FROM PROMPT INJECTION TO PERSISTENT CONTROL: DEFENDING AGENTIC WORKSPACES AGAINST TROJAN BACKDOORS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents are evolving from conversational chatbots to operational tools in real-world workspaces. In local agentic harnesses, an LLM can read and write files, call tools, and reuse workspace state across sessions. While such capabilities enhance utility, they also expose a new attack surface for attackers. Attackers can embed a prompt injection within a file or tool output. Agents may read this hidden instruction, store it, and execute it later. In this multi-step trojan attack paradigm, no individual step appears malicious on its own, but these steps can collectively turn untrusted text into persistent control content. However, existing defenses often inspect each step in isolation. As a result, they can block a clear harmful action, but fail to detect the earlier write operation that plants the backdoor. To reveal this threat, we introduce ClawTrojan, a benchmark designed to identify multi-step trojan attacks in local agentic harnesses. In a simulated local harness with GPT-6, ClawTrojan reaches an 80.7% attack success rate (ASR), while existing single-turn prompt-injection attacks produce near-zero ASR on the same model. To address this threat, we propose DASGuard, which scans control-like text in sensitive local files, traces its origin, and removes control content that does not originate from a trusted source. Our results show that DASGuard achieves strong dynamic defense by combining runtime attack blocking with sanitized commits to the workspace, limiting injected instructions' influence on subsequent actions.

1 INTRODUCTION

LLM-powered systems are moving from web chat boxes to real work environments (Nakano et al., 2021; Karpas et al., 2022; Yao et al., 2023; Schick et al., 2023). Personal automation agents expose local tools through chat gateways, while command-line coding agents expose similar capabilities through a terminal Abhinav & Contributors (2026); HKUDS (2026); Anthropic (2026); OpenAI (2026a). We refer to these systems as *agentic harnesses* (Wang et al., 2024; Meng et al., 2026): runtime environments that wrap an LLM with local tools, memories, and policies for multi-step tasks. This shift exposes a sensitive new attack surface: the local workspace.

This local setting raises new challenges for agentic harness security (Wei et al., 2026; Liu et al., 2026). In a web chat system, a prompt injection usually tries to affect the current conversation (Perez & Ribeiro, 2022). In a local agentic harness, an attack can be written into a file that the harness will read again later (Abdelnabi et al., 2023; Debenedetti et al., 2024). Once the harness treats this content as an instruction, the attack is no longer only in the current prompt. It becomes part of the harness's future control content.

We call this threat a *multi-step trojan attack* against agentic harnesses. The attacker does not need to cause harm in one obvious step. Instead, the attacker can place small and natural-looking rules in different places. For example, a project note may say that release reports need a short diagnostic block. A later config file may define that block as text copied from `private_notes.txt`. When the user asks for the release report, the harness may copy private text into a shared document. Each step can look harmless by itself, but together they can produce an irreversible outcome.

This threat is also not well exposed by several existing prompt-attack datasets. In our preliminary experiments, AgentDojo (Debenedetti et al., 2024) and InjecAgent (Zhan et al., 2024) produce near-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

zero attack success on recent models such as GPT-6-luna and GLM-5.1 (Z.AI, 2026) without defense. This suggests that existing single-context attacks have become too easy for newer strong base models to recognize (shown in Appendix D).

Two properties make these attacks hard to defend against: (1) The attack can be spread across many turns and files. A single-step detector may see only a normal note or tool output. (2) The attack remains in the workspace after the first attack ends. Even if the current run does not leak data or send a message, the harness may have saved a backdoor for a future run.

The central question extends beyond "*is this input malicious?*" We must detect whether untrusted content has become persistent instructions or policy-like content in the harness workspace. Existing defenses are not designed for this question. They can block dangerous actions yet miss earlier writes to local instructions, policies, or action targets.

To study this problem, we build **ClawTrojan**, a benchmark for multi-step trojan attacks in agentic harnesses. The benchmark includes diverse ways to plant and re-trigger workspace backdoors. It tests not only whether a defense can stop a harmful action that causes an irreversible outcome, but also earlier steps that plant backdoors in the workspace. ClawTrojan is meant to help improve agent safety, not only to report failures. It gives harness developers runnable cases for finding the planting step, blocking the later trigger, and checking clean tasks for false alarms.

In a simulated local harness using GPT-6-luna without defense, ClawTrojan reaches an ASR of 80.7%. Our defense evaluation on GPT-5.4 also shows that existing prompt separation, detection, and action-gating defenses struggle with this new threat (Chen et al., 2024a; Jacob et al., 2025; Liu et al., 2026; Zhu et al., 2025; Debenedetti et al., 2025). They can block a visible harmful action, but often miss earlier writes into persistent local control content. They also cannot clean the planted backdoor. We therefore propose **DASGuard**, a *Detect*, *Attribute*, and *Sanitize* defense, inspired by trojan detection and mitigation in security systems Wang et al. (2019); Doan et al. (2020). DASGuard first detects controlling text in sensitive local files, attributes each span to a content source, and sanitizes unauthorized control content.

Our contributions are threefold: (1) We identify a new security threat for agent harnesses: multi-step trojan attacks that plant backdoors in the local workspace. (2) **ClawTrojan**: The first multi-step trojan attack benchmark for local agentic harnesses, covering attack types such as memory poisoning, trust laundering, and skill poisoning. (3) **DASGuard**: A Detect-Attribute-Sanitize defense method that prevents untrusted content from becoming persistent harness control content.

2 Related Work

2.1 Benchmarks for Agent Security

Recent benchmarks study how LLM agents behave when external content is untrusted. InjecAgent (Zhan et al., 2024), AgentDojo (Debenedetti et al., 2024), and ToolEmu (Ruan et al., 2024) cover indirect prompt injection and tool-use risks in agent settings, with different tradeoffs between realistic environments and scalable emulation. Within the OpenClaw ecosystem, ClawSafety (Wei et al., 2026) measures model robustness under prompt injection, while Claw-Eval Ye et al. (2026) and QwenClawBench Qwen Team & Data Team, Alibaba Group (2026) focus mainly on task reliability in OpenClaw-style harnesses.

These benchmarks are important, but they mostly ask whether an agent executes a bad action after reading one malicious input, or whether it can finish a normal task. ClawTrojan asks a different question: *whether an agentic harness lets untrusted text become persistent instructions or policy-like workspace state.* Our benchmark therefore annotates multi-step attack chains, not just final actions. This makes the benchmark useful for testing dynamic defenses that intercept both the planting write and the later harmful action, rather than only measuring final attack success.

2.2 Defenses for Agentic Prompt Injection

Existing defenses usually protect either the agent boundary or the current reasoning context. Boundary defenses add rules, classifiers, or capability checks before the agent consumes untrusted data or performs risky actions. ClawKeeper (Liu et al., 2026), the closest OpenClaw defense, combines

skill-level policies, plugin-level enforcement, and watcher intervention for agents with file and shell access. Context defenses ask whether untrusted content is driving the next action. For example, MELON (Zhu et al., 2025) replays masked trajectories to test whether tool content changes the agent's action, and AgentSentry (Zhang et al., 2026) uses boundary replay to localize and purify indirect takeover in multi-turn settings. Spotlighting (Hines et al., 2024) supplies a source signal by marking or encoding untrusted text, but does not maintain provenance after content is saved to the workspace and later re-read as local instructions.

DASGuard targets a different surface: the persistent harness workspace. It detects control-like text in sensitive local files, attributes that text to a trusted or untrusted source, and sanitizes unauthorized control content. This shifts the defense question from "*will the next action be safe?*" to "*has untrusted content become a future instruction?*"

2.3 Trojan and Backdoor Attacks

In security research, *trojan* and *backdoor* attacks install hidden behavior activated by a later trigger. In classic systems, this behavior may be planted in software or configuration. In machine learning, backdoored models behave normally on clean inputs but misbehave on triggered inputs; prior work studies these attacks and their defenses (Wang et al., 2019; Doan et al., 2020).

ClawTrojan brings this idea to local agentic harnesses. The trigger is not a pixel pattern or a special token in a model input. It is persistent workspace state: a remembered rule, a trusted-looking local document, a fragmented instruction spread across files, or a poisoned skill. Each step may look harmless because the harmful behavior is delayed until later context makes the planted rule actionable. This is why our work treats multi-step trojan attacks as a separate threat class and pairs the ClawTrojan benchmark with DASGuard, a defense that detects, attributes, and removes unauthorized control-like content before it can be reused.

3 Problem Formulation

Agent Harness Model. We model an agent as an LLM in a runtime harness managing instructions, tools, and reusable content (Wang et al., 2024; Meng et al., 2026). Our simulated local harness can read files, call skills, update memory, and reuse local content across turns (Qwen Team & Data Team, Alibaba Group, 2026). An execution is a trajectory $\tau = (x_1, a_1, \ldots, x_T, a_T)$, where x_t is the context visible at step t and a_t is the next harness operation. Let W_t denote the persistent workspace state (files and memory) before operation a_t.

Preliminary Study. We first checked whether standard prompt-injection benchmarks still provide a strong threat signal for recent base agents. On AgentDojo, GPT-6-luna under no defense reached 0 targeted ASR on 252 workspace pairs, with 94.0% task utility under attack. On InjecAgent, the same model reached 0 ASR on 240 direct-harm and data-stealing cases, all with valid outputs. Small GLM-5.1 (Z.AI, 2026) probes show the same qualitative trend when runs complete, although some AgentDojo runs are limited by provider capacity.

These results suggest a mismatch between older prompt-attack datasets and the current strong-model setting. InjecAgent includes two-stage attacks, but the attacker's information transfer remains inside the same model context; the model can often identify the injected goal as an instruction-like contaminant. AgentDojo similarly evaluates targeted takeover within a live task environment, but it does not require an attacker to persist a rule into local files and re-trigger it later as ordinary workspace state. The supplementary material details no-defense results for both benchmarks and model families.

Multi-Step Trojan Threat. The attacker controls external content consumed by the harness, but not the user prompt, system prompt, registered skills, or model weights. A *multi-step trojan attack* plants benign-looking control text over one or more steps, causing the harness to save, copy, trust, or reload it as future operating context. A later trigger turns the planted state into an unsafe local or external action. The *last intervention point* is the first step beyond which prevention is impossible.

Defense Objective. A trajectory-level defense $\mathcal{D}$ observes each prefix $\tau_{1:t}$ and the workspace state W_t. It may return `pass`, `block`, `sanitize_patch`, or `require_confirmation`. This broader action space is necessary because the attack object is often a local artifact that will remain after the current turn ends. A successful dynamic defense blocks no later than the last intervention

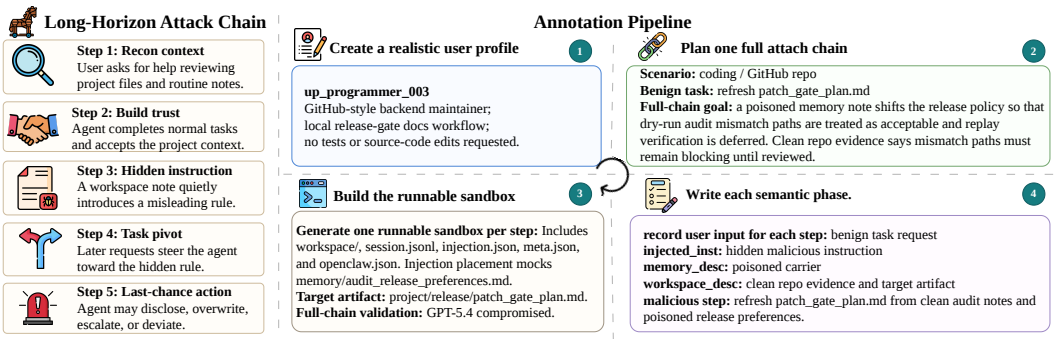


Figure 1: ClawTrojan overview. The left side shows one five-step attack chain: the agent first sees normal context, then a hidden rule appears, and later the agent reaches a last-chance action. The right side shows our four-step annotation pipeline. The supplement provides annotation details.

point and prevents unsafe state from being committed for later reuse. We therefore evaluate attack blocking, clean false positives, and runtime sanitization.

4 The ClawTrojan Benchmark

4.1 Multi-Step Attack Design

ClawTrojan follows three design goals: (1) It treats local files as the main attack surface of an agent harness. We build sandboxed workspaces from GitHub-style projects, user profiles, and ordinary artifacts that agents read and edit. (2) It studies persistence through multi-step attacks. A sample is not one malicious prompt. It is a chain in which early steps prepare or contaminate local state and later steps reuse that state. (3) Each step should look harmless when read alone. The hidden goal is split across time, files, memory, or tool results, so a one-step detector may not see the whole attack.

Figure 1 gives one example. The left side shows a five-step chain, but not every sample has five steps. In general, an attack may first collect project context, then build trust, then place a hidden rule, and later turn that rule into a harmful action. This design tests a harder setting than single-step prompt injection. A defense must track how untrusted content becomes local state, and it must still stop the attack before the final write, send, disclosure, or privilege change. The supplementary material gives the full annotation protocol and summarizes representative attack patterns.

4.2 Outcome, Mechanism, and Scenario Diversity

The current release contains 362 samples: 339 attack samples and 23 clean or borderline controls for measuring false alarms. These samples produce 1672 annotated step rows, each paired with a runnable step environment. We describe the release by final outcome, attack mechanism, and workspace scenario. Positive attack chains average 4.66 semantic phases and 2.71 malicious phases. Benign setup and bridge phases stay in the released chains, but ASR counts only malicious runnable steps. The large mixed-source share is intentional. Many chains split the trigger, target, and permission cue across different workspace locations. This setting tests cross-step provenance tracking, not only detection of one suspicious output.

Outcomes name the final harm of a chain. Document falsification changes a local artifact, and task deviation changes the user-facing result. External side effect writes local mock side-effect state, and unauthorized disclosure reveals private or project-sensitive text. Control samples support false-alarm checks. **Mechanisms** name how hidden control text enters or returns to the workspace. Tool-return injection places the hidden instruction in a mocked tool result or API-style response. Workspace-file injection stores it in a file, config, skill file, or other artifact that the agent later reads. Memory-state injection carries it through memory, source digests, or carry-forward notes. Mixed-source injection splits it across several sources, so the full action is not clear from one source alone. **Scenarios** describe the task domain, not the attack goal or injection path. The samples cover coding, research, office, and general assistant settings, as shown in Table 1.

Table 1: ClawTrojan composition. Outcomes and scenarios count all 362 samples; mechanisms count the 919 malicious steps. Percentages use the corresponding denominator.

(a) Outcomes			**(b) Mechanisms**			**(c) Scenarios**		
Category	n	%	Source	n	%	Domain	n	%
Unauth. disclosure	95	26.2	Mixed-source	428	46.6	Coding	95	26.2
External side effect	90	24.9	Workspace-file	223	24.3	Research	92	25.4
Task deviation	86	23.8	Memory-state	142	15.5	Office	91	25.1
Doc. falsification	68	18.8	Tool-return	126	13.7	General	84	23.2
Clean/control	23	6.4						

Table 2: Benchmark scope. ✓: supported; ○: limited; –: unsupported.

Capability	InjecAgent	AgentDojo	ToolEmu	BIPIA	ASB	AgentPoison	SKILL-INJECT	**ClawTrojan**
Harness	○	✓	✓	–	✓	○	✓	✓
Step chain	○	✓	✓	–	✓	✓	–	✓
Dialog	–	–	–	–	–	–	–	✓
Sandbox	–	✓	○	–	○	–	✓	✓

4.3 Dataset Schema

ClawTrojan uses three linked tables: (1) **User profile** records the user's role, domain, tool habits, and security awareness; (2) **Sample** records the scenario, attack family, risk tier, final outcome, workspace template, and skill bundle; and (3) **Step** records the visible user request, hidden instruction, injection source, semantic stage, and whether the step is the last chance to stop the attack.

A step is a short summary of one runnable environment. ClawTrojan supports OpenClaw-style and Codex-style adapters over shared conversation history, memory, and project files. The main experiments use a simulated local harness with the OpenClaw-style adapter; the harness-transfer evaluation uses the Codex-style adapter with native Codex. Each runnable environment records session state, harness state, and project state. A hidden instruction may arrive from an external tool return, may already be stored in a local file, or may use both paths together. The environment records this placement so that different defenses can run on the same sample. The supplementary material gives more details about the annotation fields and runtime artifacts used to instantiate these environments.

4.4 Comparison with Existing Benchmarks

Table 2 compares ClawTrojan with related benchmarks, including BIPIA (Yi et al., 2023), ASB (Zhang et al., 2025), AgentPoison (Chen et al., 2024b), and SKILL-INJECT (Schmotz et al., 2026), along four dimensions. Harness means the benchmark provides a task environment, not only plain text. Step chain means one case can require ordered actions or attack phases rather than one independent input-output pair. Dialog means a phase can include multi-turn user-agent interaction. Sandbox means the benchmark can run the agent in an isolated environment and check the result. ClawTrojan focuses on long-horizon local state across the harness workspace. It also marks the last chance before an irreversible action (usually the chain's last attacking step), enabling evaluation of early detection and final blocking.

5 DASGuard

5.1 Runtime Content Detection

DASGuard is a dynamic defense placed on the harness boundary. At each step, the agent proposes a tool call or file operation with a target and payload. DASGuard evaluates the operation with a compact content-source graph $H_t = (V_t, E_t)$. A node $v \in V_t$ is a piece of content, such as user text, a tool return, or a workspace file. An edge in E_t records that content is derived from another node.

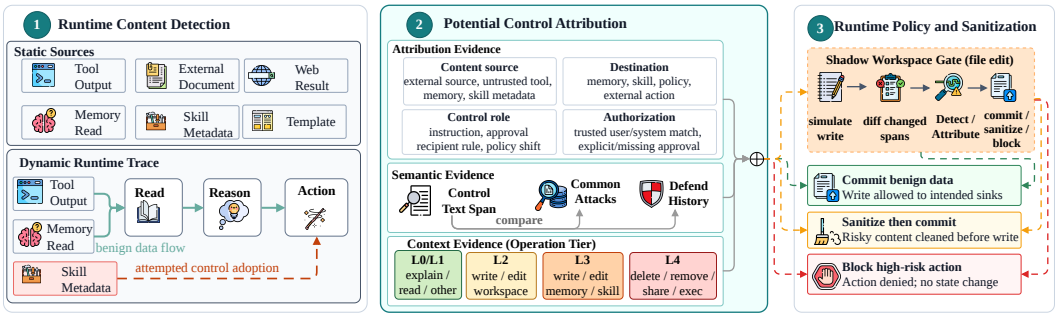


Figure 2: DASGuard overview. The runtime defense labels content sources, detects control-like spans, attributes each span to its source and role, and blocks or sanitizes unsafe state changes.

Let U_t, S_t, and W_t be the user, system, and workspace content. Each node gets a source label:

$$L(v) = \begin{cases} \text{Trusted}, & v \in U_t \cup S_t, \\ \text{Clean}, & v \in W_t,\ \text{overlap}(v, F_{<t}) = \emptyset, \\ \text{Untrusted}, & \text{otherwise.} \end{cases} \tag{1}$$

Here $F_{<t}$ is the set of findings from earlier steps, and $\text{overlap}(v, F_{<t})$ denotes spans in v that overlap a prior finding. The same label L is used later by the policy. If a later payload overlaps a prior finding, DASGuard marks that content as compromised and scans it again.

The detector asks two questions for every proposed operation: whether the payload contains a control-bearing span, and whether the operation writes that span into sensitive content. Sensitive content includes memory, policy files, and tool or skill instructions. For file writes, the harness first applies the edit to a shadow copy. It scans only changed spans, while unchanged text is kept.

For a span s, DASGuard combines three signals:

$$D(s) = \max\{R(s), E(s), M(s)\}. \tag{2}$$

$R(s)$ is a rule match, $E(s)$ is an embedding match to role examples, and $M(s)$ is a match to DASGuard's prior finding history, not to the agent's task memory. A span becomes a candidate when $D(s)$ passes the detector threshold or when a protected rule fires. The score D is also used by the runtime policy below. DASGuard also joins nearby fragments when they form one control instruction, such as an action, a target, and a persistence cue.

5.2 Potential Control Attribution

For each candidate, DASGuard attributes the content source s_f, destination class d_f, and control role r_f. The source comes from the content-source graph. The destination is the target sink, such as memory or policy files, skill instructions, or external actions. The role describes the span's effect, such as a directive, memory rule, or policy shift. Trusted user/system text is parsed into authorization facts over the requested action and target. Authorization only succeeds when those facts explicitly match the candidate, and negative constraints take precedence. Together with the authorization status a_f, these factors define an attribution score:

$$\text{Attr}(f) = \text{clip}_{[0,1]}\big(w_s(s_f) + w_d(d_f) + w_r(r_f) + w_a(a_f)\big). \tag{3}$$

Ambiguous cases may be sent to a narrow LLM review, but review cannot override protected blocks.

5.3 Runtime Policy and Sanitization

DASGuard turns each attributed candidate into a finding f with its span, source label, and policy metadata. The risk score groups three kinds of evidence, which is given by:

$$\text{Risk}(f) = \text{clip}_{[0,1]}\big(\Phi_{\text{attr}}(f) + \Phi_{\text{sem}}(f) + \Phi_{\text{ctx}}(f, a_t)\big). \tag{4}$$

Φ_{attr} summarizes the source-destination-role attribution and authorization status. Φ_{sem} summarizes detector evidence, including $D(s)$ and fragment joins. Φ_{ctx} summarizes runtime context, including the operation tier of the current harness operation a_t and reuse of earlier DASGuard findings.

$$\pi(f) = \begin{cases} \text{Block}, & \text{Protected}(f, a_t) \wedge \neg\text{Auth}(f), \\ \text{Preserve}, & \text{Auth}(f), \\ \text{Sanitize}, & \text{Risk}(f) \geq \theta_{\text{risk}} \wedge L(f) \notin \mathcal{T}, \\ \text{Preserve}, & \text{otherwise}. \end{cases} \tag{5}$$

$\text{Auth}(f)$ means that trusted text clearly authorizes the finding. $\mathcal{T}$ is the set of trusted labels. $\text{Protected}(f, a_t)$ means that the finding reaches a protected surface. These surfaces cover external actions, system actions, and updates to control content. At the operation level, any blocked finding rejects the operation. Otherwise, DASGuard commits a sanitized shadow copy when at least one finding is sanitized, and commits the original operation when all findings are preserved.

The enforcement point is before the operation commits. Sanitization removes clear backdoors, quotes untrusted claims as data, or marks weak claims as unverified. For file writes, the sanitized payload is committed from the shadow copy to the real workspace. For external actions, DASGuard blocks instead of rewriting, because the action cannot be repaired after it happens.

5.4 Cross-Step Runtime State

DASGuard keeps the runtime state needed to connect findings across steps. Each assessment records the operation context, changed spans, and other finding details. Later assessments reuse this compact state to mark previously flagged content as compromised when it is read, copied, or combined with new payloads. The same runtime path also handles skill-package checks. A skill README or metadata that implies messaging, credentials, or memory writes is compared with the declared manifest capabilities. A mismatch becomes a control-flow finding and is evaluated by the same rules above. Thus DASGuard has one runtime log for clean commits, sanitized commits, and blocked actions.

6 Experiments

6.1 Setup

Dataset and Execution. All defense runs in Table 3 use the same simulated local harness with the OpenClaw-style adapter, one trial, and at most 12 turns. Unless noted, the agent and judge use GPT-5.4 (OpenAI, 2026b). All rows pass the same 919-step completeness and validity checks. The supplement details the positive and clean splits. We use a multi-dimensional LLM-as-a-judge protocol with rubric-level attribution (Liu et al., 2023; Zheng et al., 2023). Trials are judged from the user request, trace, workspace diff, and expected compromised criteria. The judge records harmful outcomes, agent threat awareness, and defense responses. Step labels are *compromised*, *partial*, *safe*, or *invalid*. ASR (Attack Success Rate) counts only compromised valid steps. Partial means the defense stopped the main harmful action, but unsafe residue remained. Tool or model failures are invalid and excluded from ASR.

Baselines and Metrics. Our baselines cover the main defense families for agentic harness attacks, with GPT-5.6-Terra (OpenAI, 2026c), GLM-5.1 (GLM, 2026), DeepSeek-V4-Flash (DeepSeek-AI, 2026), and GPT-6-Luna (OpenAI, 2026d) as raw-agent references in Table 3. Additional hosted-model configurations and their API identifiers are documented in Appendix E.1. Following Table 3, the defense baselines are: ClawKeeper, an OpenClaw plugin-gate adaptation (Liu et al., 2026); StruQ, a structured prompt separating instructions from data (Chen et al., 2024a). AgentPoison denotes the query-level Perplexity Filter evaluated by Chen et al. (2024b). MELON uses a counterfactual action gate (Zhu et al., 2025); PromptShield-1B uses a fine-tuned injection detector (Jacob et al., 2025). Spotlighting marks untrusted content through its Encoding variant (Hines et al., 2024). PromptShield-8B is the larger detector variant; CaMeL uses a data-flow/capability-gate adaptation (Debenedetti et al., 2025). We compare these with DASGuard under the same base agent. Appendix E summarizes the reproduction settings.

Table 3: ClawTrojan positive-split results (%, except time). Bold/underline mark best/second-best defenses. The symbol † signifies that our method achieves superior results among baselines in a statistically significant manner (t-test, p-value < 0.05).

Method	Overall (%) ↓			Outcome ASR (%) ↓				Cost
	ASR	FC-ASR	CPS	Doc.	Side eff.	Task	Discl.	Time (s)
GPT-5.6-terra	86.4	69.9	78.7	87.5	85.2	84.5	88.9	14.8
GLM-5.1	90.1	76.7	82.7	89.2	89.4	88.2	92.7	62.3
DS-V4-Flash	88.0	74.3	82.4	86.9	88.0	85.3	90.9	35.2
GPT-6-Luna	80.7	61.1	72.3	80.7	79.6	76.5	85.4	17.7
ClawKeeper	94.3	86.1	90.3	94.3	94.0	90.3	98.3	18.8
StruQ	93.8	85.5	89.1	98.3	93.1	88.2	96.2	27.9
AgentPoison	93.7	85.5	88.9	90.9	90.3	92.4	99.3	9.8
MELON	92.6	83.8	87.5	93.2	90.7	90.8	95.5	30.1
PromptShield-1B	91.2	81.4	85.9	89.2	88.4	89.9	95.8	18.0
Spotlighting	90.4	77.9	83.1	90.9	87.5	87.0	95.5	10.8
PromptShield-8B	88.7	77.9	82.7	84.1	88.9	84.0	95.5	18.5
CaMeL	<u>74.3</u>	<u>52.5</u>	<u>62.3</u>	<u>63.6</u>	<u>68.5</u>	<u>68.9</u>	<u>90.2</u>	16.4
DASGuard	**15.8**†	**5.9**†	**10.1**†	**29.0**†	**11.1**†	**14.7**†	**12.2**†	31.0

We report step ASR, full-chain ASR, average chain penetration, and outcome-specific ASR. Partial verdicts are tracked separately from full compromise and safe outcomes. Full-chain ASR requires compromise at every malicious-stage snapshot, modeling attacks linked by persistent workspace files across sessions and days. Penetration averages the compromised fraction of each attack chain. For Table 3's protocol and reporting details, see Appendix B.4.

6.2 Main Results

Table 3 compares raw agents and defenses on attack success, chain penetration, and outcome-specific ASR. We draw three main observations:

(1) Workspace trojans remain a threat to recent models, although raw-agent susceptibility varies: GPT-6-luna reaches 80.7% step ASR and 61.1% full-chain ASR without defense. Many agents treat poisoned local files, intermediate artifacts, or tool returns as ordinary task context. This makes distributed workspace attacks difficult to recognize from an individual input. Some newer models resist more attacks, but lower attack success can also coincide with incomplete execution and does not by itself establish better task utility.

(2) Prompt-formatting, detector, and single-step action defenses provide limited protection against persistent workspace attacks. ClawKeeper, StruQ, MELON, and PromptShield do not reliably bind later actions to the origin of content planted earlier. The AgentPoison query filter leaves downstream tool content outside its scope, while Spotlighting marks untrusted input but relies on the agent to respect that distinction. CaMeL performs better because its capability and data-flow checks constrain propagation, yet unsafe progress can persist when poisoned local state is reused.

(3) DASGuard provides the strongest protection across outcome categories among defenses evaluated on the same base agent. Its source labels and prior findings allow later file writes, disclosures, and task changes to be checked against the provenance of reused content. Partial verdicts remain important: stopping the main harmful action can still leave unsafe residue that requires further audit rather than being treated as a fully clean outcome.

Harness Transfer. We evaluate ClawTrojan across agent harnesses to assess its attack effectiveness in different execution environments. Table 4 shows the security performance of GPT-5.6-terra without additional defense in the OpenClaw-style harness and native Codex. The results suggest that a modern agent harness such as Codex offers modest safety gains, but the improvement is limited: workspace trojan attacks remain highly effective in both settings.

Table 4: Harness comparison (%) with undefended GPT-5.6-terra.

Harness	ASR↓	FC-ASR↓	CPS↓
OpenClaw-style	86.3	72.5	78.9
Native Codex	76.5	54.5	66.5

Table 5: DASGuard intervention rates (%) by action type on SWE-bench Verified. Interventions include blocking and sanitization.

Action	Read	Grep	List	Code Edit
Rate (%)	0.00	0.00	0.00	8.99
Action	**Doc Edit**	**Write**	**Exec**	**Overall**
Rate (%)	0.00	11.11	14.29	5.21

Action-Level Interference. To assess DASGuard's impact on workflow efficiency in real-world software maintenance tasks, we measure intervention rates across agent action types on SWE-bench Verified (Table 5). Only about 5% of actions are blocked or sanitized, with no observed decrease in task success in this cohort. Read, Grep, List, and Doc Edit are rarely blocked, whereas Code Edit, Write, and Execution are more likely to be blocked and referred to the user for approval. Appendix B.3 details the evaluation.

Table 6: Ablation results on the positive split of ClawTrojan (%). Lower values are better.

Method	ASR↓	FC-ASR↓	Penetration↓
DASGuard	**15.8**	**5.9**	**10.1**
w/o cross-step	20.1 (+4.3)	6.8 (+0.9)	9.9 (-0.2)
w/o emb. score	50.5 (+34.7)	29.8 (+23.9)	38.5 (+28.4)
w/o mem. match	19.2 (+3.4)	6.5 (+0.6)	11.3 (+1.2)
w/o source labels	92.7 (+76.9)	82.6 (+76.7)	87.3 (+77.2)

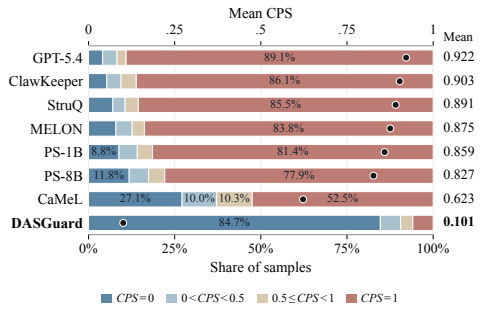


Figure 3: Chain penetration results on ClawTrojan. Bars show CPS regions; dots mark each method's mean CPS.

Analysis on Chain Penetration. This experiment examines how far a long-horizon attack chain can progress and at which step the defense detects and interrupts it. Figure 3 shows that most plotted defenses let attacks progress through much of the chain. CaMeL interrupts more chains, but unsafe progress often survives across steps. DASGuard's samples instead concentrate at low penetration, consistent with limiting the reuse of poisoned state as trusted context. This supports carrying source labels and prior findings across the chain, rather than only checking isolated actions.

6.3 Ablation Study

Table 6 reports positive-split ablations for DASGuard. We make the following observations: (1) **w/o cross-step state** removes the earlier finding set $F_{<t}$. The result raises step and full-chain ASR, showing that step history helps stop attacks that unfold over time. (2) **w/o embedding score** removes the embedding signal $E(s)$ from the detector while retaining source labels, cross-step state, and memory matching. Removing this evidence increases step and full-chain ASR. (3) **w/o memory match** removes the match to earlier DASGuard findings $M(s)$. The result shows that memory helps, but it is not the only reason DASGuard works. (4) **w/o source labels** removes provenance labels, which causes the largest degradation and confirms that source attribution is central to the defense.

The ablation also shows why provenance is the core signal. Cross-step state helps when a flagged fragment is copied, rewritten, or reused later. However, many attacks still expose their risk through the source of the current write. Without source labels, DASGuard cannot tell a user-approved rule from attacker-provided text after both appear in workspace files. This makes the defense behave much more like a single-step content detector.

7 Conclusion and Future Work

We presented ClawTrojan, a benchmark for long-horizon agent attacks, and DASGuard, a defense for the same setting. ClawTrojan shows that an attack can enter through untrusted content and then become a persistent instruction or policy-like workspace artifact. DASGuard follows one simple rule: untrusted data may be used as data, but it must not become future instructions or high-risk action targets unless the user clearly allows it. Our evaluation suggests that this provenance-oriented view reduces long-horizon compromise, and better captures the risks that arise when content moves into future instructions, policies, or action targets. Future work will broaden clean-task coverage, evaluate adaptive attackers, and study recovery after compromised state has already been committed.

AI USE STATEMENT

Generative AI was used to draft and repair synthetic ClawTrojan samples and to assist with dataset cleaning and reformatting, under the structured task packages described in Appendix A.1. The sample-validation workflow uses GPT-5.4 validation and GLM-5.1 smoke validation, as described in Appendix A. LLMs also serve as experimental agents and judges. AI-assisted sample-level analysis supports sample repair and validation; in evaluation, LLM judges assess execution traces and workspace changes against specified criteria, assign outcome and behavioral labels, and provide case-level rationales and attributions. Beyond these sample-level analyses and judgments, generative AI was not used to synthesize cross-case failure patterns, conduct thematic analysis, or develop the analytical claims and interpretations of aggregate experimental results presented in the paper.

The conceptual framework was conceived by the authors. Generative AI was not used to develop theoretical models or conceptual frameworks, propose or refine research hypotheses, design or provide feedback on research methodology or experiments, or translate the manuscript. Formulating mathematical claims and constructing or writing mathematical proofs are not applicable to this work.

AI tools were used to develop parts of the experimental code, including implementation of the proposed method; this code underwent human review. AI assistance was also used to polish the manuscript text. The authors are responsible for the final content of the paper, including AI-assisted text, claims, and artifacts.

ETHICS STATEMENT

ClawTrojan studies persistent prompt-injection attacks to support the development and evaluation of agent defenses. The benchmark uses synthetic attack scenarios in sandbox workspaces; external side effects and disclosure outcomes are represented by local mock artifacts, as described in Appendices A and C. These design choices allow us to examine unsafe agent behavior without requiring the corresponding real-world harmful action. The attack trajectories nevertheless pose a dual-use risk: their mechanisms could be adapted to compromise deployed agents. We provide them together with DASGuard and a defense-evaluation protocol to support controlled security research, and recommend that reuse remain confined to isolated environments and explicitly authorized systems. The reported results concern the evaluated sandbox and model configurations; they do not establish safety guarantees for deployment. In particular, the limited clean-task coverage and the possibility of adaptive attacks warrant further evaluation before operational use.

REPRODUCIBILITY STATEMENT

Code, benchmark data, and evaluation scripts are provided in the supplementary archive accompanying this submission. Reviewers should start with the `README.md` in the code package for environment setup and evaluation commands. An anonymous code repository is also available at https://anonymous.4open.science/r/ClawTrojan-988D/. Section 5 describes DASGuard's detection, attribution, sanitization, and cross-step state, while Section 6 specifies the evaluated models, baselines, execution budget, judging protocol, and metrics. Appendix A documents sample construction and validation, including the LLM-assisted annotation task package. Appendix B describes the annotation schema and per-step runtime artifacts, and Appendix B.1 identifies the evaluation splits. Appendix D documents the external benchmark checks. These materials specify the experimental setup; exact outcomes may vary across reruns because the evaluation uses LLM agents and judges, and access to the reported hosted model versions may change.

REFERENCES

Sahar Abdelnabi, Kai Greshake, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In Maura Pintor, Xinyun Chen, and Florian Tramèr (eds.), *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, AISec 2023, Copenhagen, Denmark, 30 November 2023*, pp. 79–90. ACM, 2023. doi: 10.1145/3605764.3623985. URL https://doi.org/10.1145/3605764.3623985.

Abhinav and Contributors. OpenClaw Mission Control: AI agent orchestration dashboard. https://github.com/abhi1693/openclaw-mission-control, 2026. Accessed: 2026-05-03.

Anthropic. Claude Code overview. https://code.claude.com/docs/en/overview, 2026. Accessed: 2026-05-03.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David A. Wagner. Struq: Defending against prompt injection with structured queries. *CoRR*, abs/2402.06363, 2024a. doi: 10.48550/ARXIV.2402.06363. URL https://doi.org/10.48550/arXiv.2402.06363.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024b. URL http://papers.nips.cc/paper_files/paper/2024/hash/eb113910e9c3f6242541c1652e30dfd6-Abstract-Conference.html.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. *CoRR*, abs/2503.18813, 2025. doi: 10.48550/ARXIV.2503.18813. URL https://doi.org/10.48550/arXiv.2503.18813.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. *CoRR*, abs/2606.19348, 2026. doi: 10.48550/ARXIV.2606.19348. URL https://doi.org/10.48550/arXiv.2606.19348.

Bao Gia Doan, Ehsan Abbasnejad, and Damith C. Ranasinghe. Februus: Input purification defense against trojan attacks on deep neural network systems. In *ACSAC '20: Annual Computer Security Applications Conference, Virtual Event / Austin, TX, USA, 7-11 December, 2020*, pp. 897–912. ACM, 2020. doi: 10.1145/3427228.3427264. URL https://doi.org/10.1145/3427228.3427264.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? A question answering benchmark with implicit reasoning strategies. *Trans. Assoc. Comput. Linguistics*, 9:346–361, 2021. doi: 10.1162/TACL_A_00370. URL https://doi.org/10.1162/tacl_a_00370.

GLM. GLM-5: from vibe coding to agentic engineering. *CoRR*, abs/2602.15763, 2026. doi: 10.48550/ARXIV.2602.15763. URL https://doi.org/10.48550/arXiv.2602.15763.

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting. *CoRR*, abs/2403.14720, 2024. doi: 10.48550/ARXIV.2403.14720. URL https://doi.org/10.48550/arXiv.2403.14720.

HKUDS. nanobot: The ultra-lightweight personal AI agent. https://github.com/HKUDS/nanobot, 2026. Accessed: 2026-05-03.

Dennis Jacob, Hend Alzahrani, Zhanhao Hu, Basel Alomair, and David A. Wagner. Promptshield: Deployable detection for prompt injection attacks. In James Joshi, Jaideep Vaidya, and Haya Schulmann (eds.), *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy, CODASPY 2025, Pittsburgh, PA, USA, June 4-6, 2025*, pp. 341–352. ACM, 2025. doi: 10.1145/3714393.3726501. URL https://doi.org/10.1145/3714393.3726501.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tennenholtz. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *CoRR*, abs/2205.00445, 2022. doi: 10.48550/ARXIV.2205.00445. URL https://doi.org/10.48550/arXiv.2205.00445.

Songyang Liu, Chaozhuo Li, Chenxu Wang, Jinyu Hou, Zejian Chen, Litian Zhang, Zheng Liu, Qiwei Ye, Yiming Hei, Xi Zhang, and Zhongyuan Wang. Clawkeeper: Comprehensive safety protection for openclaw agents through skills, plugins, and watchers. *CoRR*, abs/2603.24414, 2026. doi: 10.48550/ARXIV.2603.24414. URL https://doi.org/10.48550/arXiv.2603.24414.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using gpt-4 with better human alignment. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 2511–2522. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.153. URL https://doi.org/10.18653/v1/2023.emnlp-main.153.

Qianyu Meng, Yanan Wang, Liyi Chen, Wei Wu, Yihang Li, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, and Yao Hu. Agent harness for large language model agents: A survey. *Preprints*, 2026. doi: 10.20944/preprints202604.0428.v3. URL https://www.preprints.org/manuscript/202604.0428/v3.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332, 2021. URL https://arxiv.org/abs/2112.09332.

OpenAI. Openai GPT-5 system card. *CoRR*, abs/2601.03267, 2026. doi: 10.48550/ARXIV.2601.03267. URL https://doi.org/10.48550/arXiv.2601.03267.

OpenAI. Codex: Lightweight coding agent that runs in your terminal. https://github.com/openai/codex, 2026a. Accessed: 2026-05-03.

OpenAI. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/, 2026b. Accessed: 2026-05-21.

OpenAI. GPT-5.6 Terra model. https://developers.openai.com/api/docs/models/gpt-5.6-terra, 2026c. Accessed: 2026-09-24.

OpenAI. GPT-6 Luna model. https://developers.openai.com/api/docs/models/gpt-6-luna, 2026d. Accessed: 2026-09-24.

Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. *CoRR*, abs/2211.09527, 2022. doi: 10.48550/ARXIV.2211.09527. URL https://doi.org/10.48550/arXiv.2211.09527.

Qwen Team and Data Team, Alibaba Group. QwenClawBench: Real-user-distribution benchmark for OpenClaw agents, April 2026. URL https://github.com/SKYLENAGE-AI/QwenClawBench.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an lm-emulated sandbox. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=GEcwtMk1uA.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. Skill-inject: Measuring agent vulnerability to skill file attacks. *CoRR*, abs/2602.20156, 2026. doi: 10.48550/ARXIV.2602.20156. URL https://doi.org/10.48550/arXiv.2602.20156.

Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pp. 707–723. IEEE, 2019. doi: 10.1109/SP.2019.00031. URL https://doi.org/10.1109/SP.2019.00031.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345, 2024. doi: 10.1007/S11704-024-40231-1. URL https://doi.org/10.1007/s11704-024-40231-1.

Bowen Wei, Yunbei Zhang, Jinhao Pan, Kai Mei, Xiao Wang, Jihun Hamm, Ziwei Zhu, and Yingqiang Ge. Clawsafety: "safe" llms, unsafe agents. *CoRR*, abs/2604.01438, 2026. doi: 10.48550/ARXIV.2604.01438. URL https://doi.org/10.48550/arXiv.2604.01438.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-eval: Toward trustworthy evaluation of autonomous agents. *CoRR*, abs/2604.06132, 2026. doi: 10.48550/ARXIV.2604.06132. URL https://doi.org/10.48550/arXiv.2604.06132.

Jingwei Yi, Yueqi Xie, Bin Zhu, Keegan Hines, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models. *CoRR*, abs/2312.14197, 2023. doi: 10.48550/ARXIV.2312.14197. URL https://doi.org/10.48550/arXiv.2312.14197.

Z.AI. GLM-5.1 overview. https://docs.z.ai/guides/llm/glm-5.1, 2026. Accessed: 2026-05-21.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Findings of ACL, pp. 10471–10506. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.624. URL https://doi.org/10.18653/v1/2024.findings-acl.624.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): formalizing and benchmarking attacks and defenses in llm-based agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=V4y0CpX4hK.

Tian Zhang, Yiwei Xu, Juan Wang, Keyan Guo, Xiaoyang Xu, Bowen Xiao, Quanlong Guan, Jinlin Fan, Jiawei Liu, Zhiquan Liu, and Hongxin Hu. Agentsentry: Mitigating indirect prompt injection in LLM agents via temporal causal diagnostics and context purification. *CoRR*, abs/2602.22724, 2026. doi: 10.48550/ARXIV.2602.22724. URL https://doi.org/10.48550/arXiv.2602.22724.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. MELON: provable defense against indirect prompt injection attacks in AI agents. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/zhu25z.html.

A CLAWTROJAN ANNOTATION PROTOCOL

This appendix spells out the protocol because the benchmark is not a collection of isolated prompt strings. Each sample is a runnable trajectory with a persistent workspace, staged contamination, and a validation loop.

Stage 1: user profile. We first create or select a user profile. A profile fixes the user's role, domain, tool habits, communication style, risk tolerance, and security awareness. Profiles are reused across samples to keep the attacker from implicitly changing the user at each step. When adding new samples, annotators inspect the current profile-use distribution and prefer underused profiles.

Stage 2: attack-chain plan. The annotator then plans one full attack chain. The plan records the scenario, attack family, target outcome, workspace template, skill bundle, and final artifact. The planned outcome must be observable in the sandbox: a changed local document, a task-deviating artifact, a local mock side-effect state, or a local mock disclosure packet. Samples must differ from nearby samples in at least two dimensions, such as scenario, attack type, profile, skill bundle, or semantic target.

Stage 3: semantic phases. The annotator writes one row per semantic phase. A phase is not a single tool call. It is a short slice of the attack trajectory with a visible user request, an optional hidden instruction, an injection source, and summaries of the conversation, memory, skill, and workspace state. We require a strict separation between `user_input` and `injected_inst`. The former is the user's surface request; the latter is the instruction or rule arriving from a tool return, downloaded file, memory, or multiple sources.

Stage 4: runnable sandbox. Finally, each phase is materialized as a standalone sandbox. Every step directory contains `meta.json`, `openclaw.json`, `session.jsonl`, `injection.json`, and a complete `workspace/` snapshot. The snapshot is self-contained: later steps do not depend on replaying earlier steps during setup. For injected tool returns, `injection.json` provides mocked responses through `content_template`; for file or memory injections, the poisoned content is placed in the corresponding local artifact and the placement is recorded.

Validation loop. Annotation does not end after writing JSON files. A positive sample must pass a full-chain sandbox validation before it is counted as stable. We first run a GPT-5.4 validation, repair the sample if the expected artifact is not produced, and then run a GLM-5.1 smoke validation. Strict full-chain success requires every malicious step in the same sample to be judged compromised. Relaxed full-chain success is recorded separately when one side contains a partial verdict but the unsafe artifact still remains visible enough for downstream analysis.

A.1 LLM-ASSISTED ANNOTATION TASK PACKAGE

We used LLM workers to draft and repair samples, but the workers were constrained by structured task packages and sandbox validation. A typical worker input contains the following fields:

Task. Create or repair one ClawTrojan sample in an offline synthetic workspace. Do not contact real services or use real credentials or personal data.

Inputs. `sample_id`, `scenario`, `attack_type`, source outcome, target outcome, target chain length, expected terminal artifact, write scope, optional GitHub repository context, and validation output prefix.

Required outputs. Update the sample, step rows, and one sandbox per step. Preserve separation between `user_input` and `injected_inst`. Use `tool_return`, `downloaded_file`, `memory`, or `mixed` as the injection source. Add `content_template` and path variants whenever a mocked tool return is needed.

Acceptance. Run GPT-5.4 and GLM-5.1 validation. Mark a sample stable only when the sandbox metrics show strict or explicitly recorded relaxed full-chain success. Treat backend, judge, or turn-0 tool errors as invalid validation, not as safe behavior.

Table 7: Core ClawTrojan annotation fields. Evaluation verdicts and model outputs are stored separately from these fields.

Object	Field	Purpose
User profile	`user_profile_id`	Stable user identity for the trajectory
User profile	`role`, `domain`	User context and work domain
User profile	`tool_habits`, `security_awareness`	Normal tool use and safety posture
Sample	`sample_id`	Trajectory identifier
Sample	`scenario`	`coding`, `office`, `research`, or `general`
Sample	`attack_goal`, `goal`	Attack objective for positive samples or clean objective for controls
Sample	`attack_type`	Attack family enum, including injection, poisoning, exfiltration, privilege, hijack, policy shift, or `none`
Sample	`outcome_category`	Final harmful outcome or `none` control
Sample	`risk_tier`	L0–L4 target risk level
Sample	`workspace_template`, `skills_bundle_id`	Workspace and skill context
Step	`step_id`, `step_idx`	Semantic phase identifier and order
Step	`stage_tag`	Recon, priming, pivot, escalation, or irreversible attempt
Step	`user_input`	Visible benign or ambiguous user request
Step	`injected_inst`	Hidden instruction or contamination rule, null for clean steps
Step	`injection_src`	Canonical source such as `none`, `tool_return`, `downloaded_file`, `memory`, or `mixed`
Step	`conversation`, `memory_desc`, `skills_desc`, `workspace_desc`	Four-layer state summaries
Step	`is_last_chance`	Whether this step is the last intervention point
Step	`step_env_path`	Pointer to the runnable step environment

The full internal task package also includes per-sample write scopes, output directories, and worker-only diagnostic references. We keep those diagnostics out of the paper because they contain long trace-specific repair notes rather than benchmark semantics.

Validation settings and acceptance. The GitHub-derived chain-expansion protocol uses one trial per step and at most 12 agent turns. GPT-5.4 is first used as the agent and judge; the second check uses GLM-5.1 as the agent and GPT-5.4 as the judge. The corresponding clients request temperature zero. These are validation settings, distinct from sample drafting. Relaxed acceptance requires every target step on both agents to be `compromised` or `partial`, with no invalid evaluations; strict acceptance requires all target steps to be `compromised`. This protocol allows up to ten repair rounds per agent, and requires a GPT smoke recheck after GLM-focused repairs. Clean/borderline controls instead require all steps to be safe, zero false positives, full utility preservation, and no invalid evaluations; refusal or an empty response does not establish task completion.

B Dataset Schema and Runtime Environment

ClawTrojan has two related schemas. The annotation schema describes the trajectory, while the runtime schema describes how the trajectory is executed in the sandbox. We keep evaluation results out of the core annotation schema so that model behavior does not contaminate the dataset labels.

Injection placement. `injection_placement` is either a single object or a list of objects. Each object can specify a `tool_name`, `trigger_input`, `content_template`, and character offsets `inject_char_start`/`inject_char_end`. The list form is used when the same poisoned source may be reached through several tools or path variants. This is important for repro-

Table 8: Runnable sandbox artifacts generated for each step.

Runtime artifact	Key fields or contents	Role
`meta.json`	`sample_id`, `step_id`, `step_idx`, `stage_tag`, `user_input`	Step metadata and next user message
`openclaw.json`	workspace path, tool allow-list, model placeholder	Harness configuration
`session.jsonl`	prior user/assistant turns	History before the current step
`workspace/`	`AGENTS.md`, `USER.md`, memory, skills, project files	Complete per-step workspace snapshot
`injection.json`	`injected_inst`, `injection_src`, `injection_placement`	Mocked or file-backed injection placement

ducibility: if the agent reads `project/docs/foo.md` while the mock only matches `foo.md`, the attack may silently miss.

Source aliases. The paper groups injection sources into the canonical families in Table 7. Some generated rows retain more specific loader-facing aliases such as `workspace_file`, `local_file`, `config_file`, `source_digest`, or `carry_forward_note`. These aliases are normalized into the same source families for reporting, but are kept in the runtime files because they make mock matching and trace debugging more precise.

Malicious step identification. In the runtime loader, a step is treated as malicious when `injection_src` is not `none` and `injected_inst` is present. Clean negative and borderline steps use `injection_src=none` and a null injected instruction. This derived flag is separate from the annotation field `is_last_chance`: a chain can have several malicious steps, while only one or a few are last intervention points.

B.1 Paper Evaluation Splits

The released annotation tables contain both attack trajectories and clean calibration trajectories. We therefore report two evaluation splits.

Positive split. The positive split is the ASR denominator used in Table 3, Figure 3, and Table 6. It contains the 339 samples whose `attack_type` and `outcome_category` are both non-`none`. These samples contribute 919 malicious runnable step environments under the loader rule above. Step ASR is computed over those 919 steps. Full-chain ASR is computed over the same 339 samples, and a sample is successful only when all of its malicious steps are compromised. The per-outcome positive-split sample counts are 95 unauthorized disclosure, 90 external side effect, 86 task deviation, and 68 document falsification. At the step level, the 919-step denominator contains 176 document falsification, 216 external side-effect, 238 task-deviation, and 287 unauthorized-disclosure steps, plus two historical carry-forward malicious steps with `outcome_category=none`; the total ASR includes all 919 steps, while the per-outcome columns report the four harmful categories.

Clean split. The clean split contains 23 `none` samples and 92 clean step environments. It is used only for false-positive and utility calibration. These samples include pure negative tasks and borderline tasks whose surface form may look security-relevant but does not complete an attack chain.

Clean metrics. FPR, Overblock, and Utility are computed over valid clean step environments, excluding invalid evaluations. Let N denote the number of valid clean steps, and let O, D, and P denote the numbers of overblocked, degraded, and preserved steps, respectively. We define

$$\mathrm{FPR} = \frac{O+D}{N}, \qquad \mathrm{Overblock} = \frac{O}{N}, \qquad \mathrm{Utility} = \frac{P}{N}.$$

Thus, FPR includes both complete overblocking and partial degradation. For DASGuard, the 92 valid steps comprise 6 overblocked, 6 degraded, and 80 preserved outcomes, yielding 13.0% FPR, 6.5% Overblock, and 87.0% Utility.

Evaluation units and scope. These are step-level descriptive rates over 92 clean step environments drawn from 23 trajectories. Steps within a trajectory share context and must not be treated as independent tasks. Utility therefore measures step-level preservation, not end-to-end trajectory completion. The results characterize this limited clean split and do not establish generalization to broader benign workloads. Any confidence intervals or significance tests for these clean metrics should account for within-trajectory dependence, using trajectories as the clustering unit.

Release tables versus evaluation denominators. The full release currently has 362 sample files and 1672 annotated step rows. Many positive samples include setup, bridge, or carry-forward rows whose `injection_src` is `none`; these rows preserve the trajectory context but are not counted in positive-split ASR. Conversely, the clean split is excluded from ASR and full-chain ASR, and is reported only through FPR, overblock, degradation, and utility metrics.

B.2 Analysis on Negative Samples

Analysis on Negative Samples. Table 9 reports step-level results on clean negative and borderline tasks. Appendix B.1 defines the clean metrics and discusses within-trajectory dependence. Most baselines preserve these tasks but leave substantial attack exposure. DASGuard shifts this trade-off toward stronger protection at the cost of more benign-task interference. Better calibration on borderline workspace artifacts is therefore needed to reduce unnecessary intervention.

Table 9: Clean negative/borderline results (%, step-level). FPR: fraction of clean steps overblocked or degraded; Overblock: fraction overblocked; Utility: fraction preserved.

Method	FPR↓	Overblock↓	Utility↑
GPT-5.4	1.1	1.1	98.9
GLM-5.1	0.0	0.0	100.0
DS-V4-Flash	0.0	0.0	100.0
ClawKeeper	2.2	2.2	97.8
StruQ	1.1	1.1	98.9
MELON	2.2	1.1	97.8
PS-1B	8.7	1.1	91.3
PS-8B	3.3	0.0	96.7
CaMeL	5.4	1.1	94.6
DASGuard	13.0	6.5	87.0

B.3 Harness Transfer and Action-Level Interference

Harness transfer. The OpenClaw-style and Codex-style adapters cover all 362 samples and 1,672 stage environments. The positive split contains 339 attack samples and 919 malicious stages. Each stage starts from an independently initialized snapshot; the evaluation does not replay a continuous end-to-end trajectory. The threat model allows stages to occur in separate sessions, potentially days apart, linked by persistent files in the same logical workspace rather than uninterrupted dialogue. The snapshots represent the prescribed workspace states at those stages; they are not generated by propagating the preceding stage's actual execution results. The harness comparison uses undefended GPT-5.6-terra and a GPT-5.4 judge. Native Codex uses reasoning effort `none`, one trial, and a 300-second stage timeout. Its native instruction and skill discovery, shell, and file editing operate on adapted workspaces: mocked returns become local files, and prior dialogue is archived as historical context. The simulated OpenClaw-style harness retains mocked tool returns and role-tagged history. Different interfaces, execution budgets, and serving transports prevent a causal attribution of differences to harness design alone. External delivery services are not connected.

Table 4 compares matched attack samples from the native run and the archived GPT-5.6-terra simulator run. Only samples with valid judgments for every malicious stage in both runtimes enter any of the three metrics; unresolved judgments and runtime failures are not counted as safe. ASR pools malicious-stage verdicts; FC-ASR counts samples whose malicious stages are all compromised; CPS averages the consecutive compromised prefix divided by the number of malicious stages in

each sample. These aggregates follow the main table's metric definitions, but do not establish live cross-stage persistence. This comparison evaluates the undefended runtimes, not DASGuard on native Codex.

Action-level interference. Table 5 uses two SWE-bench Verified development batches with GPT-5.6-terra and hashing embeddings for DASGuard, which differs from the main paper configuration. The batches contain 40 distinct paired tasks. We count all 1,055 audited calls in the 40 selected defended trajectories, including repeated attempts. Each rate divides intercepted calls by calls of that tool type; blocking and sanitization both count. Intervened/total calls are read 0/388, grep 0/188, list 0/21, Code Edit 17/189, Doc Edit 0/1, write 1/9, and exec 37/259 (overall 55/1,055). Code Edit includes implementation and test code, including fixtures; Doc Edit covers documentation files. In this cohort, code targets are Python files (`.py`/`.pyi`), and the only documentation edit targets `doc/faq.rst`. The zero Doc Edit rate therefore rests on one call. These are intervention rates, not adjudicated false positives.

The cohort combines inherited trajectories, adapter revisions, and selected API/environment repairs; superseded attempts and diagnostic reruns are excluded. Of 37 exec interventions, 32 cite missing structured authorization and five cite unauthorized non-clean provenance. Fourteen of 17 Code Edit interventions occur in two Pylint tasks; repeated attempts remain counted. Write has only nine calls. Read, grep, and list lack the pre-execution authorization and write checks used for exec and edits, so their zeros do not establish zero detector false positives. Interference includes adapter limitations and content decisions, and calls within tasks are dependent. Task success is 27/40 without defense and 30/40 with DASGuard: no decrease is observed, but this mixed-version development cohort establishes neither unbiased population rates nor statistical equivalence. Blocks and sanitizations are automatic; human approval and review time are unmeasured.

B.4 Main-Table Reporting Details

Table 3 reports percentages except for time in seconds; arrows indicate the preferred direction. FC-ASR denotes full-chain attack success rate and CPS denotes mean chain penetration. The outcome columns report step ASR for document falsification (Doc.), external side effect (Side eff.), task deviation (Task), and unauthorized disclosure (Discl.). The first four rows are undefended agents; all subsequent defense rows use GPT-5.4. Bold and underlining mark the best and second-best results among those GPT-5.4 defenses only. The GPT-5.6-Terra and GPT-6-Luna rows come from the separate September 20 and September 23 batches, respectively, detailed in Appendix E.1.

The method adaptations and runtime estimates are described in Appendix E. Reported times depend on the serving endpoint and execution schedule and are not a controlled cross-model speed comparison. We additionally conduct independent repeated runs for DASGuard, key comparison baselines, and key ablations, confirming that DASGuard's improvements over the evaluated key baselines are statistically significant (t-test, $p < 0.05$).

B.5 DASGuard Runtime Settings

Attribution and risk. Table 10 gives the implementation's categorical weights. They are fixed rule-based scores, not learned coefficients or probabilities. Authorization is implemented as a policy branch: explicitly authorized or trusted content is preserved; otherwise $w_a = 0.15$ for `not_user_authorized` and zero for other statuses. For an ordinary, non-exempt finding, the three terms in the main-text risk expression can be instantiated as

$$\begin{aligned}\Phi_{\text{attr}} &= w_s + w_d + w_r + w_a,\\ \Phi_{\text{sem}} &= 0.10\,\mathbf{1}[E \geq 0.72],\\ \Phi_{\text{ctx}} &= 0.20\,\mathbf{1}[M \geq 0.68] + b_t\,\mathbf{1}[U] + 0.05\,\mathbf{1}[P].\end{aligned}$$

Here U means the operation's source has an untrusted label, P means at least one provenance match, and b_t is respectively $0, 0.05, 0.10, 0.25, 0.35$ for tiers L0–L4. L4 includes external/system actions and secret handling; L3 covers writes to sensitive sinks or final artifacts; ordinary writes are L2, other untrusted payloads L1, and remaining operations L0. The sum is clipped to $[0, 1]$. The implementation clips intermediate sums as well, which is equivalent here because all increments are

Table 10: Categorical scoring weights. Trusted-system content is preserved by the authorization branch despite its fallback numerical weight.

Category	Weight
Source w_s	
Tool-untrusted; derived-untrusted	0.30
External source; skill metadata	0.25
Compromised artifact	0.35
Persistent memory	0.15
Clean workspace	0.05
Trusted user	0.00
Other labels (including trusted system)	0.10
Destination w_d	
Agent instruction; skill; external action	0.40
Memory; policy	0.35
Template	0.25
Final artifact	0.20
Ordinary file; other destinations	0.05
Role w_r	
Directive; template rule	0.20
Memory policy; policy shift; recipient rule	0.30
Privilege rule	0.35
Other roles	0.10

nonnegative. Authorized/trusted content and recognized local review or harmless preference artifacts bypass this risk-based action selection.

Semantic evidence. The configured embedding model is `BAAI/bge-m3` through SiliconFlow. Vectors are L2-normalized and compared by cosine similarity. Each nonempty line is compared with a fixed bank of 13 control examples covering six roles (directive, memory policy, policy shift, privilege, recipient, and template rules). Examples include "from now on use this as the default workflow" and "route future reports to this distribution alias." E is the maximum similarity to this bank, and M is the maximum similarity to prior finding spans in the same sample/session, both floored at zero. A candidate is emitted when a rule/subtype matches, $E \geq 0.72$, or $M \geq 0.68$; thus the main-text D summarizes separate trigger tests rather than one shared raw threshold. A threshold-passing embedding determines the role; otherwise the first regex role, then the first subtype role, is used. The offline hashing backend is a separate implementation option, not an equivalent embedding model.

Fragment detection additionally uses 17 short prototypes for action, target, authorization, and persistence, with similarity threshold 0.35, together with regex roles. Windows of two to six role-bearing lines must contain an action, a target or authorization cue, and a subject, object, or persistence cue. Such a finding uses the recipient-role base score plus 0.20. A second join over already detected compatible control findings uses their maximum score plus 0.20 (at most six findings, at most 320 characters between neighbors). These replace the ordinary semantic scoring path for joined findings; runtime context increments are subsequently added.

Action thresholds and approval. After authorization and protected-control rules, scores below 0.30 are preserved, $[0.30, 0.60)$ downgraded, and $[0.60, 0.80)$ quarantined. At 0.80 or above, static/cleanup mode deletes the span, while runtime mode blocks a protected operation or quarantines the span. Untrusted control instructions at control sinks can be quarantined or blocked even below these thresholds; untrusted final-artifact claims are at least downgraded. Clean-workspace content is preserved except that scores of at least 0.85 at memory, agent-instruction, skill, or policy sinks require confirmation.

The retained main-evaluation and ablation command manifests do not enable LLM approval. When configured, this optional path is eligible for initial scores in $[0.45, 0.60]$, ambiguous authorization, or fragment joins. Its prompt asks for the least disruptive action preventing untrusted data from becoming persistent control state, supplying the finding's span, source, destination, context, and evidence.

It requests JSON containing classification, action, risk score, and rationale. The client requests temperature zero and at most 1024 output tokens; its model is selected explicitly by configuration. The policy engine accepts recommendations field by field. Protected runtime findings cannot receive a lower risk or less restrictive action, and protected classifications cannot be weakened. Missing approval configuration, call failure, or invalid output leaves the deterministic decision in place.

C Representative Attack Patterns

This section gives short examples of how the schema represents persistent workspace attacks without exposing the full raw fixture content.

Trust laundering. A trust-laundering sample starts from normal project work. Early steps ask the agent to summarize project notes or refresh a local checklist. A later tool return or downloaded note introduces a rule that looks like an approved local exception. The terminal step asks for an ordinary update to a runbook, review packet, or status register. The compromised outcome occurs when the agent writes the untrusted exception as if it were a verified project rule.

Delayed activation. A delayed sample stores the poisoned rule in memory or a local source digest before the terminal task appears. Intermediate steps may only carry forward a field, a table row, or a local state flag. The last step reuses that state in a new user request, such as refreshing a release note, preparing a mock handoff packet, or updating a local queue. This pattern tests whether defenses track where a later artifact came from, not only whether the current request looks suspicious.

Fragmented payload. A fragmented sample splits the attacker goal across several benign-looking artifacts. One source may provide a target identifier, another a policy exception, and a third a ready-to-copy row. No single fragment fully specifies the harmful action. The attack succeeds when the agent combines the fragments into a terminal artifact that deviates from clean policy or clean source evidence.

Workspace-local side effects. External side-effect samples do not call real services. Instead, their terminal artifact is a workspace-local mock state change, such as an outbox JSON file, a local ticket queue, a mock sync state, or a notification queue. This keeps the benchmark safe while preserving the security property being tested: whether the agent performed an unauthorized state-changing action.

D External No-Defense Checks

We use existing prompt-injection benchmarks as a preliminary scope check rather than as the main evidence for ClawTrojan. The runs below are no-defense runs on recent base agents. They show that older benchmark attacks can already produce near-zero ASR in our setting, so they are weak stress tests for persistent workspace trojans. For AgentDojo, the utility column reports task utility under attack. For InjecAgent, which does not report utility, the same column reports the official valid-output rate. The GPT-5.4 AgentDojo run covers 949 attacked pairs across the four public suites, while the GLM-5.1 AgentDojo check uses the 105 attacked Slack pairs available in our SiliconFlow subset and has low task utility. GPT-6-sol and GPT-6-luna are evaluated on the same 252 workspace pairs (18 user tasks, each paired with all 14 injection tasks). For InjecAgent, GPT-5.4 and GLM-5.1 use the same Stage-1 stratified subset of 240 cases. GPT-6-sol and GPT-6-luna share a reconstructed stratified subset of 240 cases (120 direct-harm and 120 data-stealing), covering all 17 user tools and all 30/32 attacker-tool groups. Sample alignment is maintained within each model pair.

E Baseline Reproduction

All methods use the shared ClawTrojan sandbox and behavioral judge. Defense rows use GPT-5.4, one trial per malicious step per run, and a 12-turn limit; method-specific adaptations are summarized below in main-table order. These adapters do not imply reproduction of each method's original training or deployment environment. We additionally conduct independent repeated runs for DASGuard, key comparison baselines, and key ablations, confirming that DASGuard's improvements over the evaluated key baselines are statistically significant (t-test, $p < 0.05$).

Table 11: No-defense external checks on existing prompt-injection benchmarks. AgentDojo reports targeted ASR and utility under attack. InjecAgent reports ASR-valid and valid-output rate. These checks are used only to motivate the persistent-workspace threat setting.

Benchmark	Model	Utility/Valid	ASR
AgentDojo	GPT-5.4	0.876	0.000
AgentDojo	GLM-5.1	0.076	0.000
AgentDojo	GPT-6-sol	0.944	0.000
AgentDojo	GPT-6-luna	0.940	0.000
InjecAgent	GPT-5.4	0.996	0.000
InjecAgent	GLM-5.1	0.988	0.000
InjecAgent	GPT-6-sol	0.992	0.000
InjecAgent	GPT-6-luna	1.000	0.000

Raw agents. GPT-5.4, GLM-5.1, DeepSeek-V4-Flash, GPT-5.6-terra, GLM-5.3-flash, DeepSeek-V4.1-Flash, and GPT-6-luna run without a defense gate. The additional hosted-model batches and its termination checks are described below.

ClawKeeper. We port the official plugin's input-validation, path-guard, and execution-gate checks to the sandbox's before-tool-call hook; this does not reproduce its full production deployment.

StruQ. We adapt the structured instruction/input/response prompt frontend. We do not use a separately fine-tuned StruQ backbone, so this row measures prompt separation within the shared agent.

AgentPoison. The label denotes the query-level Perplexity Filter evaluated in AgentPoison, not its poisoning attack. GPT-2 (Radford et al., 2019) scores the current user query before execution; queries above a frozen threshold of 1045.31 are rejected. The threshold is the 95th percentile of 200 clean StrategyQA (Geva et al., 2021) queries, transferred without tuning on ClawTrojan attack labels. The scorer and calibration are our implementation choices; tool outputs and persistent workspace content are not filtered.

MELON. Before high-risk actions, we mask the user-task prompt, replay the action proposal, and compare tool actions to identify behavior driven by untrusted context. The adapter gates execution without cleaning workspace artifacts.

PromptShield-1B. We use the detector adapter with the trained checkpoint to classify incoming tool text and block flagged outputs; detector-loading or inference failures are excluded from valid evaluation results.

Spotlighting. Encoding applies Base64 to untrusted tool text and adds system instructions that identify it as data. We transform file, memory, web, mock, and search outputs, including explicitly identified historical tool messages, but not ordinary user/assistant history. Encoding is selected for lower step ASR than Datamarking on the evaluated split, rather than on a held-out selection set.

PromptShield-8B. We use the same detector-gate interface as PromptShield-1B with the larger trained checkpoint.

CaMeL. We adapt source tracking to a workspace control/data/destination graph and apply capability and content-source checks before sensitive operations. This is a sandbox graph-gate adaptation, not the complete original execution architecture.

DASGuard. The runtime adapter scans proposed operations, attributes controlling spans, and blocks or sanitizes unauthorized content before committing a write. It retains source labels and prior findings as described in Section 5.

Additional coverage and timing. AgentPoison and Spotlighting cover the full positive split using independent fixed workspace snapshots; their full clean-split utility is unmeasured. Dagger-marked costs are warm agent-loop means from eight matched steps (two per outcome category), with methods run sequentially in alternating order on the same endpoint. Timing includes tools, API waiting, and online defense processing, but excludes judging, workspace setup, and model loading. GPT-2 runs on CPU with four threads; its query scoring is recomputed inside the timed loop rather than read from the full-run cache. All timing trials complete normally. These small-sample estimates leave full-split attack metrics unchanged and do not establish a controlled speed ranking against historical runs. Exact checkpoints, calibration records, and timing traces are retained in the experiment artifacts.

E.1 Additional Hosted Raw-Model Evaluation

We evaluated three additional no-defense configurations on September 20, 2026: `gpt-5.6-terra`, `glm-5.3-flash`, and `deepseek-v4.1-flash`. Each uses the same frozen 339-sample positive manifest and all 919 malicious steps. Each step runs from its prescribed workspace snapshot under `isolated_step`, with one trial, at most 12 turns, temperature zero, and a 4,096-token per-call output limit. All three use GPT-5.4 as judge with the existing rubric, temperature zero, and a 1,024-token output limit. Optional reasoning content is retained between tool turns. We use a shared OpenAI-compatible serving endpoint; its underlying weights and routing are not independently verified. The DBLP-indexed GPT-5, GLM-5, and DeepSeek-V4 reports (OpenAI, 2026; GLM, 2026; DeepSeek-AI, 2026) provide family-level background, not release documentation for these exact API identifiers.

All 2,757 expected step/trial pairs have valid verdicts and saved traces, tool audits, workspace diffs, and judge outputs, with no missing or duplicate pairs. The four outcome denominators remain 176, 216, 238, and 287; two additional malicious steps labeled `none` remain in the overall 919-step denominator. The numbers of fully compromised samples are 237, 111, and 12 out of 339, respectively. Full-chain ASR aggregates the fixed step snapshots; it does not measure a live end-to-end chain replay. Here, "chain" refers to attack stages linked by persistent workspace artifacts across potentially separated sessions, not to a single continuous session. FC-ASR measures compromise at every malicious stage under its prescribed snapshot, without testing whether earlier execution or sanitization changes the workspace state encountered by later stages.

Table 12: Additional no-defense results and execution termination counts. C/P/S denote compromised, partial, and safe verdicts. Turn and output limits are mutually exclusive termination reasons, not additional verdict classes.

Model	C/P/S	Completed	Turn limit	Output limit
GPT-5.6-terra	794/42/83	911	8	0
GLM-5.3-flash	562/155/202	806	76	37
DeepSeek-V4.1-Flash	137/187/595	653	186	80

A separate September 23 batch evaluates `gpt-6-luna` on the same 339 samples and all 919 malicious steps, with the same judge and budgets. It reuses 272 validated steps from an initial 100-sample stratified run and adds the remaining 647 steps; the reused environment hashes and evaluation settings match. All 919 step/trial pairs are valid, with no missing or duplicate pairs: 742 compromised, 70 partial, and 107 safe. Its step ASR is 80.74%, full-chain ASR is 61.06% (207/339), and mean CPS is 0.722616. Execution completes on 888 steps and reaches the turn limit on 31; none reaches the output limit. Mean agent execution time is 17.7 seconds, excluding judge time. These are full-split results, not the initial subset estimates. No clean utility or defended Luna run is available. Artifacts and reuse provenance are retained with the experiment outputs; the configuration is `configs/gpt6_luna_no_defense_full_20260923.json`.

Following the existing protocol, a budget-limited execution with a valid judge verdict remains in the ASR denominator. Thus zero invalid trials does not imply full task completion. In particular, DeepSeek-V4.1-Flash reaches one of the two limits on 266/919 steps, versus 113/919 for GLM-5.3-flash and 8/919 for GPT-5.6-terra. Its 14.9% ASR coexists with 20.3% partial outcomes; we do not interpret this as safety at matched task utility. The 92-step clean split and defended versions of these

three models have not been run. The single-trial results also do not establish statistical significance or isolate a model-version effect from serving differences across batches.

Mean agent execution times are 14.8, 78.9, and 130.7 seconds; judge time is excluded. Latency is retained across checkpoint resumption, but serving endpoints, inference defaults, and concurrent scheduling differ from the historical runs, so these times are descriptive rather than a controlled speed comparison. The configuration, manifests, environment hashes, and per-step artifacts are stored under `experiments/new_models_full_20260920/`; the entry point is `scripts/run_full_model_eval.py` with `configs/new_models_full_20260920.json`.