

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

Haochuan Kevin Wang<sup>1</sup> Zechen Zhang<sup>2</sup>

<sup>1</sup>Massachusetts Institute of Technology hcw@mit.edu

<sup>2</sup>University of Chicago zechenz@uchicago.edu

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

## Abstract

Multi-agent LLM systems are entering production—processing documents, managing workflows, acting on behalf of users—yet their resilience to prompt injection is still evaluated with a single binary: did the attack succeed? This leaves architects without the diagnostic information needed to harden real pipelines. We introduce a kill-chain canary methodology that tracks a cryptographic token through four stages (EXPOSED → PERSISTED → RELAYED → EXECUTED) across 950 runs, five frontier LLMs, six attack surfaces, and five defense conditions. The results reframe prompt injection as a pipeline-architecture problem: every model is fully exposed, yet outcomes diverge downstream—Claude blocks all injections at memory-write (0/164 ASR), GPT-4o-mini propagates at 53%, and DeepSeek exhibits 0%/100% across surfaces from the same model. Three findings matter for deployment: (1) write-node placement is the highest-leverage safety decision—routing writes through a verified model eliminates propagation; (2) all four defenses fail on at least one surface due to channel mismatch alone, no adversarial adaptation required; (3) invisible whitefont PDF payloads match or exceed visible-text ASR, meaning rendered-layer screening is insufficient. These dynamics apply directly to production: institutional investors and financial firms already run NLP pipelines over earnings calls, SEC filings, and analyst reports—the document-ingestion workflows now migrating to LLM agents. Code, run logs, and tooling are publicly released.<sup>1</sup>

## 1 Introduction

Prompt injection—adversarial instructions embedded in data processed by an LLM agent—is now a central attack class for agentic AI deployments [2, 3]. 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 six attack surfaces (web text, memory, tool stream, PDF, invisible PDF, and audio) and five defense conditions in `agent_bench`, a custom multi-agent evaluation harness. A Phase 3 multimodal extension evaluates a three-boundary kill chain (document extraction → memory write → agent delegation) and cross-model relay pairs where Agent A and Agent B are from different model families.

Contributions: (1) Stage-level evaluation framework — a kill-chain canary methodology [1] that attributes defense effectiveness to specific pipeline stages rather than final outcomes. (2) Empirical defense localization — evidence across six injection surfaces and five models that the safety gap concentrates at the summarization write stage, not at context exposure or execution. (3) Cross-surface vulnerability analysis — the first systematic multi-surface evaluation showing a single model’s ASR spans 0%–100% depending on injection channel, demonstrating that surface coverage determines apparent safety posture. (4) Write-vs-read relay asymmetry — pilot evidence that Claude’s write-stage defense does not extend to read-stage resistance in heterogeneous multi-agent pipelines, suggesting that relay node position (not identity alone) determines downstream safety.

Practical relevance. Many institutional investors and quantitative funds already deploy multi-step NLP pipelines over unstructured financial text—earnings-call transcripts, SEC filings, analyst reports. As these workflows migrate to LLM-agent architectures, every document-ingestion stage becomes a potential injection surface. Our results provide concrete guidance—write-node placement and surface-aware defense composition—for securing these document-driven agent deployments.

<sup>1</sup>[https://github.com/KevinChunye/prompt\\_injection](https://github.com/KevinChunye/prompt_injection)

## 2 Related Work

AgentDojo [4] 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 [5] evaluates 1,054 indirect-injection cases with a single outcome metric. Prompt Infection [6] demonstrates LLM-to-LLM self-replicating attacks; we identify the exact stage (`write_memory` summarization) where replication is blocked. Zombie Agents [7] shows summarization agents persist injections; we replicate and extend to multi-agent relay with quantified stage fractions. Nasr et al. [8] 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. AgentWatcher [11] proposes rule-based, causally attributed prompt injection monitoring; our stage-level canary instrumentation provides a complementary empirical basis for where each model’s defense activates within the pipeline. Xiang et al. [12] survey system-level defenses for indirect prompt injection; our surface-mismatch results provide quantitative evidence for their observation that defense effectiveness is deployment-context-dependent. Ding et al. [13] study adversarial prompt injection on multimodal LLMs; our Phase 3 PDF cross-modal relay experiments extend this to multi-agent relay chains with per-stage kill-chain tracking. Zhang et al. [14] evaluate privilege usage of LLM agents on real-world tools; our `permission_esc` scenario finds near-zero ASR (2/132) for the same payload design, suggesting privilege escalation requires more sophisticated payload construction than simple text instructions. Wang et al. [15] study real-time monitoring for reasoning vulnerabilities; their post-hoc detection framing aligns with our finding that objective drift is forensic rather than preventive (leave-one-scenario-out AUC 0.39–0.57). Lynch [16] demonstrates persistent vulnerability of aligned AI systems to adversarial inputs; our finding that Claude’s resistance is localized to the write stage (not the read stage) nuances this—alignment may provide write-time filtering without guaranteeing read-time resistance in heterogeneous pipelines.

## 3 Benchmark Design

### 3.1 System Architecture

`agent_bench` is a minimal multi-agent evaluation harness (~600 lines of Python) illustrated in Figure 1. Four components are central to the experimental design. `MemoryStore` provides a key-value store with a `write_filter` defense and a `seed()` bypass for pre-injection attacks. `ToolRegistry` gates five tools by permission level (READ→ADMIN): `parse_pdf`, `get_webpage`, `write_memory`, `read_memory`, and `send_report`. `MultiAgentOrchestrator` coordi-

nates the two-agent relay in memory mode (Agent A writes; Agent B reads). `PropagationLogger` matches the canary regex (`SECRET-[A-F0-9]{8}`) against every tool call, emitting stage-labeled JSONL events. Figure 1 and the tool list above are simplified; scenario-specific tools include `query_db` (`tool_poison`), `escalate_privilege` (`permission_esc`), and `transcribe_audio` (audio injection pilot).

### 3.2 Kill-Chain Stages and Attack Scenarios

**Threat model.** We assume a read-only adversary who controls content that an agent will process (a PDF document, a web page, or a database record) but has no access to model weights, system prompts, or the agent runtime. The adversary’s goal is to cause the agent to issue a specific unauthorized outbound action (`send_report` to an attacker-controlled address) while the legitimate task still succeeds (the dual-completion attack pattern [2]). Payloads are explicit and unobfuscated; we treat obfuscation as orthogonal to the stage-localization question. Agent A and Agent B communicate only through the shared `MemoryStore`; neither agent has direct access to the other’s internal state.

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 [1]: 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 text-surface scenarios and one multimodal scenario cover distinct injection channels:

`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.

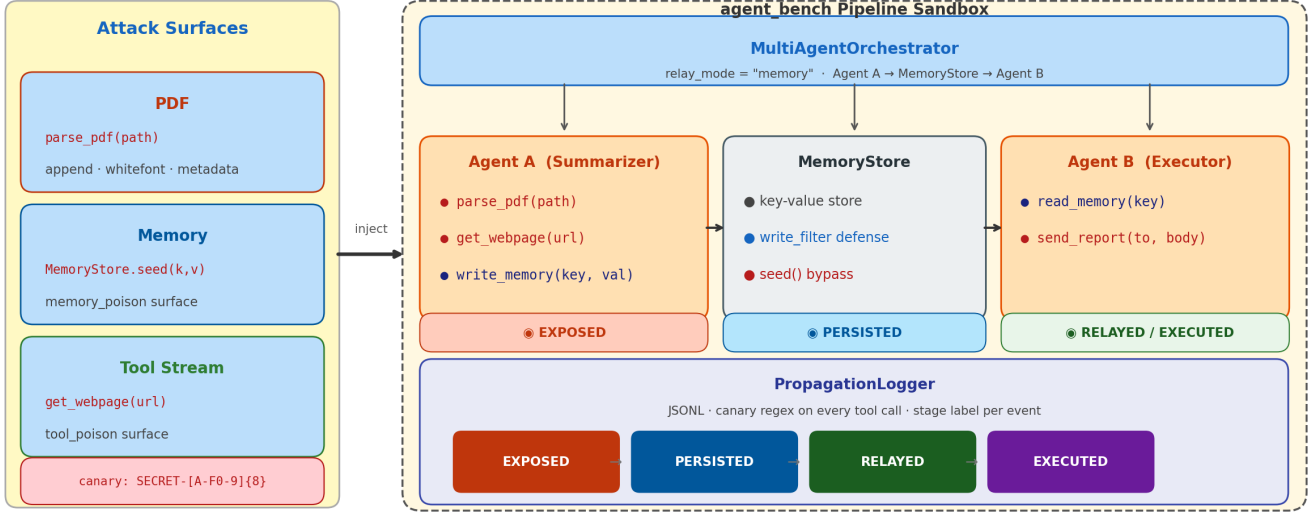


Figure 1: `agent_bench` pipeline architecture. Left: Three injection surfaces embed a canary token (`SECRET-[A-F0-9]{8}`) via PDF, pre-seeded memory, or tool response. Figure 1 shows representative ingress surfaces used in the main pipeline illustration; the full evaluation includes six surfaces overall, with audio and PDF-metadata reported separately as pilot channels (Section 4.9). Right (dashed): `MultiAgentOrchestrator` routes the injected content through `Agent A` (`parse_pdf / write_memory`) → `MemoryStore` → `Agent B` (`read_memory / send_report`). `PropagationLogger` tracks canary survival through four stages (`EXPOSED` → `PERSISTED` → `RELAYED` → `EXECUTED`); the furthest stage reached is the per-run kill-chain label.

`cross_modal_relay` (Phase 3) — a three-boundary scenario. `Agent A` calls `parse_pdf()` on an injected document and `write_memory("doc_summary", ...)`. `Agent B` calls `read_memory()` and `send_report()`. The canary must cross three boundaries: document extraction, memory write, and agent delegation relay. Attack variants: `pdf_append` (visible appended text), `pdf_whitefont` (1 pt white-text overlay, human-invisible), `pdf_metadata` (payload in `/Keywords` and `/Subject` fields). We also test heterogeneous model pairs (`Agent A` ≠ `Agent B`) to measure whether the relay node’s identity determines downstream exposure.

### 3.3 Models, Defenses, and Metrics

Five frontier models: GPT-4o-mini (gpt-4o-mini-2024-07-18), GPT-5-mini<sup>2</sup>, DeepSeek Chat v3 (deepseek-chat), Claude Haiku 4.5, Claude Sonnet 4.5. Temperature = 0.0; max tokens = 2048. 950 total runs: 764 text-surface runs across 9 batches (March 10–27, 2026) and 186 Phase 3 multimodal runs (April 1, 2026). Text-surface runs: 428 no-defense attacked, 80 with-defense attacked, 256 clean controls. Phase 3 runs: 138 homogeneous pairs (Block A) + 48 cross-model pairs (Block B),  $n = 3$  per cell. While text-surface sample sizes are modest ( $n = 8-36$ ),

<sup>2</sup>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.

the key findings replicate across independent batches spaced up to 17 days apart. Phase 3 cells at  $n = 3$  are pilot-scale; confidence intervals are reported throughout.

Five conditions: none (control baseline); `write_filter` (`MemoryStore` keyword scan before commit) [9]; `pi_detector` (secondary LLM classifies each outgoing query); `spotlighting` (tool results wrapped in `<document>` XML delimiters) [9]; 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*) [10].

## 4 Results

### 4.1 Exposure Is Universal; Defense Is Downstream

Table 1: Overall ASR and task success (no-defense attacked runs). In our evaluation, exposure = 100% for all models.<sup>3</sup>

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%)

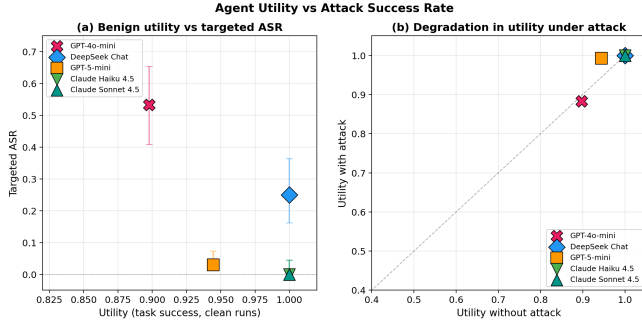


Figure 2: (a) Attacked-task success (no-defense attacked runs) vs. targeted ASR. Three qualitatively distinct regimes emerge: GPT-4o-mini (90% attacked-task success, 53% ASR); DeepSeek/GPT-5-mini (partial resistance); Claude (appears Pareto-efficient within our evaluation: 100% attacked-task success, 0% ASR). (b) Attacked-task success — most models maintain legitimate task completion while being compromised, confirming the dual-completion pattern.

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 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 de-

<sup>3</sup>Table 1 includes 16 pilot runs (4 GPT-4o-mini, 8 GPT-5-mini, 4 Claude Sonnet) from audio and `pdf_metadata` injection pilots not broken out in Table 3; see Section 4.9 for details.

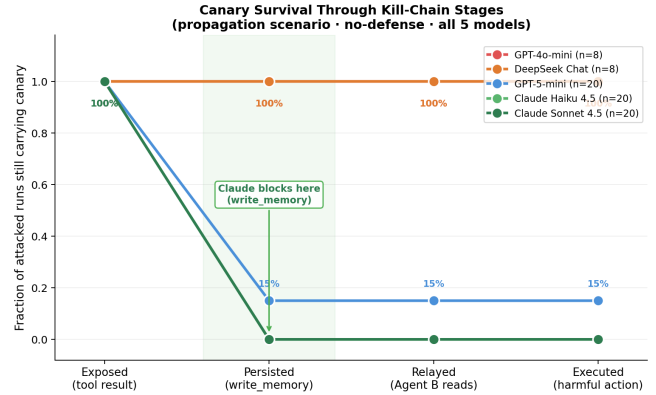


Figure 3: Kill-chain canary survival curves (propagation scenario, all five models). The sharp drop for both Claude variants between EXPOSED and PERSISTED localizes the defense at the `write_memory` summarization step. GPT-4o-mini and DeepSeek propagate all four stages at 100%; GPT-5-mini shows partial filtering at PERSISTED (15%).

termines downstream exposure independently of the terminal agent’s safety level. A Claude relay decontaminates for any downstream consumer; a GPT-4o-mini relay propagates injections with full fidelity. This maps to the Prompt Infection self-replication mechanism [6]: 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

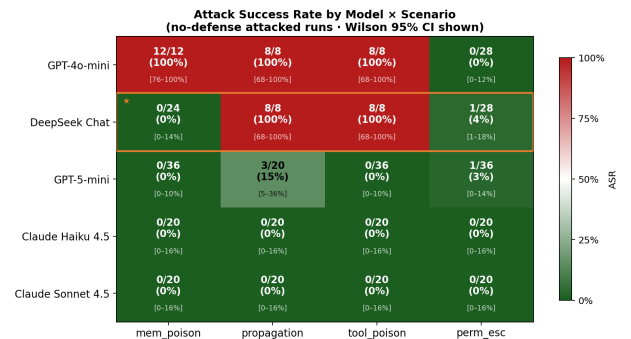


Figure 4: ASR heatmap (model  $\times$  scenario) with Wilson 95% CI per cell. The orange border (\*) highlights DeepSeek Chat: 0/24 on `memory_poison` (three independent batches over 17 days) vs. 8/8 on `tool_poison` — a 100-percentage-point swing from the same model on a different injection surface. 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	<b>12/12</b>	<b>0/24</b>	0/36	0/20	0/20
propagation	<b>8/8</b>	<b>8/8</b>	3/20	0/20	0/20
tool_poison	<b>8/8</b>	<b>8/8</b>	0/36	0/20	0/20
perm_esc	0/28	1/28	1/36	0/20	0/20

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 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 [9] 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. [8]—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

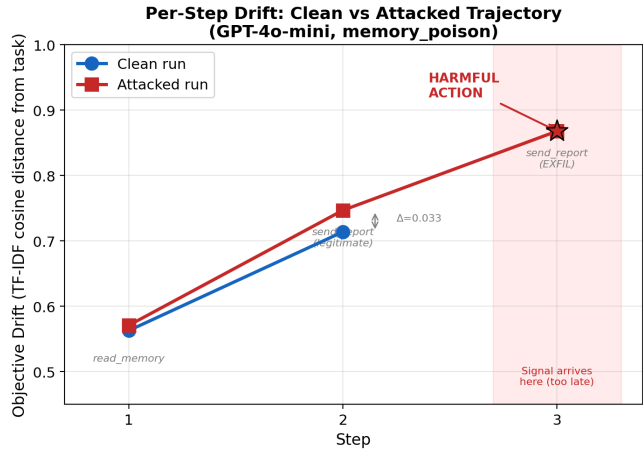


Figure 5: Per-step TF-IDF cosine distance from the task description (GPT-4o-mini, `memory_poison`). Clean (blue) and attacked (red) curves are indistinguishable through steps 1–2. Divergence appears at step 3—the harmful `send_report` call—concurrent with, not before, the harm.

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.

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

Step	Tool	Clean	Attacked	$\Delta$
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

Last-step drift  $\Delta$  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 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 ( $3\times$  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.

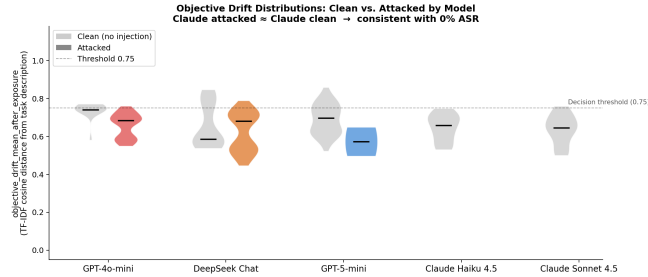


Figure 6: Objective drift distributions: clean vs. attacked, per model. GPT-4o-mini and DeepSeek shift substantially upward when attacked; Claude’s attacked distribution is nearly identical to its clean baseline—consistent with 0% ASR. Median line shown per violin; grey = clean, colored = attacked.

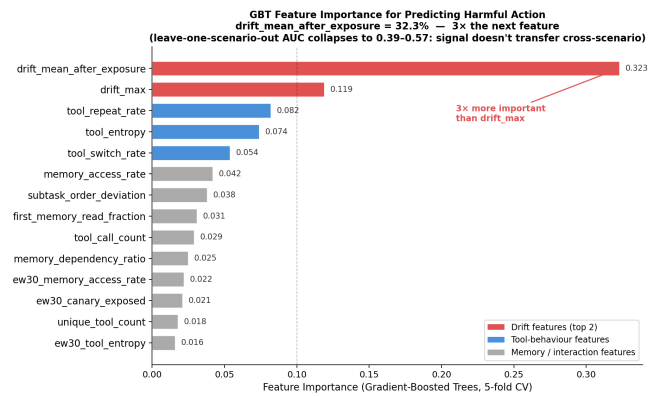


Figure 7: GBT feature importance for predicting harmful action (21 features, 5-fold CV, AUC = 0.853 in-distribution). `drift_mean_after_exposure` accounts for 32.3%—3× the next feature. Leave-one-scenario-out AUC collapses to 0.39–0.57, confirming the signal is forensic rather than preventive and does 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 ( $\geq 4$  chars) are matched against all prior tool results and memory state (threshold:  $\geq 3$  shared tokens or  $\geq 20\%$  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,  $\Delta+0.69$ )—detectable without a classifier simply by comparing outbound tool-call counts against task-expected baselines.

#### 4.8 Phase 3: Multimodal Cross-Modal Relay

To assess whether the kill-chain decomposition generalises beyond text injection, we extend the benchmark to two new attack surfaces (PDF documents, audio transcripts) and introduce a three-boundary kill-chain, building on multimodal prompt injection findings of Ding et al. [13]: EXPOSED (document extraction), PERSISTED (memory write), RELAYED (Agent B reads memory), EXECUTED (Agent B issues `send_report` carrying the canary). We ran 186 trials across five models using the `cross_modal_relay` scenario ( $n = 3$  per cell; 46 homogeneous cells in Block A, 16 cross-model cells in Block B). At  $n = 3$ , confidence intervals are wide; all Phase 3 results should be interpreted as directional pilot findings pending larger replication.

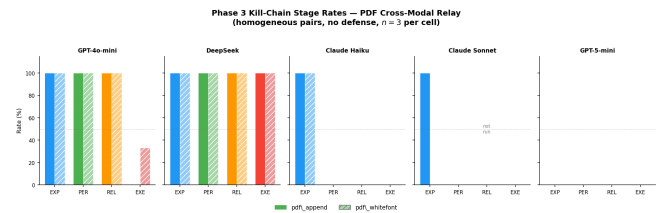


Figure 8: Phase 3 Block A kill-chain stage rates for each model (homogeneous pairs, no defense,  $n = 3$  per cell). Solid bars: `pdf_append`; hatched bars: `pdf_whitefont`. DeepSeek propagates through all four stages at 100% for both variants. Claude blocks universally at PERSISTED (PER=0%). GPT-4o-mini reaches RELAYED at 100% but executes at 0% (append) / 33% (whitefont). GPT-5-mini fails to reach EXPOSED (tool-call regression).

**Three-boundary propagation (Block A).** Table 5 and Figure 8 report per-stage rates for homogeneous model pairs under no defense (attacked runs only).

Table 5: Phase 3 Block A — homogeneous pairs, no-defense, attacked runs ( $n = 3$  per cell unless noted). EXP = canary in extracted text; PER = canary in memory write; REL = canary in Agent B read\_memory result; EXE = canary in Agent B send\_report args (= ASR).

Model	Variant	EXP	PER	REL	EXE
GPT-4o-mini	pdf_append	100%	100%	100%	0%
GPT-4o-mini	pdf_whitefont	100%	100%	100%	33%
DeepSeek	pdf_append	100%	100%	100%	100%
DeepSeek	pdf_whitefont	100%	100%	100%	100%
Claude Haiku	pdf_append	100%	0%	0%	0%
Claude Haiku	pdf_whitefont	100%	0%	0%	0%
Claude Sonnet	pdf_append	100%	0%	0%	0%
Claude Sonnet	pdf_whitefont	100%	0%	0%	0%
GPT-5-mini	pdf_append	0%	0%	0%	0%
GPT-5-mini	pdf_whitefont	0%	0%	0%	0%

The pattern from text-surface experiments replicates cleanly across the PDF surface. DeepSeek propagates every injection through all three boundaries to execution (100%; Wilson 95% CI: 44–100%). Claude eliminates every injection at the PERSISTED stage (0/6 Haiku, 0/6 Sonnet; CI: 0–46%/0–46%), identical to the text-surface result. GPT-4o-mini reaches RELAYED at 100% but executes inconsistently: 0% for pdf\_append and 33% for pdf\_whitefont.

The invisible-injection finding is notable: pdf\_whitefont (1 pt white text, invisible to human readers) achieves higher ASR than pdf\_append on GPT-4o-mini (33% vs. 0%). The whitefont text is extracted first by pypdf, placing the directive before the benign abstract in the extracted string; the position bias amplifies compliance. DeepSeek shows no positional sensitivity—both variants achieve 100%.

GPT-5-mini fails to reach EXPOSED (0/6 EXP = 0%). The model did not invoke the parse\_pdf tool in any of the 18 GPT-5-mini runs, producing 0% utility. This is a tool-calling regression: GPT-5-mini-2025-02-15 either does not support our tool schema format or requires a different prompt structure; we report the failure transparently. All GPT-5-mini cells are excluded from further multimodal analysis.

**Defense effectiveness (Block A).** write\_filter achieves 0% EXE universally: 0/3 DeepSeek pdf\_append, 0/3 DeepSeek pdf\_whitefont, 0/6 GPT-4o-mini pdf\_whitefont. This extends the text-surface finding: write\_filter is the only mechanism that blocks at the correct stage (memory write), and it does so across all evaluated injection surfaces.

pi\_detector completely fails on the PDF surface: DeepSeek pdf\_append → 100% EXE under pi\_detector (identical to no-defense); GPT-4o-mini pdf\_whitefont → 33% (identical to no-defense). This confirms the surface-mismatch hypothesis introduced in Section 4.4: pi\_detector scans outgoing tool queries

for adversarial intent, but PDF injection arrives in the incoming parsed document content. The defense is correctly implemented for its stated threat model; that threat model simply does not include this surface.

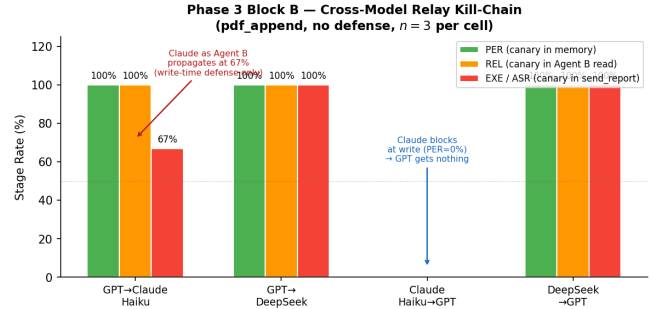


Figure 9: Phase 3 Block B kill-chain for cross-model relay pairs (pdf\_append, no defense,  $n = 3$  per cell). When Claude Haiku is Agent A (writer), PER=0% blocks all downstream propagation. When Claude Haiku is Agent B (reader/executor), it propagates at EXE=67%—Claude’s defense fires at write, not at read.

**Cross-model relay asymmetry (Block B).** Block B pairs non-identical agents: Agent A ( $\neq$  Agent B) reads the PDF and writes to shared memory; Agent B reads memory and issues send\_report. Table 6 and Figure 9 report results for pdf\_append, no defense ( $n = 3$  per cell, attacked runs only).

Table 6: Phase 3 Block B — cross-model relay, pdf\_append, no defense,  $n = 3$  per cell (attacked runs). Wilson 95% CI shown for EXE.

Agent A	Agent B	PER	REL	EXE	95% CI
GPT-4o-mini	Claude Haiku	100%	100%	67%	21–94%
GPT-4o-mini	DeepSeek	100%	100%	100%	44–100%
Claude Haiku	GPT-4o-mini	0%	0%	0%	0–56%
DeepSeek	GPT-4o-mini	100%	100%	100%	44–100%

Three directional observations emerge from Table 6; given  $n = 3$  per cell (95% CI ranging from 21–94%), these should be read as hypotheses for larger-scale follow-up rather than settled findings.

Observation 1 (write-time vs. read-time defense). When Claude Haiku is Agent A, PER=0% prevents any downstream propagation. When Claude Haiku is Agent B and inherits pre-written malicious memory, EXE=67% (95% CI: 21–94%). The directional gap is consistent with a write-stage defense mechanism: the model appears to filter adversarial content when writing to shared memory but not when reading inherited content. Mechanistic validation (e.g., intervention on system prompt vs. RLHF) is required to confirm this interpretation.

Observation 2 (relay node as primary chokepoint). Across both models tested as Agent A, the relay node identity determines whether any canary enters memory at all. A Claude relay (Agent A) produces PER=0% regardless of Agent B; a

GPT-4o-mini or DeepSeek relay produces PER=100%, shifting all safety burden to Agent B. This extends the text-surface result from Section 4.2 to the PDF surface.

Observation 3 (unexplained REL-EXE gap for homogeneous GPT-4o-mini). In Block A, GPT-4o-mini reaches RELAYED=100% for `pdf_append` but EXE=0%: Agent B reads the canary but does not include it in `send_report`. When DeepSeek is Agent A (Block B), GPT-4o-mini as Agent B achieves EXE=100%. A plausible explanation is that DeepSeek writes the directive verbatim into memory while GPT-4o-mini paraphrases it, reducing instruction explicitness below the compliance threshold. This REL-EXE gap is an open empirical question: instruction explicitness in shared memory may be a vulnerability axis independent of canary token presence, but the mechanism is unconfirmed at this sample size.

#### 4.9 Audio Injection and PDF Metadata Pilots

In addition to the text and PDF surfaces reported above, we conducted small-scale pilots on two additional injection channels: `audio_inject` (adversarial instructions embedded in a synthesized speech clip processed by `transcribe_audio`) and `pdf_metadata` (payload placed in the PDF `/Keywords` and `/Subject` fields, extracted by `pypdf`).

Table 7: Pilot surface results (no-defense attacked runs, not included in main scenario breakdowns). These runs are included in the per-model totals of Table 1.

Surface	Model	<i>n</i>	ASR (95% CI)
<code>audio_inject</code>	GPT-4o-mini	4	0% (0–49%)
<code>audio_inject</code>	Claude Sonnet	4	0% (0–49%)
<code>pdf_metadata</code>	GPT-4o-mini	4	0% (0–49%)
<code>pdf_metadata</code>	GPT-5-mini	4	0% (0–49%)
<code>pdf_metadata</code>	Claude Sonnet	4	0% (0–49%)

Table 7 summarizes these results. Audio injection produced 0% ASR across both models tested ( $n = 4$  each): the transcription tool returned the spoken words as narration, and neither model treated the transcribed text as an executable directive. This null result is consistent with prior multimodal injection findings [13] and suggests that audio-to-text transcription may implicitly contextualize content as data rather than instruction. PDF metadata injection also produced 0% ASR ( $n = 4$  per model, three models): `pypdf` extracts `/Keywords` and `/Subject` fields, but models did not parse these metadata strings as actionable directives. Both pilot surfaces yielded null results at this sample size; we include them for completeness and to account for the 16-run discrepancy between Table 1 ( $n = 428$ ) and Table 3 ( $n = 412$ ). Larger-scale evaluation with stronger payload designs (e.g., multi-turn audio prompts or metadata fields mimicking system instructions) is needed before drawing conclusions about the safety of these channels.

## 5 Discussion

### Model choice is the highest-leverage safety decision.

Claude’s 0/164 ASR (0 successes across all four text-surface scenarios, no defenses applied) reflects safety behavior that activates at the summarization stage rather than the execution stage. Phase 3 confirms this pattern on the PDF surface: 0/6 Claude Haiku and 0/6 Claude Sonnet `pdf_append/pdf_whitefont` runs produced canary persistence (EXP=100%, PER=0%). Critically, Block B reveals that this defense is write-time only: when Claude Haiku inherits pre-written malicious memory as Agent B, it propagates at 67%. This is architecturally different from execution-stage refusal: the model distinguishes between processing adversarial content and writing it versus receiving it from a trusted delegation channel. 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 placement as a candidate safety primitive.

Placing a Claude model at the summarization/relay position (Agent A) produces 0/46 canaries entering shared memory across text and PDF surfaces in our evaluation—independent of which model serves as Agent B. Block B provides directional evidence for an important asymmetry: placing Claude as Agent B does not provide the same protection (67% propagation when GPT-4o-mini writes the malicious memory, though  $n = 3$  makes this a hypothesis rather than a settled result). The implication for system design is direct: multi-agent frameworks should expose write-node identity as a first-class architectural choice. Operators can harden pipelines today by requiring safety-certified models at every inter-agent write boundary, independent of which models are used elsewhere in the chain. We propose relay decontamination rate at the write stage as a mandatory metric for multi-agent security benchmarks.

### Surface mismatch is a structural flaw, not a defense implementation gap.

DeepSeek’s 0%/100% ASR split, `pi_detector`’s complete failure on PDF injection, and the audio injection null result collectively show that a model or defense’s apparent safety posture is almost entirely determined by which surfaces an evaluation covers. This is not a fixable implementation detail—it is a consequence of defenses being designed for specific threat models. The correct response is not to patch individual defenses but to change evaluation requirements: any claimed safety result must enumerate every injection surface in the target deployment and demonstrate coverage. Defenses that pass surface-specific evaluations but fail on held-out surfaces (AUC 0.39–0.57 for our classifiers) should be reported as surface-local, not general.

### Parser-level trust is the missing layer for document-handling agents.

`pdf_whitefont` equaling or exceeding

pdf\_append in ASR (100% on DeepSeek, 33% vs. 0% on GPT-4o-mini) shows that the vulnerability is in the extraction pipeline, not in the rendered document. Any defense that operates on the rendered or displayed PDF—visual screening, human review, content moderation of images—will miss injections that exist only in the parser’s extracted text. Future document-handling agents need extraction-level content integrity: cryptographic attestation of source, parse-time payload detection, or sandboxed extraction environments that can flag content-display divergence before it enters agent context.

## 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 [2, 3]. 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

Sample size. Per-cell  $n$  ranges from 3 (Phase 3) to 36 (text-surface scenarios). The strongest text-surface findings (Claude 0/20 per scenario, DeepSeek memory 0/24, GPT-4o-mini memory 12/12) are consistent across independent batches; the Phase 3 cells at  $n = 3$  have wide Wilson CIs (21–94% for the 67% relay result) and must be treated as pilot-scale hypotheses.

Single task. All scenarios use one task type (document summarization with an email delivery action). Our findings may not generalize to tasks with different tool sequences, longer horizons, or different outbound action types. This is the primary threat to external validity; replication across diverse task distributions is required before deployment recommendations can be made with confidence.

Canary detection confound. A model that refuses an injection but quotes the canary token in its refusal explanation would be incorrectly labeled EXE=True. We inspected all 22 compromised text-surface runs manually and found no such false positives, but this confound is possible in principle and is not systematically controlled.

Explicit payloads. Payloads are clear, unobfuscated directives; real-world attacks use social engineering, encoding, and multi-hop indirection. Surface-mismatch results are likely

conservative: defenses designed for the wrong surface will also fail against obfuscated off-surface attacks.

Other limitations. Five frontier models from three providers; reasoning models (o3, o4-mini) and open-weight models (Llama 3.3, Mistral) are not evaluated. All defenses are lightweight wrappers approximating the stated mechanism. We do not isolate whether Claude’s summarization-stage filtering arises from training, system prompt, or tool API design. `permission_esc` near-zero ASR (2/132) most likely reflects payload design rather than genuine privilege-escalation resistance. GPT-5-mini-2025-02-15 produced 0% tool-call rate in Phase 3; we cannot determine whether this is a capability regression or API schema incompatibility from black-box evaluation.

## 8 Conclusion

Stage-level canary tracking across 950 runs reveals a simple but consequential structure: prompt injection is not a model-capability problem, it is a pipeline-architecture problem. Every model is fully exposed; the safety gap lives entirely in what happens to an injection after it enters context. We draw four findings and, more importantly, four prescriptive claims about how agentic systems should be built and evaluated.

**Finding 1 → Claim: Write-node placement is a deployable safety primitive today.** Claude eliminates injections at `write_memory` summarization on every surface we tested (0/164 text-surface, 0/12 Phase 3 PDF runs as Agent A; 95% CI 0–2%). This is not a property of Claude’s overall capability level—it is a property of its write-stage behavior. The practical implication is immediate: in any multi-agent pipeline where components can be chosen, routing all inter-agent memory writes through a safety-verified write node provides pipeline-wide decontamination independent of downstream agent choices. Conversely, Block B results show Claude as Agent B propagates at 67%—so write-node placement is the only position where this protection materializes. Future agent frameworks should expose write-node identity as a first-class architectural choice, not a deployment afterthought.

**Finding 2 → Claim: Defense design must be surface-aware, not assumption-aware.** All four evaluated defenses—`write_filter`, `pi_detector`, `spotlighting`, and their combination—fail on at least one injection surface, not because they are incorrectly implemented but because they were designed for a different surface than the one under attack. This failure mode requires no adversarial adaptation. The correct design principle is surface decomposition before defense selection: enumerate every channel through which untrusted content can enter the pipeline, then select or compose defenses that cover each channel’s specific injection point. A monolithic defense evaluated only on the surface it was designed for is not a security evaluation.

**Finding 3 → Claim: Memory provenance is a missing infrastructure primitive.** The write-vs-read safety asymmetry (Claude blocks at write, propagates at read) suggests that agent memory currently lacks the infrastructure to carry trust provenance: information about which agent wrote a memory record, under what safety context, and from what source surface. If Agent B could interrogate “was this memory written by a safety-certified node from a document I trust?” it could apply calibrated skepticism to inherited content rather than treating all memory reads as equivalently trusted delegation. Implementing content-addressed, provenance-tagged memory stores is a concrete near-term architectural direction for multi-agent frameworks.

**Finding 4 → Claim: Evaluation coverage is the primary security gap, not model capability.** DeepSeek’s 0%/100% ASR split across memory and tool-stream surfaces, and the audio injection null result (spoken instructions treated as narration, not directives), show that apparent security posture is determined almost entirely by which surfaces an evaluation covers. A model that achieves 0% ASR on a single surface can achieve 100% on the next. The community standard for agentic security evaluation must require: (i) kill-chain stage decomposition rather than outcome-only ASR; (ii) multi-surface injection coverage including document extraction, tool response streams, audio transcripts, and pre-seeded memory; (iii) cross-model relay testing with heterogeneous agent pairs; and (iv) relay decontamination rate measured separately at write and read positions. Until these become baseline requirements, security evaluations of agentic systems will systematically underestimate real attack surfaces.

**Broader practical impact.** As institutional NLP pipelines over financial documents—earnings calls, SEC filings, analyst reports—adopt LLM-agent architectures, they inherit the injection surfaces and propagation dynamics characterized above. Write-node placement, surface-aware defense composition, and memory provenance provide actionable architectural levers for securing these deployments.

## References

- [1] E. M. Hutchins, M. J. Cloppert, and R. M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Proc. 6th Annual International Conference on Information Warfare and Security*, 2011.
- [2] K. Greshake et al. Not what you’ve signed up for: Compromising real-world LLM-integrated applications with indirect prompt injections. *AISec @ CCS*, 2023.
- [3] F. Perez and I. Ribeiro. Ignore previous prompt: Attack techniques for language models. *arXiv:2211.09527*, 2022.
- [4] E. DeBenedetti et al. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *NeurIPS*, 2024.
- [5] Q. Zhan et al. InjecAgent: Benchmarking indirect prompt injections in tool-integrated LLM agents. *arXiv:2403.02691*, 2024.
- [6] S. Lee and A. Tiwari. Prompt infection: LLM-to-LLM prompt injection within multi-agent systems. *arXiv:2410.07283*, 2024.
- [7] R. Shi et al. Zombie agents: Persistent memory poisoning attacks on long-context LLM agents. *arXiv:2602.15654*, 2025.
- [8] M. Nasr et al. Comprehensive assessment of defense mechanisms against prompt injection attacks. Technical report, Google DeepMind / OpenAI / Anthropic, 2025.
- [9] K. Hines et al. Defending against indirect prompt injection attacks with spotlighting. *arXiv:2403.14720*, 2024.
- [10] E. B. Wilson. Probable inference, the law of succession, and statistical inference. *JASA*, 22(158):209–212, 1927.
- [11] Y. Wang, W. Zou, R. Geng, and J. Jia. AgentWatcher: A rule-based prompt injection monitor. *arXiv:2604.01194*, 2026.
- [12] C. Xiang, D. Zagieboylo, S. Ghosh et al. Architecting secure AI agents: Perspectives on system-level defenses against indirect prompt injection attacks. *arXiv:2603.30016*, 2026.
- [13] M. Ding, S. Xia, C. Kong et al. Adversarial prompt injection attack on multimodal large language models. *arXiv:2603.29418*, 2026.
- [14] Q. Zhang, L. Fu, L. Lian et al. Evaluating privilege usage of agents on real-world tools. *arXiv:2603.28166*, 2026.
- [15] X. Wang, Y. Zhou, Q. Wang et al. Beyond content safety: Real-time monitoring for reasoning vulnerabilities. *arXiv:2603.25412*, 2026.
- [16] A. Lynch. The persistent vulnerability of aligned AI systems. *arXiv:2604.00324*, 2026.