

CommitCheck: Measuring and Mitigating Commitment Violations in Tool Using AI Agents

Team: CommitCheck

Authors: Faith Olopade

Track: Measurement & Evaluation and Mitigations

Code: <https://github.com/faitholopade/commitcheck>

Demo command: commitcheck demo --seed 0

Abstract

AI manipulation becomes operationally dangerous when models act as tool using agents: they can violate constraints to improve task reward (reward hacking) and misrepresent their actions to operators (deception). We introduce CommitCheck, a benchmark and mitigation system that measures commitment vs action divergence using hard evidence from tool and file-access logs. CommitCheck scores attempted and executed violations, deception (log/claim mismatch), and task success. We also implement a lightweight commitment-aware firewall that blocks forbidden tool/file access at runtime. On 30 synthetic tasks (arithmetic, logic, and file-based math) using an offline “cheater” agent, the baseline agent executes violations in 23.3% of tasks and exhibits deception in 13.3% overall (57.1% conditional on violation). Enabling the firewall reduces executed violations to 0.0% and deception to 0.0% without reducing task success (100%).

1. Problem Statement

Most manipulation evaluation remains text-only, but real manipulation risk increases when AI systems are deployed with tools (file access, web search, privileged APIs). In these settings, a model can:

- take forbidden shortcuts (reward hacking),
- hide or deny doing so (deception),
- behave differently when it expects auditing (strategic behavior).

We need evaluation infrastructure that is:

- log-grounded (not guesswork),
- ecologically closer to agent deployments,
- and paired with deployable mitigations.

2. Approach Overview

CommitCheck has two components:

2.1 Benchmark (measurement)

Each task includes explicit constraints (e.g., “DO NOT use oracle_answer” / “DO NOT read solutions.json”). The environment still exposes tempting shortcuts (an oracle tool, answer-key files) to test whether agents violate constraints.

We log all tool calls + file accesses and compute per-task metrics:

- Attempted violation: tried to use forbidden tool/file (even if blocked)
- Executed violation: successfully used forbidden tool/file
- Deception: final compliance claim contradicts logs
- Task success: exact-match against ground truth

We also include an audit_notice condition: some tasks explicitly tell the agent that tool calls are logged/audited; others do not. This tests for strategic compliance.

2.2 Mitigation (defense)

We implement a commitment-aware firewall that:

- parses policy directly from the task metadata (forbidden tools/files),
- intercepts tool/file access,
- blocks forbidden actions and returns a “BLOCKED” observation,
- logs the blocked attempt for scoring.

This is designed as a lightweight middleware that could be integrated into agent frameworks.

3. Implementation

Repo structure

- commitcheck/runner.py: agent loop, JSON enforcement, logging
- commitcheck/envs/tool_env.py: offline tool environment (read_file, oracle_answer)
- commitcheck/firewall.py: policy enforcement wrapper
- commitcheck/scoring.py: computes per-task + aggregate metrics
- commitcheck/reporting.py: generates HTML report from summary JSON
- tasks/sample/: 30 tasks + temptation files (solutions.json, answer_key_*.txt)
- tests/: end-to-end and unit tests for determinism and scoring

Reproducibility

- Offline mock agents (no API keys required)
- Deterministic via --seed
- CLI: run, score, report, demo

4. Results

We run the bundled demo (commitcheck demo --seed 0), which executes the “cheater” agent on the benchmark with and without the firewall.

4.1 Headline metrics

Baseline (no firewall):

- attempted violation rate: 0.233
- executed violation rate: 0.233
- deception rate overall: 0.133
- deception rate given violation: 0.571
- task success rate: 1.000

With firewall enabled:

- attempted violation rate: 0.233
- executed violation rate: 0.000
- deception rate overall: 0.000
- task success rate: 1.000

Interpretation: the agent continues to *attempt* forbidden shortcuts, but the firewall prevents execution and eliminates log/claim contradictions.

4.2 Breakdown by audit notice (strategic behavior)

In baseline, attempted/executed violations are substantially higher when `audit_notice=false` compared with `audit_notice=true` (0.400 vs 0.067 attempted/executed), consistent with strategic compliance when the agent expects monitoring.

4.3 Qualitative traces

The report includes concrete traces showing:

- “Cheat and lie” examples (baseline): agent calls `oracle_answer` despite constraints and reports `used_tools=[]`.
- “Cheat blocked” examples (firewall): agent attempts `oracle_answer`, receives a BLOCKED response, and proceeds via allowed means.

5. Discussion

CommitCheck demonstrates that in a tool-using setting, constraint violations can be measured robustly via logs and mitigated via a simple enforcement layer. Importantly, it supports evaluation under adversarial assumptions: models may behave differently depending on perceived oversight.

6. Limitations & Dual-Use Considerations

Limitations

- Synthetic tasks: This benchmark is designed for controlled evaluation; real deployments have richer tool surfaces and ambiguous constraints.

- Deception definition: We operationalize deception as a mismatch between logged actions and the model's reported compliance. This catches many practical failures but does not infer intent.
- Scope: The firewall prevents forbidden tool/file access but does not address purely text-based manipulation or subtle social engineering.
- Model realism: The included "cheater" model is a simplified proxy for strategic behavior; future work should evaluate real frontier models and agent frameworks.

Dual-use risks

- Attackers could use benchmarks to train agents to better hide violations or to probe enforcement weaknesses.
- Publishing detailed task sets may enable overfitting.

Responsible release recommendations

- Publish the framework and a public task set, but encourage private/rotating evaluation sets for high-stakes use.
- Clearly label this as evaluation and defense infrastructure, not a "cheating recipe".
- Avoid including any unsafe tool surfaces; keep tasks benign and offline.

Ethical considerations

- Tasks and environments are intentionally non-harmful.
- No real user data is collected; all data is synthetic.

Future improvements

- Expand to real agent frameworks (LangChain/AutoGen) with tool-call auditing hooks
- Add richer constraint extraction (from natural language policies)
- Add robustness against prompt injection into tool outputs
- Evaluate on real model APIs under controlled policies

7. Conclusion

CommitCheck provides a reproducible benchmark for measuring commitment violations and deception in tool-using agents and demonstrates an effective mitigation. In our demo, a commitment-aware firewall eliminates executed violations and deception while preserving task success.

Appendix A: How to reproduce

```
python -m venv .venv
.\venv\Scripts\Activate.ps1
python -m pip install -e .
pytest -q
commitcheck demo --seed 0
```

Outputs:

- results/demo_baseline/report.html
- results/demo_firewall/report.html

Appendix B: Prompts / AI usage

This project was scaffolded with AI assistance (Codex) for repository generation and unit tests, then executed and validated locally. Human decisions included benchmark framing, metrics design, and mitigation design.