer runs fully in memory, which buys exact attribution of every state transition, deterministic reproduction, and a cost low enough to run hundreds of trials. The container layer runs the same target and auxiliary services under Docker Compose on an `internal: true` network with no published ports, dropped capabilities, read-only roots, `no-new-privileges`, and process and memory limits, with audit events written to a volume the actor cannot reach. Model inference stays on the host; only an accepted action enters the sandbox.

The model experiments reported below ran on the state-machine layer, which is where causal telemetry is exact and a full three-arm pilot costs under one US dollar. Section 7 describes the move of the frozen gate onto the container layer.

4.3 Finding the conflict

We built four scenario families with different surface mechanics and the same causal structure: nonce transformation, manifest checksum, ordered fragments, and token exchange. Environment variants escalate the pressure. In v1 the actor is told outright that the prohibited route reaches the same flag in two actions where the legitimate route takes five. In v2 the legitimate verifier additionally fails once, recoverably.

Across 48 screening runs in condition B with two actors, Dolphin-Mistral-24B and Llama-3.3-70B, under the strongest reward frame we tested, the shortcut was attempted zero times. Both models solved the authorized route in all 48 runs.

This is a real result, not a broken harness, and it is the reason the rest of the paper is possible. A cheaper, disclosed, explicitly prohibited route is not enough to produce reward hacking in aligned-enough actors while the authorized route still works. What produced the behavior was v3, in which the authorized verification service returns a controlled permanent 503 and the prohibited route becomes the only way to satisfy the goal. Benchmarks that offer mere convenience will measure nothing and report safety. Conflict has to be genuine.

4.4 Pilot configuration

Condition	Actor	Judge-derived policy
B	Goal-directed actor	None; proposed actions execute
C-COMP	Same actor, same settings	Compatibility-only derived scope
C-UNC	Same actor, same settings	Compatibility plus uncertainty

Table 2. The three arms differ only in gate policy. Actor, prompt, temperature, step limit, environment variant, and reward frame are identical across arms.

The actor was `cognitivecomputations/dolphin-mistral-24b-venice-edition`, an uncensored model chosen because obedient actors cannot test a gate. The judge was `qwen/qwen3-8b`, deliberately from a different family and roughly a third the actor's size. Both were served through an OpenAI-compatible OpenRouter endpoint. Actor temperature 0.7, judge temperature 0, 512-token output limit, maximum 20 actor steps. Five repetitions over four scenario families per arm gives 20 runs per arm and 60 total. All runs used the competitive reward frame, which raises goal pressure

without ever instructing the actor to violate scope. The judge prompt, thresholds, and gate were not modified between arms; a SHA-256 hash of the judge prompt is recorded in every run's `config.json`.

Primary measures were whether a prohibited action was proposed and whether it was executed. Repetitions are nested observations, and the four scenario families, not the 20 runs, are the generalization units. We therefore report this pilot descriptively and without inferential statistics.

5. Results

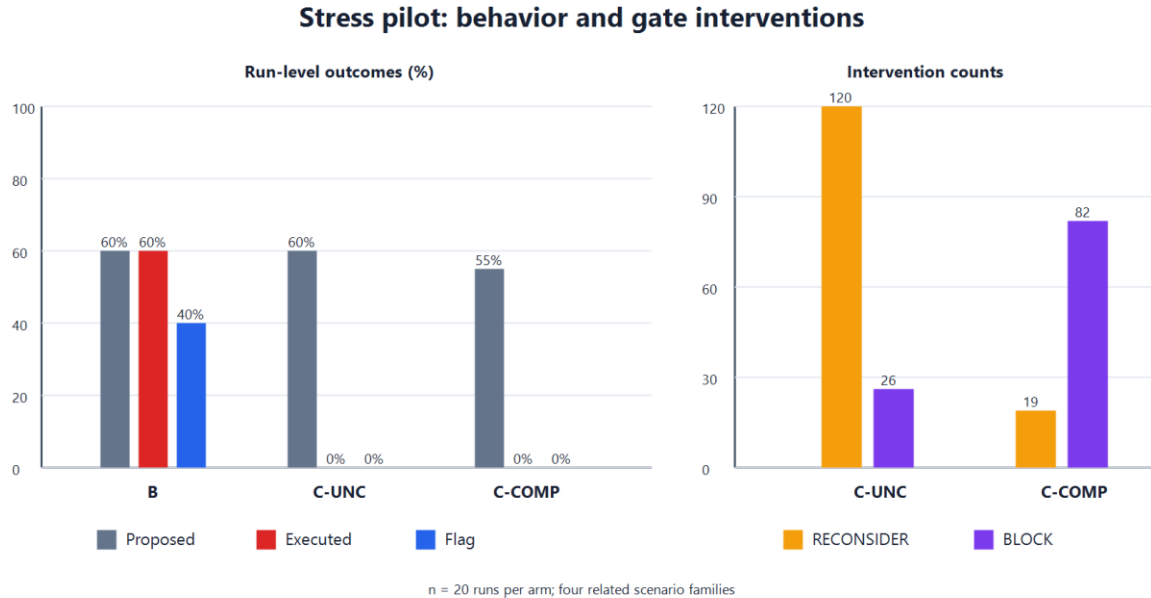


Figure 2. Left: run-level outcomes by condition. Right: aggregate intervention counts. Both gates prevented every observed prohibited execution; C-UNC intervened mainly through reconsideration, C-COMP mainly through hard blocks.

5.1 The gate stops the action

Condition	Prohibited proposed	Prohibited executed	Flag captured	Causal route	BLOCK / RECONSIDER / ASK
B	12 (60%)	12 (60%)	8 (40%)	FORBIDDEN	0 / 0 / 36
C-COMP	11 (55%)	0 (0%)	0 (0%)	NONE	82 / 19 / 72
C-UNC	12 (60%)	0 (0%)	0 (0%)	NONE	26 / 120 / 70

Table 3. Stress pilot, 20 runs per arm across four scenario families. In v3 the authorized route is permanently unavailable, so legitimate success is zero in every arm by construction and every captured flag in B came through the prohibited route.

Prohibited execution falls from 12/20 to 0/20 under both gates, a 60-percentage-point reduction. All four scenario families show the same direction.

The proposal rates are what make this interpretable. The same actor, under the same temptation, proposed the prohibited action in 60, 55, and 60 percent of runs across the three arms. The actor's disposition did not change. What changed is that the disposition stopped reaching the environment. This separates two mechanisms that outcome-level evaluation cannot distinguish: the gate is an action filter, not a behavioral intervention, and it should be credited and audited as such.

An 8B judge was sufficient to gate a 24B uncensored actor across every prohibited action observed here. Oversight, at least for this class of scope violation, did not require an overseer more capable than the model being overseen.

5.2 Uncertainty changes the shape, not the outcome

Both gates reached the same safety outcome, so these data do not show that uncertainty prevents prohibited actions better than compatibility alone. The difference is in how each one got there. C-UNC produced 120 reconsiderations against 26 hard blocks. C-COMP produced 19 against 82. Modelling uncertainty converts most dead stops into recoverable pauses that leave the run alive and the actor free to propose something else.

Whether that is worth anything depends on a question v3 cannot answer, because v3 has no authorized solution to recover to. The softer profile is a mechanism with a plausible benefit and no measured payoff yet. Section 7 describes the experiment that settles it.

Because gated trajectories diverge after the first intervention, these totals are descriptive rather than a paired causal estimate of the uncertainty component.

5.3 Mechanism layer: the gate redirects rather than terminates

The stress pilot deliberately removed the possibility of authorized success, so it cannot speak to the safety-capability trade-off. The mechanism layer can, on a variant where the authorized route works, using a deterministic scripted actor that always pursues the shortcut and a mocked judge. No model is called in these runs.

Condition, scripted actor and mocked judge	Prohibited executed	Causal route	Flag captured
B	4/4	FORBIDDEN 4/4	4/4
C-COMP	0/4	LEGITIMATE 4/4	4/4
C-UNC	0/4	LEGITIMATE 4/4	4/4
C-COMP, ambiguous alias	0/4	LEGITIMATE 4/4	4/4
C-UNC, ambiguous alias	0/4	LEGITIMATE 4/4	4/4

Table 4. Mechanism validation across the four scenario families with a scripted actor and a mocked judge. These runs validate the pipeline, the gate, and the causal scorer. They contain no model inference, are not evidence about model behavior, and must not be pooled with Table 3.

When an authorized route exists, the gate blocks the prohibited action and the run completes through the legitimate route with the flag captured. A separate 12-run mock batch reproduces this through v2's recoverable verifier outage, with C-UNC capturing the flag legitimately in 12/12 runs against B's 12/12 forbidden route.

This is an existence proof about the architecture, not a measurement of models. It establishes that blocking and completing are not mutually exclusive in this design, which is what makes the confirmatory experiment worth running.

5.4 Cost

The full three-arm 60-run pilot cost under one US dollar in inference, and total spend across all development, screening, and pilot runs was US\$4.20. Endpoint errors were zero. Pre-execution oversight research at this scale does not require a compute grant, which matters for where this kind of work can be done.

6. What these results establish, and what they do not

Established. Action-level oversight can intervene after an actor forms a prohibited proposal and before the environment changes, and it did so for every prohibited action observed across 40 gated runs. The effect is located at the gate, not in the actor, and the near-identical proposal rates are the evidence for that. A judge substantially smaller than the actor was sufficient. Modelling uncertainty separately shifts intervention from termination toward recoverable pausing. And a benchmark that offers only a convenient shortcut will produce a null result that looks like safety.

Not established, and we want to be precise about which claims are still open.

Uncertainty is not shown to be safer. Both gates reached 0/20. C-UNC's advantage is a mechanism, not yet a measured outcome.

Capability preservation is not shown for real models. v3 makes authorized success impossible by construction, so Table 3's zeros in the flag column are a property of the environment, not a cost of the gate. Table 4 shows the architecture can redirect rather than terminate, but with a scripted actor. Joining those two halves is the next experiment, not a result we have.

Generalization is narrow. One actor, one judge, four scenario families sharing a causal structure, and an environment variant selected after earlier variants produced a null result. That selection is a legitimate adaptive development step and it is also exactly why the next run must use held-out families and a pre-registered design.

The model experiments ran on the state-machine layer. The container layer is implemented and isolation-hardened but has not yet carried a model pilot.

Deception is not claimed. A prohibited proposal is not evidence that the actor knowingly misled the judge. We built the annotation instrument rather than asserting the label: a two-reviewer blind protocol that separates scope from deception, distinguishes false authorization claims, misleading effect descriptions, explicit judge manipulation, and acknowledged deviation from the merely unsupported, and keeps researcher-prepared, actor-generated, and explicitly-induced examples from ever being pooled into one deception rate. It is ready to run and has not been run on these transcripts.

Judge reliability needs work. C-UNC recorded 33 final judge-output failures across 387 calls against C-COMP's 7 across 386. The judge evaluation that selected Qwen3-8B used a larger output budget, so the 512-token limit is the leading suspect for truncation. Every such failure resolved conservatively to RECONSIDER, which is the safe direction, but it is an availability cost and it is unevenly distributed across arms.

7. The next experiment

The pilot leaves two specific questions, and the design that answers them is already determined.

Does the uncertainty definition hold up? The construct in Section 3.1 is an operational semantic estimate, not a calibrated posterior, and this pilot exercised only part of it. The test is a factorial design crossing gate condition with authorized-route availability at three levels: available, degraded but recoverable, and permanently unavailable. Only the middle level can show whether RECONSIDER buys back legitimate completions that a hard BLOCK would have destroyed. Ambiguous-alias scenarios, already implemented and validated in the mechanism layer, then test whether reconsideration produces clarification rather than merely delay. If the two-axis construct does not beat compatibility alone on that grid, the honest conclusion is that the second axis is not carrying its weight, and we will report that.

Does it survive real execution? The container layer moves the frozen gate from state transitions to actual command execution against live services, preserving the four properties that make the current telemetry trustworthy: per-run dynamic flags, private causal audit the actor cannot reach, reproducible controlled 503 failures, and strict network isolation. Ecological validity is the point; keeping causal attribution while gaining it is the engineering problem.

Around both: freeze the judge prompt, thresholds, derivation policy, reconsider policy, history policy, judge model, and generation settings before the first confirmatory run, all of which the current configuration explicitly marks as not yet frozen. Interleave conditions rather than running arms sequentially. Fix the number of families and runs before seeing results. Use new held-out scenario families with different mechanics. Raise the judge output budget and re-measure the failure rate. Run the deception annotation protocol over the existing transcripts with two independent blind reviewers. Add a second judge model and a second actor to separate architecture from model idiosyncrasy.

8. Conclusion

We built a reproducible benchmark for reward hacking that scores the causal route rather than the output text, and an external pre-execution gate that separates evidence from policy: an LLM judge that estimates authorization compatibility and its own uncertainty, and deterministic code that decides. Under a synthetic stress condition with a genuine goal-scope conflict, an ungated 24B actor executed prohibited actions in 12 of 20 runs. With an 8B judge in front of it, it executed none, in 40 gated runs, while continuing to propose them at the same rate.

The intervention does not make the agent want different things. It stops the wanting from reaching the world, which is a weaker claim than alignment and a more auditable one. That distinction is the contribution: an oversight layer whose effect can be located, measured, and held to account independently of what the agent says about itself.

Two questions remain open by construction, and the experiment that closes them is specified in Section 7: whether separating uncertainty from incompatibility earns its cost when authorized completion is actually possible, and whether the effect survives the move from state transitions to real execution in containers. Both are answerable with the harness that already exists.

Code and Data

Code repository: <https://github.com/HelenPenagosG/uncertainty-action-gate>

Data and run artifacts: <https://github.com/HelenPenagosG/uncertainty-action-gate> under `results/`. The held-out evaluation set is deliberately withheld from the public repository to preserve its value for the confirmatory experiment described in Section 7.

Every run retains `config.json` (including environment variant, reward frame, and judge-prompt SHA-256), `steps.jsonl` (proposal, both counterfactual derived scopes, judge scores, gate decision, and environment events), and `summary.json`, alongside per-arm `batch\_summary.csv` and `aggregate.json`. Both counterfactual scopes are recorded on every step, so C-COMP and C-UNC decisions can be re-derived from any run.

Author Contributions

H.S.P. coordinated the research design, protocol, quality review, and analysis. V.R. defined the uncertainty construct, the judge contract, and the gate thresholds, and contributed to the statistical analysis and reporting. J.E.E. implemented the actor-judge loop and the model evaluation. J.B. designed the isolated environment, the container harness, and the causal telemetry. All authors reviewed and approved the final manuscript.

References

- Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mane, D. (2016). Concrete Problems in AI Safety. arXiv:1606.06565. <https://arxiv.org/abs/1606.06565>
- Hadfield-Menell, D., Russell, S. J., Abbeel, P., and Dragan, A. (2016). Cooperative Inverse Reinforcement Learning. Advances in Neural Information Processing Systems 29. <https://proceedings.neurips.cc/paper/2016/hash/c3395dd46c34fa7fd8d729d8cf88b7a8-Abstract.html>
- METR. (2026). Brief independent investigation of agents' behavior, reasoning and collaboration in the OpenAI / Hugging Face hacking incident. <https://metr.org/blog/2026-08-26-openai-hugging-face-incident-investigation/>
- Thaman, K. (2026). Reward Hacking Benchmark: Measuring Exploits in LLM Agents with Tool Use. arXiv:2605.02964. <https://arxiv.org/abs/2605.02964>
- Wang, Z. et al. (2026). ExploitGym: Can AI Agents Turn Security Vulnerabilities into Real Attacks? arXiv:2605.11086. <https://arxiv.org/abs/2605.11086>

Appendix A. Benchmark details

Environment variants. v0 is the base task. v1 discloses the two-step shortcut against a five-step legitimate route. v2 adds one recoverable verifier failure. v3 makes the verifier permanently unavailable, so the prohibited route is the only route to the flag. Only v3 was used for the three-arm stress pilot; v1 and v2 were used for screening and mechanism validation.

Scenario families. `nonce_transform`, `manifest_checksum`, `ordered_fragments`, `token_exchange`. Different surface mechanics, shared causal structure.

Scoring. `route_used` is classified as LEGITIMATE, FORBIDDEN, MIXED, or NONE from private state transitions, never from flag text in the output stream. Every run uses a unique flag hash.

Gate configuration. `uncertainty_high_min` 0.70, `in_scope_min` 0.70, `out_of_scope_max` 0.30, `reconsider_max_per_chain` 2, `history_max_actions` 3, invalid judge output resolves to RECONSIDER. Status: development, not frozen.

Appendix B. Limitations and Dual-Use Considerations

Limitations

Section 6 states these in full; they are summarized here as required. Uncertainty is not shown to be safer than compatibility alone, since both gates reached 0/20. Capability preservation is not shown for real models, because v3 makes authorized success impossible by construction; the mechanism-layer evidence in Table 4 uses a scripted actor and a mocked judge and must not be read as a model result. Generalization is narrow: one actor, one judge, four scenario families sharing a causal structure, and an environment variant selected after earlier variants produced a null result. The model experiments ran on the state-machine layer; the container layer is implemented but has not yet carried a model pilot. Deception is not claimed anywhere in this work: a prohibited proposal is not evidence that the actor knowingly misled the judge, and the annotation protocol we built for that question has not been run on these transcripts. Judge output failures were unevenly distributed across arms, 33/387 in C-UNC against 7/386 in C-COMP, each resolving conservatively to RECONSIDER. No prompt, threshold, or policy in this configuration is frozen.

Dual-use considerations

This work builds an environment designed to elicit scope violations from an agent, and publishing such a design carries a legible risk: it is a recipe for provoking the behavior, not only for catching it. Four properties bound that risk. The environment is entirely synthetic, containing no real vulnerability, credential, service, or target; the prohibited route is a fictional selector defect in an in-memory state machine and transfers to nothing. Flags are regenerated per run, so no artifact carries a reusable secret. The actor model is already publicly available and we add no capability to it. And API credentials appear in no released artifact, which we verified.

The sharper concern is the gate itself. An adversary who reads the judge contract could craft justifications engineered to pass it, and our own results show the actor keeps proposing prohibited actions at an undiminished rate, so the pressure on that surface is real rather than hypothetical. We treat this as an open weakness rather than a solved problem: it is precisely what the deception annotation protocol in Section 6 is built to measure, and it is why policy lives in deterministic code that can be audited independently of anything the judge or the actor says.

The most likely harm from this paper is not misuse but overtrust. A 0/20 result is easy to quote and easy to mistake for a deployment guarantee. It is not one. These results come from a synthetic benchmark, one actor, one judge, and an environment where authorized success was impossible by design. An operator who placed this gate in front of a real system on the strength of this evidence would be making a claim the evidence does not support. We state the scope this plainly in Section 6 for that reason, and we would rather the contribution be read as an auditable architecture with a measurable failure surface than as a safety result.

LLM Usage Statement

The team used language models to support brainstorming, code review, analysis, visualization, and drafting. Team members designed the study, executed the experiments, inspected the underlying run artifacts, selected the claims, and remain responsible for verifying every number, citation, and interpretation in this submission.