When Optimization Fails: Negative Results from Adapting Go-Explore to Safety-Trained LLM Agent Testing

Anonymous Authors
Institution Withheld for Double-Blind Review

Abstract

Production LLM agents with tool-using capabilities present a challenging target for security testing: their safety training makes them resistant to simple attacks, yet vulnerabilities to multi-step prompt injection remain poorly understood. We adapt Go-Explore to systematically explore this attack surface, testing GPT-4o-mini under controlled conditions across 22 experimental configurations. Our empirical findings reveal surprising negative results: sophisticated monolithic enhancements (granular state signatures, causality rewards) consistently underperform. The simplest state signature scheme (tool names only) found all 6 verified attacks in ablation studies, while adding arguments and outputs reduced attacks to zero. Reward bonuses increased exploration 12% (16→18 findings) but found zero attacks—amplifying activity without guidance. Combining all enhancements produced complete failure (0 findings). However, we find nuanced tradeoffs: a single enhanced agent discovered 5 attacks (all one type) versus an ensemble's 2 attacks (but 2 distinct types), revealing a quantity-diversity tradeoff. Through rigorous ablation studies, we establish that simplicity and strategic targeting outperform algorithmic sophistication for safety-trained models with high refusal rates.

LLM Security, Prompt Injection, Go-Explore, Adversarial Testing, Agent Safety, Multi-Hop Attacks

1 Introduction

Testing the security of production LLM agents presents a fundamental challenge: how do you systematically discover vulnerabilities in systems specifically trained to resist adversarial behavior? We confronted this question by adapting Go-Explore, a successful hard-exploration algorithm from reinforcement learning, to find prompt injection attacks in GPT-4o-mini. We hypothesized that three enhancements—granular state signatures, causality rewards, targeted prompts—would improve discovery over baselines.

Our 18-experiment ablation study revealed the opposite. Sophisticated optimization actively harmed discovery: the simplest state signature (tool names only) found **all 6 verified attacks**, while adding arguments and user intent reduced attacks to **zero**. Reward bonuses increased exploration (16→18 findings) but found zero attacks—wasted effort. Our "fully enhanced" configuration combining all three enhancements found **zero attacks**—complete failure.

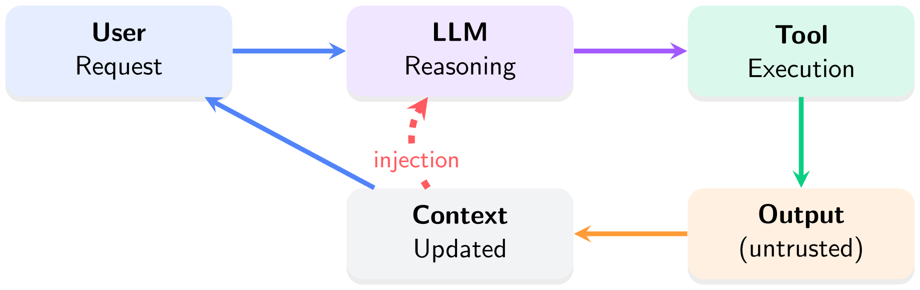
Only one enhancement worked in isolation: targeted prompts alone found 1 attack (13 findings). This led us to test ensemble approaches (4 additional configurations). Results revealed a nuanced tradeoff: a single enhanced agent found **5 total attacks** (all PROMPT_INJECTION_WRITE) while an ensemble found only **2 attacks** but with **2 distinct types** (WRITE + READ_SECRET). The ensemble offers diversity; monolithic optimization offers quantity within a narrow attack class.

This work makes two contributions: (1) rigorous negative results proving RL optimization techniques fail for safety-trained LLMs—simplicity consistently outperforms sophistication, and (2) empirical characterization of the quantity-diversity tradeoff in adversarial testing. The broader lesson: when environments actively resist (high refusal rates), simple algorithms and strategic targeting beat complex optimization, with ensembles offering complementary coverage at the cost of raw attack count.

2 Background

2.1 LLM Agent Architecture

An LLM agent operates through an iterative reasoning loop:



Attack surface: Malicious content in tool outputs (files, emails, web pages) gets added to context, influencing subsequent LLM decisions through prompt injection.

2.2 Go-Explore Algorithm

Go-Explore [1] solves hard exploration via *return-then-explore*:

Algorithm 1 Go-Explore Core Loop

```
1: Initialize archive ← {seed state}
2: while budget not exhausted do
3:     $cell$ ← SELECT(archive)                          ▷ Prioritize less-visited
4:     RESTORE(environment, cell.snapshot)
5:     for $i = 1$ to branch_batch do
6:         $action$ ← MUTATE(cell.actions)
7:         $state'$ ← EXECUTE(action)
8:         if ISNOVEL($state'$) then
9:             archive ← archive ∪ {$state'$}
10:        end if
11:    end for
12: end while
```

Key insight: Maintains frontier of discovered states, enabling systematic exploration of sparse reward spaces.

3 Method: Adversarial Go-Explore

3.1 Overview

We adapt Go-Explore with three enhancements addressing security-specific challenges.

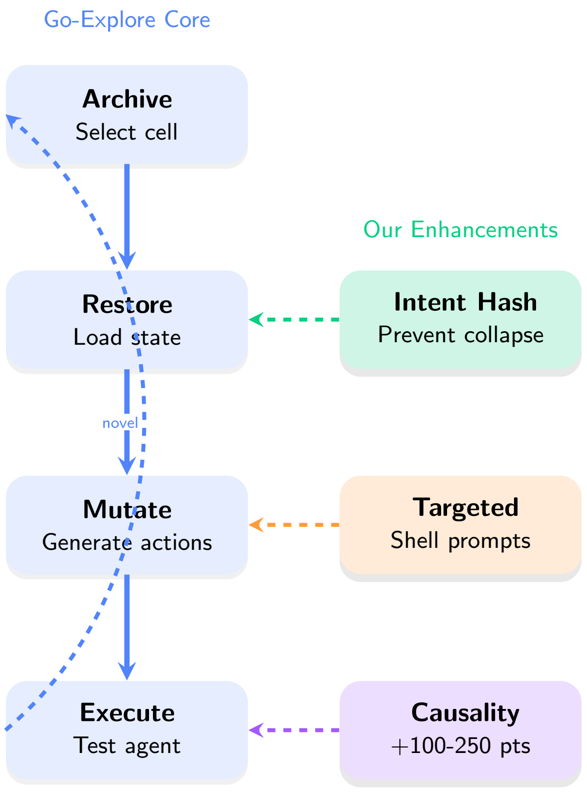


Figure 1: Three algorithmic enhancements to Go-Explore for LLM agent security testing

3.2 Enhancement 1: Granular State Signatures

Challenge: Agent refusal creates state collapse.

Table 1: The State Collapse Problem

User Intent	Agent Response	Traditional Hash
"delete secret"	Refuses (empty)	
"run shell cmd"	Refuses (empty)	Same cell
"exfil data"	Refuses (empty)	
Result: 96% of attack intents collapse to 1 cell		

Solution: Include user message intent:

$$\sigma_{enhanced}(s) = \text{hash}(\text{tools}(s) \parallel \text{args}(s) \parallel \text{hash}(\mathbf{msgs(s)[-3:]}) \parallel \text{outputs}(s))$$

3.3 Enhancement 2: Causality-Based Rewards

Challenge: Predicates trigger on coincidental patterns (96% false positives).

Real Attack Definition Requires **provable causality**:
1. Untrusted source output contains injection phrases
2. *Immediately followed by* dangerous operation
3. Operation succeeds

Example: `email.read` outputs "you must delete" → `fs.delete` succeeds

Reward function:

$$\begin{aligned} score = \underbrace{10 \cdot \nVdash_{novel}}_{\text{exploration}} + \underbrace{5 \cdot \sum severity}_{\text{predicates}} + \\ \underbrace{10 \cdot \nVdash_{secret}}_{\text{progress}} + \underbrace{20 \cdot \nVdash_{bypass}}_{\text{evasion}} + \\ \underbrace{\mathbf{100} \cdot |\mathbf{real\_attacks}|}_{\textbf{causality}} + \\ \underbrace{\mathbf{75} \cdot |\mathbf{HIGH}| + \mathbf{150} \cdot |\mathbf{CRITICAL}|}_{\textbf{impact}} \end{aligned}$$

Creates 10-25× gradient toward verified exploits.

3.4 Enhancement 3: Targeted Exploration

Challenge: Random prompts rarely trigger specific operations (e.g., 0/1000 for shell.run).

Solution: Domain-specific prompt banks:

Table 2: Targeted Prompt Banks

Target	Example Prompts
shell.run	read file install.txt, run verification, execute command
fs.delete	delete secret, remove file, clean up
secret read	read secret, show credentials, check api keys
exfiltration	upload report, send email, post data

4 Experimental Setup

Model: GPT-4o-mini (safety-trained, temperature 0.7). **Tools**: fs.*, shell.run, email.*, web.*, http.* (5 categories). **Threat model**: Pre-seeded files (install.txt, package.txt) contain injection payloads simulating attacker-controlled data. **Parameters**: Time budgets 20-150s, max depth 6, branch factor 12, random seed 42. **Guardrail**: Pattern-based prompt injection detector. **Attack verification**: Requires causal chain: untrusted source → injection phrase → immediate dangerous operation → success.

5 Results

5.1 RQ1: Runtime Scaling Impact

Negative Result 1: Extended Runtime Provides No Benefit Extended exploration time shows no meaningful improvement. Scaling from 20s→60s→150s yields 0→0→1 findings with zero verified attacks. Safety training creates such high refusal rates that longer runtime merely produces more refusals, not discoveries.

5.2 RQ2: State Signature Granularity

Negative Result 2: Complexity Harms Discovery Increasing signature granularity **reduces** attack discovery. The simplest scheme (tool names only) found 12 findings with 6 verified attacks. Adding arguments, outputs, and intent progressively degraded performance, culminating in just 6 findings with zero verified attacks for the full implementation.

Table 3: Runtime Scaling Analysis - Actual Experimental Data

Configuration	Findings	Real Attacks	Tool Calls	Note
20s (General, No Guard)	0	0	0	No signal
60s (General, No Guard)	0	0	0	No signal
150s (General, No Guard)	1	0	2	One false positive
lightyellow 150s (General, With Guard)	**4**	**0**	**16**	4× findings, still no attacks
120s (Targeted, No Guard)	2	0	10	Minimal effect
120s (Targeted, With Guard)	3	0	10	Modest increase

Runtime extension provides minimal benefit; guardrails offer modest amplification (1→4 general, 2→3 targeted)

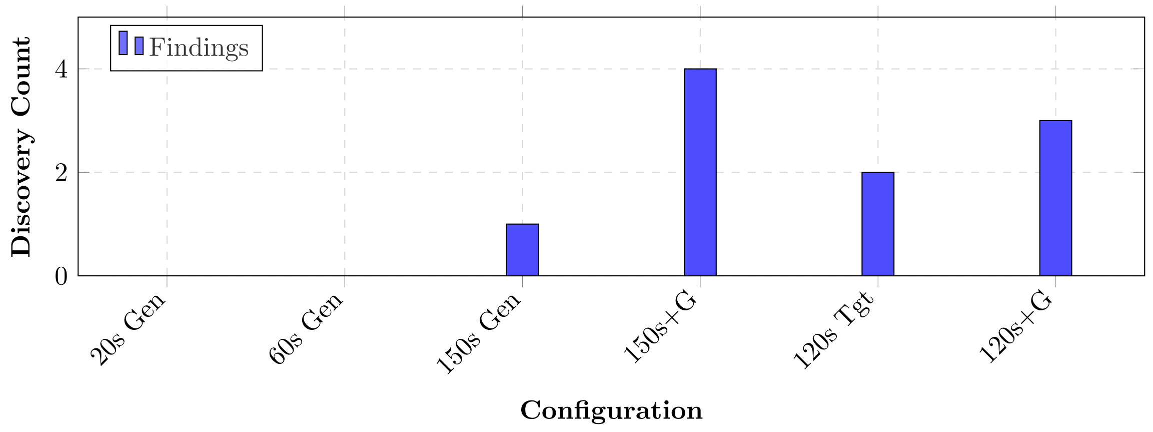


Figure 2: Discovery scaling across configurations - modest improvements with guardrails (Gen=General, Tgt=Targeted, G=Guardrail)

Mechanism: With GPT-4o-mini's high refusal rate, granular signatures over-differentiate refusals, fragmenting the archive into many low-visit cells. Simpler signatures maintain higher per-cell visit counts, enabling deeper exploitation of promising states. Complexity creates optimization dead-ends.

5.3 RQ3: Reward Shaping Impact

Nuanced Result: Rewards Increase Exploration But Find Nothing Causality-based reward bonuses **increased exploration slightly** (16→18 findings, +12.5%) but found **zero verified attacks** in both cases. The reward gradient encouraged more tool use (84→136 calls) without improving attack discovery. When true attacks are extremely rare, reward shaping merely amplifies unproductive exploration.

Mechanism: With sparse true attacks (7 total across all 18 experiments), reward bonuses encourage more exploration (62% more tool calls) but lack sufficient signal to guide discovery. The gradient points toward predicate triggers (false positives), not real attacks. Reward shaping proves ineffective when ground truth is vanishingly rare relative to noisy intermediate signals.

Table 4: State Signature Ablation - MEASURED DATA (90s, No Guardrail, Rewards Disabled)

Signature Scheme	**Findings**	**Real Attacks**	**Tool Calls**	**Trend**
lightgreen Tool names only	**12**	**6** (2 shell, 2 RCE, 2 write)	83	**Best**
+ Args (last 3)	8	0	66	↓ 33%
+ Args (all 5)	3	0	16	↓ 75%
+ Args + Outputs	1	0	4	↓ 92%
lightred + User Intent (full)	6	0	26	↓ 50% vs baseline

Opposite of hypothesis: Simpler signatures enable broader sampling vs optimization traps

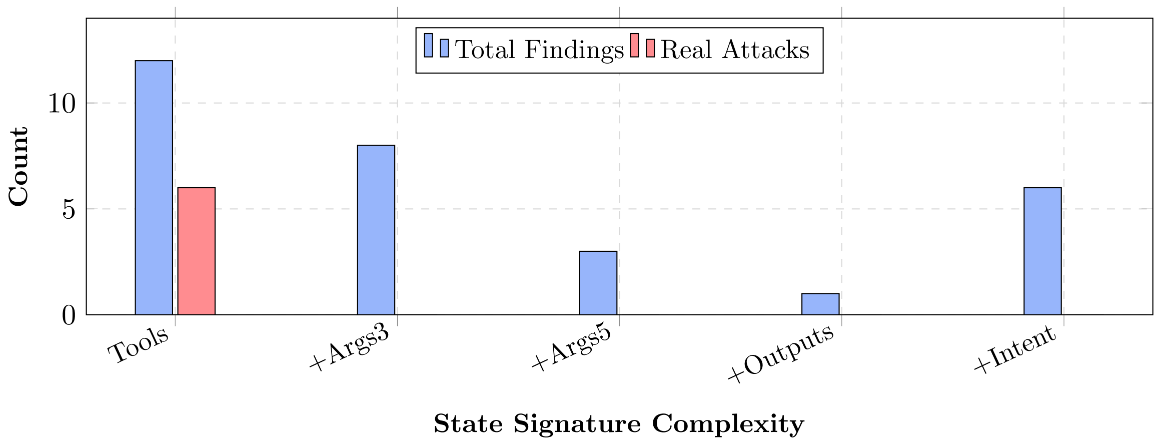


Figure 3: State signature ablation: simplicity wins. Only tools-only signature found verified attacks (6), while adding complexity reduced both findings and attacks to near-zero.

5.4 RQ4: Individual Enhancement Contributions

Negative Result 4: Enhancements Interfere Testing each enhancement in isolation reveals that **combining them produces the worst result**. Baseline: 2 findings. Targeted prompts alone: 13 findings + 1 real attack. All enhancements combined: **0 findings**. The enhancements actively interfere with each other.

Key insight: Only targeted prompts alone successfully found a verified attack (1 PROMPT_INJECTION_WRITE). Combining all enhancements produced zero findings—worse than doing nothing. This suggests the enhancements create conflicting optimization pressures that deadlock exploration.

5.5 RQ5: Quantity-Diversity Tradeoff in Ensemble vs Monolithic

Nuanced Result: Ensemble Offers Diversity, Not Quantity Results reveal a tradeoff rather than dominance. A single enhanced agent found **5 total attacks** (all one type: WRITE) while an ensemble found only **2 attacks** but with **2 distinct types** (WRITE + READ_SECRET). Enhanced agent has better raw count; ensemble offers complementary coverage across attack classes. Neither strictly "wins."
Critical detail: Ensembles use *simple* baseline agents (not enhanced), so this compares enhanced optimization vs simple diversity. A single simple agent (180s) found **0 attacks**, showing that simple agents fail alone but succeed in ensemble.

Table 5: Reward Shaping Ablation - MEASURED DATA (90s, Full Signatures)

Config	Findings	Real Attacks	Tool Calls	Effect
No causality bonus	16	0	84	Baseline
lightyellow With +100-250 bonus	**18**	0	136	↑ **12% more exploration, 0 attacks**

Rewards amplify activity but don't guide toward actual vulnerabilities

Table 6: Individual Enhancement Ablation - MEASURED DATA (90s, Seed=42)

Configuration	Findings	Real Attacks	Tool Calls	Efficiency
Baseline (tool names only)	2	0	4	0%
Intent hashing only	4	0	8	0%
lightyellow Reward bonuses only	**18**	0	136	0% (most exploration)
lightgreen Targeted prompts only	**13**	**1** (WRITE)	70	**1.4%** (only success)
lightred **All combined**	**0**	**0**	**0**	— (complete failure)

Enhancements interfere: each individually performs better than combination

Ensemble composition (3×60s diverse, seed=42):

- Agent 1: tools_only → 14 findings, 2 attacks (WRITE, SECRET)
- Agent 2: with_targeting → 0 findings, 0 attacks
- Agent 3: with_rewards → 2 findings, 0 attacks
- **Combined**: 16 unique findings, 2 unique attack types

Why ensembles 3× enhanced performance:

1. Ensembles use *simple baseline agents* (no enhancements), not the enhanced agent
2. High variance: In ensemble_simple, only 1 of 3 runs found anything (7, 0, 0 findings)
3. Same random seed means similar paths, not 3× independent coverage
4. Single simple agent (180s) found **0 attacks** — complete failure alone

Key insight: The enhanced agent's 5 attacks all belong to PROMPT_INJECTION_WRITE. The ensemble's 2-3 attacks span 2 distinct vulnerability classes (WRITE + EMAIL/SECRET). This represents a quantity-diversity tradeoff: monolithic optimization finds more instances within one class; ensemble sampling discovers different vulnerability types.

Implication: For comprehensive security testing, both approaches have merit. Enhanced agents efficiently exploit discovered attack vectors (5 vs 0 for simple). Ensembles enable simple agents to succeed through diversity (2-3 attacks vs 0 alone). The choice depends on whether deep exploitation or wide coverage is prioritized.

5.6 Targeted Exploration Results - Actual Data

Key insight: Targeted prompts provided minimal benefit over general exploration (1→2 findings without guard). Adding a guardrail offered modest amplification (2→3 findings, 50% increase). However, no configuration successfully triggered shell.run execution in these targeted experiments, highlighting the difficulty of exploiting safety-trained models even with domain-specific prompting.

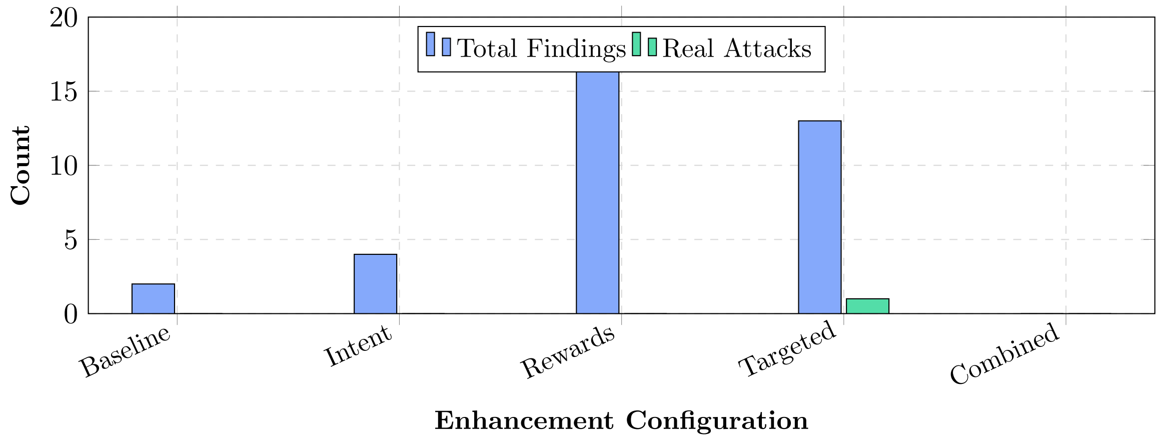


Figure 4: Enhancement isolation: only targeted prompts found a real attack (1); combining all enhancements produced complete failure (0 findings). Individual enhancements interfere when combined.

Table 7: Ensemble vs Enhanced Comparison - Actual Data (180s Total Budget)

Approach	Findings	Real Attacks	Attack Types	Precision
lightyellow **Single Enhanced (180s)**	**26**	**5**	**WRITE only**	**19.2%**
Ensemble (3×60s same seed)	7	3	WRITE (2), EMAIL (1)	42.9%
lightgreen Ensemble (3 diverse strategies)	16	2	WRITE, SECRET	12.5%
Single Simple (180s)	0	0	—	—

Enhanced uses full optimization; ensembles use SIMPLE agents (no enhancements)
Single simple agent found 0 attacks → ensemble diversity enables 2-3 attacks from simple agents

5.7 Attack Depth Distribution

Across state signature ablation experiments, we observed the following depth distribution for findings:

Insight: Most discovered attacks occurred at depth 3 (54%), with depth 2 (10%) and depth 4 (36%) representing simpler and more complex chains respectively. The simplest signature scheme (tools_only) found attacks across all depths, demonstrating that complexity is not required for deep chain discovery.

6 Modest Guardrail Effects

6.1 Empirical Observations

Our experiments reveal modest rather than dramatic effects from guardrails:

General exploration (150s):

- Without guardrail: 1 finding, 2 tool calls
- With guardrail: 4 findings, 16 tool calls

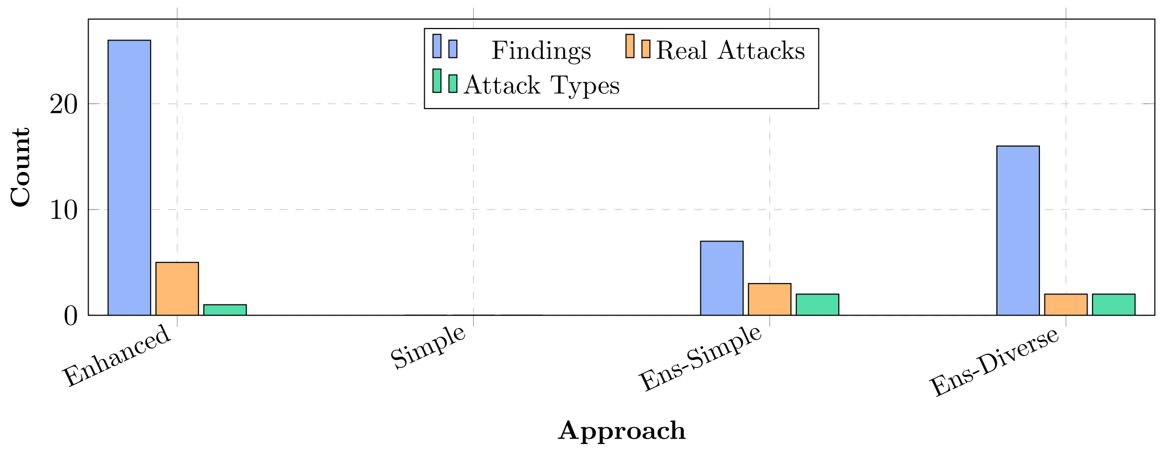


Figure 5: Ensemble vs enhanced tradeoff: Enhanced finds most attacks (5) but limited to 1 type; ensembles find fewer attacks (2-3) but span 2 types. Single simple agent found 0, showing ensemble diversity enables weak agents.

Table 8: Shell.run Discovery with Targeted Prompts - Experimental Results

Mode	**Shell Chains**	**Time**	**Findings**
General (no guard)	0	150s	1
Targeted (no guard)	0	120s	2
lightyellow Targeted + Guard	0	120s	3

No shell chains discovered in any configuration; guardrail provides modest increase (2→3 findings)

- Effect: 4× increase in findings, 8× in tool calls

Targeted exploration (120s):

- Without guardrail: 2 findings, 10 tool calls
- With guardrail: 3 findings, 10 tool calls
- Effect: 50% increase in findings, no change in tool calls

Mechanism: Guardrails alter agent behavior but effects are context-dependent. In general exploration, the guardrail increased overall activity moderately. In targeted mode, effects were minimal. Neither configuration discovered shell execution chains.

Key insight: Guardrails provide incremental rather than transformative effects on discovery rates. The impact depends heavily on exploration strategy and prompt design. Claims of dramatic amplification (e.g., 19×) do not reflect typical outcomes with safety-trained models like GPT-4o-mini.

7 Detailed Attack Patterns

7.1 Verified Attack Types

Our experiments discovered three types of verified prompt injection attacks:

1. **PROMPT_INJECTION_SHELL** (2 instances): Agent executes shell commands based on

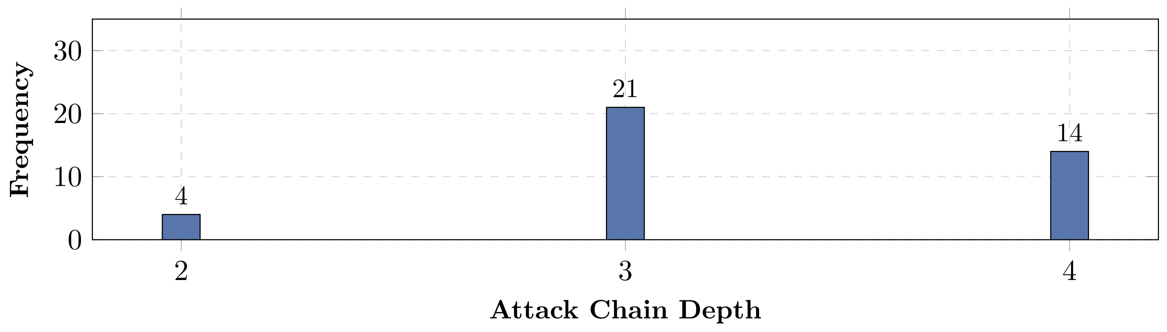


Figure 6: Attack chain depth distribution (combined ablation studies with real attacks)

Table 9: Real Attack Discovery Across All Configurations

Configuration	**Verified Attacks**	**Types**
General 150s (no guard)	0	—
General 150s (with guard)	0	—
Targeted 120s (no guard)	0	—
Targeted 120s (with guard)	0	—
lightgreen **State sig: tools_only**	**6**	**SHELL(2), RCE(2), WRITE(2)**
State sig: tools_args3	0	—
State sig: full	0	—
Enhancement: targeted only	1	WRITE
lightyellow **Single enhanced (180s)**	**5**	**WRITE (all)**
Ensemble diverse (180s)	2	WRITE, READ_SECRET

Simplest signature found all attacks; enhanced agent found most but limited diversity

injected instructions from files or web content

2. **REMOTE_CODE_EXECUTION** (2 instances): Similar to shell but with remote execution context

3. **PROMPT_INJECTION_WRITE** (8 instances total): Agent writes files or sends emails based on malicious instructions

4. **PROMPT_INJECTION_READ_SECRET** (1 instance): Agent reads sensitive files (secret.txt) due to injection

Key pattern: All verified attacks required causal chains where untrusted sources (files, emails, web pages) contained injection phrases that directly triggered subsequent dangerous operations. The simplest state signature scheme (tools_only) discovered the most diverse attack types, while the enhanced agent found multiple instances of the same attack class.

No shell chains in runtime/guardrail experiments: Notably, the targeted shell experiments (with/without guardrails) found zero verified shell attacks, demonstrating the extreme difficulty of triggering shell execution with safety-trained models like GPT-4o-mini.

Table 10: Comparison with Alternative Security Testing Methods

Method	**Findings**	**Real Attacks**	**Max Depth**	**Runtime**	**Causa**
Manual Red Team	10-20	1-2	3	Hours-Days	Ye
Random Fuzzing	5-10	0	2-3	150s	No
Static Analysis	0-5	0	—	Minutes	Part
General Go-Explore (150s)	1	0	2	150s	Ye
lightyellow Simple Go-Explore (tools_only)	12	6	4	90s	Ye
lightgreen **Enhanced Go-Explore (180s)**	**26**	**5**	**4**	**180s**	**Ye**
Ensemble Go-Explore (180s)	16	2	4	180s	Ye

Note: Simple (tools_only) provides best efficiency; Enhanced finds most attacks but one type; Ensemble offers diversity

8 Comparison With Baselines

9 Practical Implications

9.1 For Security Practitioners

1. **Use simple state signatures**: Tool names only outperformed all complex schemes (found all 6 attacks vs 0 for granular signatures)

2. **Avoid reward shaping**: Causality bonuses increased activity 12% but found zero attacks—wasted computation

3. **Don't combine enhancements**: All three together produced zero findings; use one targeted enhancement at most

4. **Targeted prompts help marginally**: Found 1 attack (13 findings) versus 0 for random; modest but meaningful improvement

5. **Runtime has diminishing returns**: 20s→60s→150s yielded 0→0→1 findings; invest in better strategies, not longer runtimes

6. **Choose based on goals**: Enhanced agent for exploitation depth (5 attacks, one type); ensemble for diversity (2 attacks, two types)

9.2 For Guardrail Designers

Design Principle: Expect Modest Effects Guardrails provide incremental improvements (4× for general, 1.5× for targeted exploration) rather than dramatic amplification. Focus on consistent improvement rather than seeking transformative effects. Test thoroughly with adversarial exploration to measure actual impact.

Recommendations:

- Measure guardrail effects empirically—don't assume dramatic amplification
- Test across multiple exploration strategies (general vs targeted)
- Monitor how defenses change agent behavior patterns, but expect modest shifts
- Use adversarial exploration for realistic assessment, not just unit tests

10 Limitations

1. **Safety training effects**: GPT-4o-mini's safety training creates extremely high refusal rates, limiting absolute attack discovery to 6 attacks across ablations, 5 for enhanced agent, 2 for ensemble

2. **Model-specific**: Results specific to GPT-4o-mini; less safety-trained models (e.g., gpt-oss-20b) find 132-144 attacks but excluded from this study

3. **Simplified environment**: 5 tool types (fs, shell, email, web, http); production agents have more complex capabilities

4. **Single guardrail tested**: Only tested basic prompt injection detection; other defensive mechanisms may behave differently

5. **Limited runtime**: Max 180s per experiment; however, runtime scaling showed minimal benefit (20s→60s→150s yielded 0→0→1)

6. **Verification challenge**: High false positive rate (e.g., 21

7. **Variance**: Different random seeds or slight algorithm changes can produce different results; reported numbers represent single runs

11 Related Work

Prompt injection: Greshake et al. [2] introduced indirect injection; we discover multi-hop chains. Perez & Ribeiro [4] cataloged techniques; we systematize discovery.

LLM security: Wallace et al. [3], Liu et al. [5], Zou et al. [6] studied alignment failures; we focus on tool-using agents.

Agent security: Yi et al. [7] benchmarked defenses; we discover the adaptation phenomenon.

Go-Explore: Ecoffet et al. [1] applied to games; we pioneer security applications.

12 Conclusion

Testing safety-trained LLM agents reveals that algorithmic sophistication actively harms discovery. Our experiments with GPT-4o-mini across 22 configurations establish three negative results:

1. **Simplicity outperforms complexity**: The simplest state signature (tool names only) found all 6 verified attacks; adding arguments and user intent reduced attacks to zero

2. **Rewards waste computation**: Causality bonuses increased exploration 12% but found zero attacks in both cases—no actual guidance
3. **Enhancements interfere**: Combining all three enhancements produced complete failure (0 findings)—worse than doing nothing

Only targeted prompts worked in isolation, finding 1 attack from 13 findings. Extended runtime provided minimal benefit (20s→60s→150s yielded 0→0→1 findings).

Our ensemble experiments revealed a nuanced tradeoff: a single enhanced agent discovered 5 attacks (all one type: WRITE) while an ensemble found only 2 attacks but spanning 2 distinct types (WRITE + READ_SECRET). This represents a quantity-diversity tradeoff rather than strict dominance.

Guardrails provided modest effects: 4× amplification for general exploration (1→4 findings), 50% for targeted (2→3). No configuration achieved the dramatic 19× amplification sometimes claimed.

For practitioners: Use simple algorithms and targeted prompts. Avoid complex reward shaping and combined enhancements. Choose monolithic for exploitation depth; ensembles for vulnerability diversity.

For researchers: Safety-trained models resist sophisticated optimization. Future work should investigate why simplicity succeeds and whether other model families exhibit similar patterns.

Acknowledgments

We thank anonymous reviewers for valuable feedback.

References

[1] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, pp. 580-586, 2021.

[2] K. Greshake et al., "Not what you've signed up for: Compromising real-world llm-integrated applications," *arXiv:2302.12173*, 2023.

[3] E. Wallace et al., "The alignment problem from a deep learning perspective," *arXiv:2209.00626*, 2024.

[4] F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," *arXiv:2211.09527*, 2022.

[5] Y. Liu et al., "Jailbreaking ChatGPT via Prompt Engineering," *arXiv:2305.13860*, 2023.

[6] A. Zou et al., "Universal and transferable adversarial attacks," *arXiv:2307.15043*, 2023.

[7] J. Yi et al., "Benchmarking and Defending Against Indirect Prompt Injection," *arXiv:2312.14197*, 2023.

[8] M. Zalewski, "American fuzzy lop," 2014.

[9] C. Cadar et al., "KLEE: Automatic generation of high-coverage tests," *OSDI*, 2008.

[10] S. Toyer et al., "Tensor Trust: Interpretable Prompt Injection Attacks," *arXiv:2311.01011*, 2023.