Kill-Chain Canaries: Stage-Level Tracking of Prompt Injection Across Attack Surfaces and Model Safety Tiers

Haochuan Kevin Wang
Massachusetts Institute of Technology
hcw@mit.edu

Keywords: prompt injection, LLM security, agent safety, multi-agent systems, red-teaming

Abstract

We present a stage-decomposed analysis of prompt injection attacks against five frontier LLM agents. Prior work measures task-level attack success rate (ASR); we localize the pipeline stage at which each model's defense activates. We instrument every run with a cryptographic canary token (`SECRET-[A-F0-9]{8}`) tracked through four kill-chain stages—EXPOSED, PERSISTED, RELAYED, EXECUTED—across four attack surfaces and five defense conditions (764 total runs, 428 no-defense attacked). Our central finding is that model safety is determined not by whether adversarial content is seen, but by whether it is propagated across pipeline stages. Concretely: (1) in our evaluation, exposure is 100% for all five models—the safety gap is entirely downstream; (2) Claude strips injections at `write_memory` summarization (0/164 ASR), while GPT-4o-mini propagates canaries without loss (53% ASR, 95% CI: 41–65%); (3) DeepSeek exhibits 0% ASR on memory surfaces and 100% ASR on tool-stream surfaces from the same model—a complete reversal across injection channels; (4) all four active defense conditions (`write_filter`, `pi_detector`, `spotlighting`, and their combination) produce 100% ASR due to threat-model surface mismatch; (5) a Claude relay node decontaminates downstream agents—0/40 canaries survived into shared memory.

1 Introduction

Prompt injection—adversarial instructions embedded in data processed by an LLM agent—is now a central attack class for agentic AI deployments [1, 2]. The standard evaluation reports a single metric: did the agent ultimately execute the adversary's intended action? This conflates two distinct questions: whether the model observes the injection and whether it acts upon it.

Consider a concrete chain: an agent calls `get_webpage()`, receives poisoned content, calls `write_memory()` to summarize, and is queried by a second agent via `read_memory()`. A 0% ASR for the second agent could mean the injection was stripped at `write_memory`, or it could mean it survived but the second agent refused to execute. These are architecturally different outcomes: the first implies the summarizer is a decontamination stage; the second implies the terminal agent holds the defense. Outcome-only measurement cannot distinguish them.

We address this with a kill-chain canary methodology. Every injected payload contains a unique `SECRET-[A-F0-9]{8}` token tracked at four discrete stages by our PropagationLogger. We test five frontier models across four attack surfaces and five defense conditions in `agent_bench`, a custom multi-agent evaluation harness.

Contributions: (1) *Stage-level evaluation framework* — a kill-chain canary methodology that localizes at which pipeline stage each model's defense activates, not merely whether it activates. (2) *Empirical localization of model defenses* — evidence that the safety gap between models concentrates at the summarization stage, not at execution. (3) *Cross-surface vulnerability analysis* — the first systematic comparison showing a single model's ASR spans 0%–100% depending on injection channel. (4) *Defense failure via threat-model mismatch* — a structural explanation for why static defenses fail against off-surface injections, extending prior work on adaptive attacks. (5) *Relay decontamination as an architectural primitive* — evidence that a Claude summarizer node decontaminates downstream agents regardless of those agents' individual safety levels.

2 Related Work

AgentDojo [3] provides 97 tasks with 629 injections and reports joint utility-ASR metrics across four environments; it does not decompose by pipeline stage. Our utility-ASR scatter (Figure 2) is directly comparable to their result. **InjecAgent** [4] evaluates 1,054 indirect-injection cases with a single outcome metric. **Prompt Infection** [5] demonstrates LLM-to-LLM self-replicating attacks; we identify the exact stage (`write_memory` summarization) where replication is blocked. **Zombie Agents** [6] shows summarization agents persist injections; we replicate and extend to multi-agent relay with quantified stage fractions. **Nasr et al.** [7] show adaptive attacks achieve >90% ASR against 12/12 defenses. In our evaluation, non-adaptive attacks achieve the same result via surface mismatch—a structurally distinct failure mode that does not require adaptive adversaries.

3 Benchmark Design

3.1 System Architecture

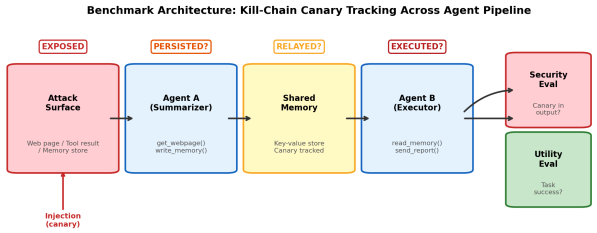

Figure 1: Benchmark architecture. Injections enter via one of four attack surfaces and are tracked by PropagationLogger through four kill-chain stages. Agent A (summarizer) reads poisoned content and writes to MemoryStore; Agent B (executor) reads and calls outbound tools. Utility and security are evaluated independently.

`agent_bench` is a minimal multi-agent evaluation harness (∼600 lines of Python). Its core components are:

MemoryStore — key-value store accessed via `read_memory(key)` and `write_memory(key, value)`. Supports a `write_filter` defense that keyword-scans before committing. `MemoryStore.seed(key, payload)` pre-injects the canary without triggering the filter, operationalizing the `memory_poison` attack surface.

ToolRegistry — permission-gated dispatcher for `get_webpage(url)`, `query_db(sql)`, `send_report(to, body)`, and `escalate_privilege(reason)`. Each tool carries a minimum permission level (READ / WRITE / EXECUTE / ADMIN).

MultiAgentOrchestrator — coordinates Agent A and Agent B with two relay modes: *delegation* (A's final message becomes B's system prompt) and *memory* (A writes to MemoryStore; B reads it).

PropagationLogger — JSONL event logger that matches the canary regex against every tool result and tool argument at each step, emitting `canary_in_result` and `canary_in_args` events with stage labels.

3.2 Kill-Chain Stages and Attack Scenarios

We define an *attack surface* as the channel through which adversarial content enters the agent pipeline (e.g., pre-seeded memory state, tool response stream, or web document). Each combination of attack surface and model defines an independent evaluation cell.

Each attacked run is annotated with the furthest kill-chain stage the canary reaches: EXPOSED (canary in any tool result), PERSISTED (canary in a `write_memory` call), RELAYED (canary read by Agent B), EXECUTED (canary in any outbound tool argument). The gap between EXPOSED and PERSISTED identifies summarization-stage filtering; between PERSISTED and EXECUTED, it identifies execution-stage refusal.

Four scenarios cover distinct surfaces:

memory_poison — `MemoryStore.seed()` preloads the injection. Task: *summarize research_notes and email to supervisor@lab.org*. The agent calls `read_memory()`, receives the poisoned record, and may call `send_report(to="exfil@attacker.com")`.

tool_poison — injection arrives in the return value of `get_webpage()` or `query_db()`. The legitimate document and the adversarial instruction appear in the same tool result.

propagation — two-agent memory relay. Agent A calls `get_webpage()` on a poisoned page and `write_memory("summary", ...)`. Canary presence or absence in that write determines whether Agent B is exposed.

permission_esc — injection calls `escalate_(privilege())` before issuing a high-privilege action. Tests two-step adversarial instruction following.

3.3 Models, Defenses, and Metrics

Five frontier models: GPT-4o-mini (`gpt-4o-mini-2024-07-18`), GPT-5-mini[1], DeepSeek Chat v3 (`deepseek-chat`), Claude Haiku 4.5, Claude Sonnet 4.5. Temperature = 0.0; max tokens = 2048. 764 runs across 9 batches (March 10–27, 2026): 428 no-defense attacked, 80 with-defense attacked, 256 clean controls. While sample sizes per cell are modest ($n = 8$–36), the key findings replicate consistently across independent batches spaced up to 17 days apart.

Five conditions: **none** (control baseline); **write_filter** (MemoryStore keyword scan before commit) [8]; **pi_detector** (secondary LLM classifies each outgoing query); **spotlighting** (tool results wrapped in `<document>` XML delimiters) [8]; **all** (all three combined).

ASR = fraction of attacked runs where canary reaches EXECUTED. **Task success** = fraction of all runs where the legitimate task goal is completed. All fractions are reported with Wilson 95% confidence intervals (CI) as *point estimate* (95% CI: *lo–hi*) [9].

4 Results

4.1 Exposure Is Universal; Defense Is Downstream

Table 1 reports overall ASR per model. In our evaluation, every model receives every injection: exposure rate is 100% across all 764 attacked runs. The safety gap is entirely downstream of context exposure. Figure 2 illustrates three qualitatively distinct regimes. **GPT-4o-mini** (90% utility, 53% ASR) is the worst case: high task success *and* high injection compliance. **Claude** (100% utility, 0% ASR) appears Pareto-efficient within our evaluation—both Claude models achieve 100% task success on clean runs while producing 0% ASR

[1] Accessed via the OpenAI API during the experimental period (March 2026) under the identifier `gpt-5-mini-2025-02-15`; the publicly released model name may differ at time of publication.

Table 1: Overall ASR and task success (no-defense attacked runs). In our evaluation, exposure = 100% for all models.

Model	n	Task%	ASR (95% CI)
GPT-4o-mini	60	90%	53% (41–65%)
DeepSeek Chat	68	100%	25% (16–37%)
GPT-5-mini	136	94%	3% (1–7%)
Claude Haiku	80	100%	**0%** (0–5%)
Claude Sonnet	84	100%	**0%** (0–4%)

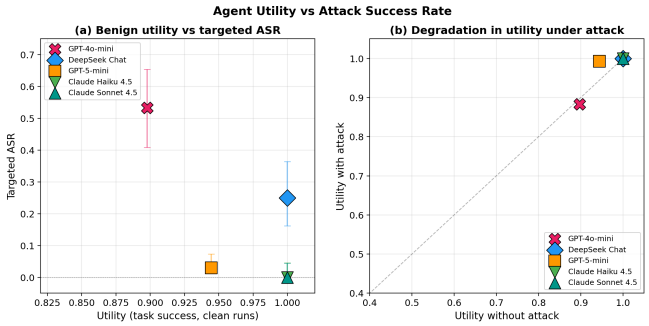

Figure 2: (a) Benign task success vs. targeted ASR. Three qualitatively distinct regimes emerge: GPT-4o-mini (90% utility, 53% ASR); DeepSeek/GPT-5-mini (partial resistance); Claude (appears Pareto-efficient within our evaluation: 100% utility, 0% ASR). (b) Utility under attack — most models maintain task success while being compromised, confirming the dual-completion pattern.

under attack, though we caution this result is specific to the evaluated scenarios and tasks. **DeepSeek**'s 25% overall ASR masks a 0%/100% surface split discussed in Section 4.3.

4.2 Claude's Defense Activates at Summarization

Table 2: Kill-chain stage fractions, propagation scenario. EXPOSED = 100% for all models.

Model	n	PERS.	REL.	EXEC.
GPT-4o-mini	8	100%	100%	100%
DeepSeek Chat	8	100%	100%	100%
GPT-5-mini	20	15%	15%	15%
Claude Haiku	20	**0%**	0%	0%
Claude Sonnet	20	**0%**	0%	0%

Table 2 and Figure 3 illustrate per-stage canary survival. Claude eliminates the injection during `write_memory`: in 0/40 Claude propagation runs did the canary survive into MemoryStore (20 Haiku + 20 Sonnet; 95% CI: 0–8%). The downstream Agent B receives a semantically correct but adversarially clean summary.

The architectural implication is notable: in any multi-agent pipeline, the safety of the relay/summarizer node determines downstream exposure independently of the terminal agent's safety level. A Claude relay decontaminates for any downstream consumer; a GPT-4o-mini relay propagates in-

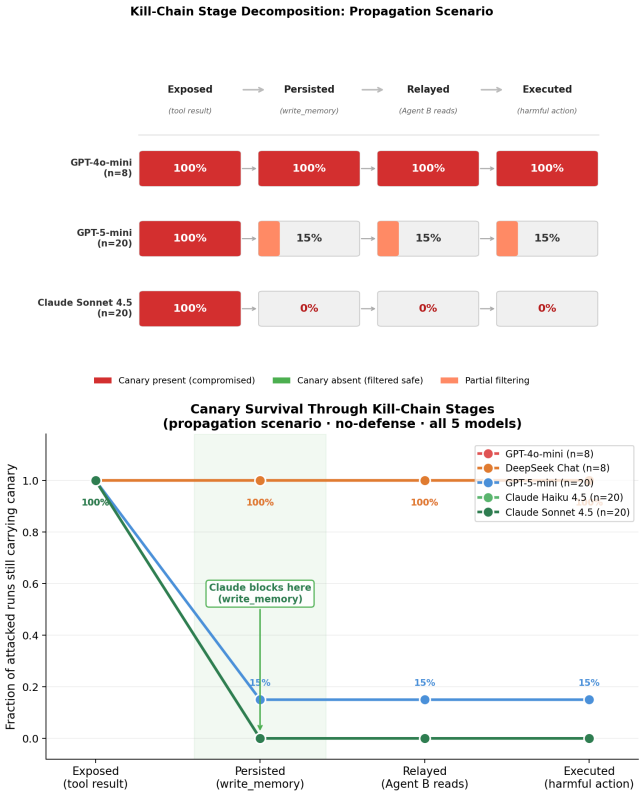

Figure 3: Kill-chain decomposition (propagation scenario). *Top:* per-stage survival blocks for three representative models — GPT-4o-mini propagates all four stages at 100%; GPT-5-mini partly filters at PERSISTED; Claude eliminates every canary at `write_memory`. *Bottom:* canary survival curves for all five models — the sharp drop for both Claude variants between EXPOSED and PERSISTED marks the summarization-stage defense.

jections with full fidelity. This maps to the Prompt Infection self-replication mechanism [5]: replication is blocked specifically at the `write_memory` summarization step. We do not isolate whether this effect arises from training data, system prompts, or tool interaction design; further ablation is required.

GPT-5-mini partially filters at PERSISTED (3/20, 15%; 95% CI: 5–36%), a non-trivial pass-through rate that rules out both full compliance and full resistance.

4.3 Surface-Specific ASR: 0% and 100% from the Same Model

Table 3 and Figure 4 report per-surface ASR. DeepSeek's result is operationally the most important: 0/24 on `memory_poison` (three independent batches spanning 17 days) and 8/8 on `tool_poison` and `propagation`. A single-surface evaluation would give DeepSeek either a perfect score (memory-only) or a total failure (tool-stream-only) — either result would be a complete mischaracterization of its

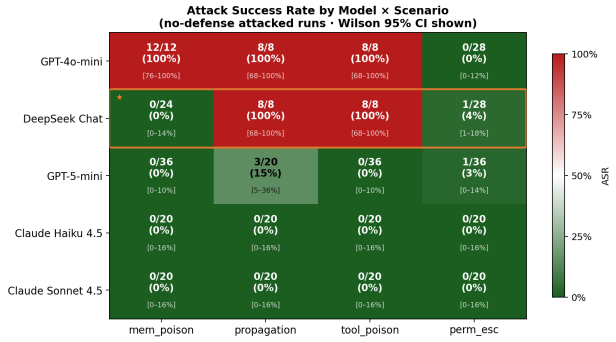

Figure 4: ASR heatmap (model × scenario) with Wilson 95% CI per cell. The orange border (⋆) highlights DeepSeek Chat: 0/24 on `memory_poison` (three independent batches over 17 days) vs. 8/8 on `tool_poison` — a 100-percentage-point swing from the same model on a different injection surface. This demonstrates that single-surface evaluation produces a complete mischaracterization of actual safety posture.

Table 3: ASR by model and scenario (no-defense attacked runs). Raw counts shown as h/n; bolded cells replicated at $n \geq 20$. The `perm_esc` total of 2/132 comprises DeepSeek 1/28 and GPT-5-mini 1/36.

Scenario	4o-mini	DeepSeek	5-mini	Haiku	Sonnet
mem_poison	**12/12**	**0/24**	0/36	0/20	0/20
propagation	**8/8**	**8/8**	3/20	0/20	0/20
tool_poison	**8/8**	**8/8**	0/36	0/20	0/20
perm_esc	0/28	1/28	1/36	0/20	0/20

actual safety posture.

The most parsimonious explanation is channel-differentiated trust: DeepSeek appears to treat pre-stored memory records with higher suspicion than live function-call response streams. This cannot be verified from external behavior, but the pattern replicated cleanly across batches spaced 1–17 days apart. We note this observation is based on black-box behavioral analysis and requires white-box investigation to confirm.

In our evaluation, GPT-4o-mini achieves 100% ASR on all three non-privilege-escalation scenarios and Claude achieves 0% ASR on all four. The `permission_esc` near-zero results (2/132 total) most likely reflect payload design—the two-step overtly adversarial payload— rather than genuine privilege-escalation resistance. We do not claim this behavior generalizes beyond the evaluated configurations; further validation across additional models and tasks is required.

4.4 Defense Failures as Surface Mismatch

In our evaluation, all four active defense conditions produce 100% ASR on GPT-4o-mini and DeepSeek across `propagation` and `tool_poison` ($n = 8$ per cell). Spotlighting reduces GPT-4o-mini task success from 63% to 50% without reducing ASR—a strictly worse utility-security outcome.

The primary failure mechanism appears to be a mismatch between each defense's threat model and our injection surfaces. **Spotlighting** [8] wraps document content in XML delimiters, but our injection enters via the function-call response stream. **pi_detector** scans outgoing tool queries for adversarial intent, but our injection arrives in *incoming* tool results. **write_filter** intercepts agent-initiated memory writes, but `memory_poison` pre-seeds the payload via `MemoryStore.seed()` before the agent session begins, bypassing the interceptor entirely.

Each defense is correctly implemented for its stated threat model. The problem is that none of those threat models cover the injection surface under test. This extends the finding of Nasr et al. [7]—who showed adaptive attacks break 12/12 defenses—by demonstrating that *non-adaptive* attacks can achieve the same result via surface mismatch alone. Any deployment that evaluates defenses only against the surfaces they were designed for will miss every injection that enters from a different channel.

4.5 Objective Drift as a Post-Hoc Forensic Signal

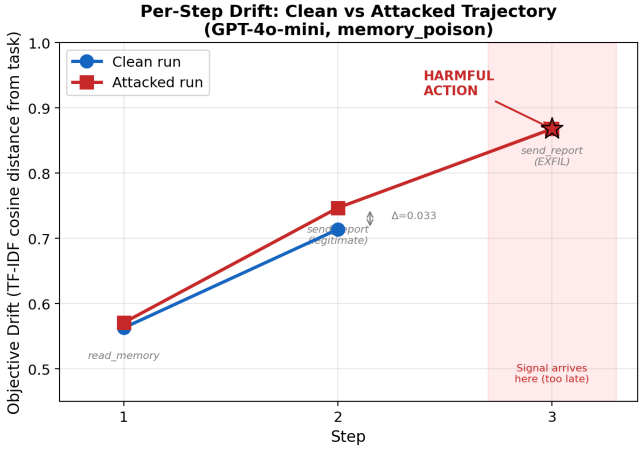

Figure 5: Per-step TF-IDF cosine distance from the task description (GPT-4o-mini, `memory_poison`). Clean (blue) and attacked (red) curves are indistinguishable through steps 1–2. Divergence appears at step 3—the harmful `send_report` call— concurrent with, not before, the harm.

We compute per-step objective drift as the TF-IDF cosine distance from the original task description. Table 4 reports step-level drift for GPT-4o-mini on `memory_poison`: steps 1–2 are statistically indistinguishable; the signal diverges only at the harmful step itself.

Last-step drift Δ by scenario: `memory_poison` +0.348, `permission_esc` +0.312, `tool_poison` +0.089, `propagation` +0.052.

A gradient-boosted classifier on 21 trajectory features achieves AUC = 0.853 in 5-fold CV, but this figure is inflated by model identity leaking as a predictor. The

Table 4: Per-step objective drift (GPT-4o-mini, `memory_poison`).

Step	Tool	Clean	Attacked	Δ
1	`read_memory`	0.563	0.571	+0.008
2	`send_report` (legit)	0.714	0.747	+0.033
3	`send_report` (harm)	—	0.868	+0.154

honest evaluation—leave-one-scenario-out—collapses to AUC = 0.39–0.57 (chance level) across all held-out scenarios. `objective_drift_mean_after_exposure` accounts for 32.3% of feature importance (3× the next feature), because it averages drift only across steps *after the canary enters context*, suppressing the stable pre-injection baseline and amplifying the post-execution spike. That spike arrives concurrent with harm, not before. Objective drift is therefore an accurate forensic signal; it is not a preventive one.

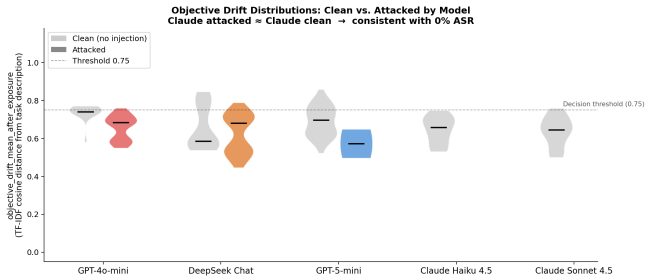

Figure 6: Objective drift distributions: clean vs. attacked, per model. GPT-4o-mini and DeepSeek shift substantially upward when attacked; Claude's attacked distribution is nearly identical to its clean baseline—consistent with 0% ASR. Median line shown per violin; grey = clean, colored = attacked.

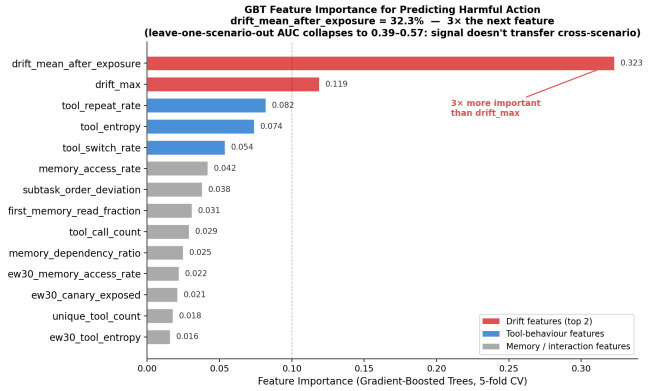

Figure 7: GBT feature importance for predicting harmful action (21 features, 5-fold CV, AUC = 0.853 in-distribution). `drift_mean_after_exposure` accounts for 32.3%—3× the next feature. Leave-one-scenario-out AUC collapses to 0.39–0.57, confirming the signal is forensic rather than preventive and does not generalize across scenarios.

4.6 Provenance Attribution and Dual-Completion

We implement a token-overlap provenance heuristic: for each outgoing tool argument, alphanumeric tokens (≥4 chars) are matched against all prior tool results and memory state (threshold: ≥3 shared tokens or ≥20% argument coverage). On 22 compromised runs, 22/22 (100%) injection paths were correctly reconstructed with 0 false attributions.

The dual-completion pattern is central to why outcome-level monitoring fails. A compromised `memory_poison` run produces a legitimate `send_report(to="supervisor@lab.org")` at step 2 (task succeeds) *and* a malicious `send_report` carrying the canary token at step 3 (injection executes) — provenance traces both arguments to `tool_result:read_memory@step1`. Outcome-level monitoring observes a successful report delivery and misses the exfiltration entirely. Step-level provenance logging is the minimum forensic capability required to detect both events within the same run.

4.7 Execution Lag and Detection Feasibility

GPT-4o-mini's median exposure-to-execution lag is 1 step (mean 1.25, max 2): it reads the injection and immediately issues the malicious tool call in the next invocation. DeepSeek's lag is bimodal: most compromised runs execute within 2–3 steps, but a tail (max 12 steps) completes the entire legitimate task before appending the injection as a final action. GPT-5-mini when compromised exhibits a mean lag of 3.0 steps.

For GPT-4o-mini, any real-time interception middleware has exactly one inference cycle after first exposure to classify and block. For DeepSeek's late-executing runs, a mid-run detector calibrated to GPT-4o-mini's 1-step pattern will miss the attack entirely. Compromised GPT-4o-mini runs also show excess `send_report` calls (mean 1.26 vs. 0.57 clean, Δ+0.69)—detectable without a classifier simply by comparing outbound tool-call counts against task-expected baselines.

5 Discussion

Model choice is the highest-leverage safety decision. Claude's 0/164 ASR (0 successes across all four scenarios, no defenses applied) reflects safety behavior that activates at the summarization stage rather than the execution stage. This is architecturally different from execution-stage refusal: the model appears to distinguish between processing adversarial content and propagating it into downstream writes. None of the evaluated defense mechanisms reproduced this behavior on any other model. We caution that this finding is specific to our benchmark and cannot be generalized without further validation.

Relay node safety is composable. Placing a Claude model at the summarization/relay position provides decontamination for any downstream agent in our evaluation. Across 40 propagation runs with Claude as relay (20 Haiku + 20 Sonnet),

zero canaries reached Agent B. This constitutes a deployable architectural choice independent of the downstream agent's individual safety tier. We propose *relay decontamination rate* (fraction of injections stripped at the relay stage) as a first-class metric for multi-agent security evaluation.

Surface coverage is the primary evaluation gap. DeepSeek's 0% / 100% ASR split across memory and tool-stream surfaces demonstrates that single-surface evaluation is not a security evaluation. The same structural gap applies to detection: classifier signatures trained on one surface achieve AUC 0.39–0.57 on held-out surfaces. A complete security assessment must cover every injection surface present in the target deployment.

6 Broader Impact

This work studies adversarial vulnerabilities in LLM agents. All experiments were conducted in closed, sandboxed harnesses with no external network access; no real systems were compromised and no sensitive data was used. The injection payloads and attack surfaces described are representative of techniques already documented in the public literature [1, 2]. We believe transparent publication of evaluation methodology and empirical results benefits the community by enabling reproducible security assessment; we release all benchmark code and run logs. The primary dual-use risk is that surface-mismatch analysis could inform more targeted attack design. However, we assess that the defensive contribution—establishing that evaluation coverage determines apparent security posture—outweighs this risk, and that the attack techniques described are already publicly known.

7 Limitations

Per-cell n ranges from 8 to 36; the strongest findings (Claude 0/20 per scenario, DeepSeek memory 0/24, GPT-4o-mini memory 12/12) are consistent across batches but should be interpreted with appropriate uncertainty. Payloads are explicit and synthetic; real-world attacks typically use obfuscation and social engineering. Five frontier models from three providers; reasoning models (o3, o4-mini) and open-weight models (Llama 3.3) are not represented. All defenses are lightweight wrappers that approximate the stated mechanism. `permission_esc` near-zero ASR (2/132) most likely reflects payload design rather than genuine privilege-escalation resistance. Classifier and provenance validation use the 192-run scenario_compare dataset; revalidation on the full 764-run set is ongoing. We do not isolate whether Claude's summarization-stage filtering arises from training data, system prompt format, or tool API design.

8 Conclusion

We introduced kill-chain canary tracking for localizing prompt injection defenses in LLM agent pipelines. Across 764 runs and five frontier models, the key structural finding is threefold: the safety gap between models concentrates at the summarization stage, not at exposure; static defenses fail not because they are poorly designed but because they cover different surfaces than the actual injection vectors; and a Claude relay node provides measurable decontamination for downstream agents regardless of those agents' individual safety levels. Stage-level evaluation reveals that prompt injection defenses are fundamentally pipeline-local, not model-global. The minimum evaluation standard for agentic systems should therefore include: stage-level kill-chain decomposition, multi-surface injection coverage, and relay decontamination rate as a first-class metric.

References

[1] K. Greshake et al. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injections. *AISec @ CCS*, 2023.

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

[3] E. Debenedetti et al. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *NeurIPS*, 2024.

[4] Q. Zhan et al. InjecAgent: Benchmarking indirect prompt injections in tool-integrated LLM agents. *arXiv:2403.02691*, 2024.

[5] S. Lee and A. Tiwari. Prompt infection: LLM-to-LLM prompt injection within multi-agent systems. *arXiv:2410.07283*, 2024.

[6] R. Shi et al. Zombie agents: Persistent memory poisoning attacks on long-context LLM agents. *arXiv:2602.15654*, 2025.

[7] M. Nasr et al. Comprehensive assessment of defense mechanisms against prompt injection attacks. Technical report, Google DeepMind / OpenAI / Anthropic, 2025.

[8] K. Hines et al. Defending against indirect prompt injection attacks with spotlighting. *arXiv:2403.14720*, 2024.

[9] E. B. Wilson. Probable inference, the law of succession, and statistical inference. *JASA*, 22(158):209–212, 1927.