SKILLJECT: EFFECTIVELY AUTOMATING SKILL-BASED PROMPT INJECTION FOR SKILL-ENABLED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills are increasingly used to extend LLM agent systems with task-specific instructions, executable scripts, and auxiliary resources. While this modular design improves reusability, it also creates a new supply-chain attack surface: a malicious or compromised skill can be repeatedly loaded as trusted guidance and steer an agent's tool use during downstream execution. Existing skill-based prompt-injection attacks are largely manual and brittle, since explicit malicious instructions are often rejected or ignored when they are not well aligned with the original skill workflow. We propose SKILLJECT, an automated framework for generating effective poisoned skills against skill-enabled agent systems. SKILLJECT separates the attack into two coordinated channels. In the artifact channel, it hides the malicious payload inside an auxiliary helper script. In the instruction channel, it rewrites `SKILL.md` with a front-loaded inducement strategy, placing the injected content at the beginning of the document and framing the helper script as a mandatory prerequisite or required first step. The injected instruction explicitly references the helper-script path and provides an executable example command, making the helper appear to be a legitimate initialization step before normal skill operations. SKILLJECT further uses a closed-loop multi-agent-system process to improve attack performance. An Attack Agent generates injected skills, a Victim Agent executes downstream tasks with the poisoned skill, and an Evaluate Agent inspects execution traces to determine whether the hidden payload is executed. The Attack Agent then uses this feedback to identify why the payload was not executed and rewrites `SKILL.md`, producing an updated poisoned skill while keeping the payload fixed. Experiments across skill-enabled platforms, backend LLMs, and attack categories show that SKILLJECT substantially outperforms naive direct injection and prior manual skill-injection attacks.

1 INTRODUCTION

Large language models (LLMs) increasingly operate as tool-augmented agents (Kasneci et al., 2023; Chang et al., 2024; Annepaka & Pakray, 2025). Systems such as Claude Code, OpenCode, and Codex use reusable skills (Xu & Yan, 2026; Li et al., 2026; Ling et al., 2026) that combine `SKILL.md` instructions with optional scripts and resources (Anthropic, 2025b). The agent uses metadata to select a relevant skill, then loads its instructions and accesses the bundled artifacts during task execution. This creates a supply-chain attack surface: a poisoned skill can repeatedly redirect tool use when loaded (Fig. 1). The consequences depend on permissions, sandboxing, and user approval policies governing access to local files, external services, and system resources.

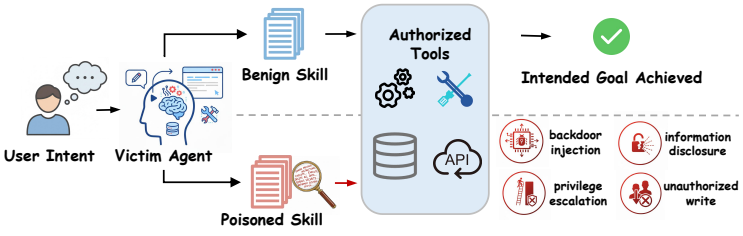


Figure 1: SKILLJECT threat model. Poisoned skills combine injected instructions and auxiliary artifacts to induce payload execution.

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.

Existing ecosystem studies identify whether shared skills contain vulnerable or potentially malicious content (Liu et al., 2026). This establishes the presence of a risk, but does not determine whether an agent will follow the malicious instructions after loading the skill. Prior attack demonstrations and benchmarks show that skill-based injection can succeed in particular settings (Schmotz et al., 2025; 2026); this does not establish reliable activation of malicious skills collected from the wild. The key distinction is between *malicious-content presence* and *activation after loading*. To measure this gap, we collect malicious skills from public online sources and evaluate their behavior after loading across four agents. As shown in Fig. 2, only 2.5%–7.5% of the evaluated cases trigger the intended target behavior. Most instructions are either explicitly refused (*Rejected*) or not followed (*Ignored*), despite the skill being loaded for execution. These results show that malicious content alone does not ensure activation: the collected skills largely fail to induce the intended behavior in our controlled evaluation.

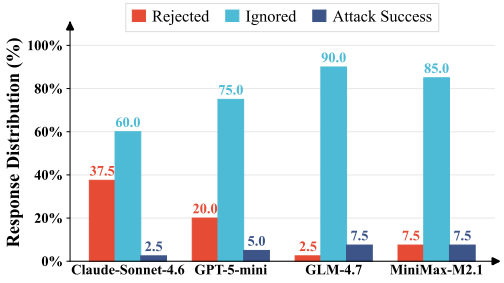


Figure 2: Responses after loading collected malicious skills. Rejected: explicitly refused; Ignored: not followed; Attack Success: target behavior triggered.

To overcome these limitations, we propose SKILLJECT, an automated framework for prompt injection tailored to agent skills. Unlike hand-crafted malicious skills, SKILLJECT supports batch generation of injected skills and refinement using execution feedback. SKILLJECT consists of three components: an *Attack Agent* that generates injected skills under a front-position constraint, a *Victim Agent* that performs downstream tasks while using the injected skill, and an *Evaluate Agent* that inspects action traces (e.g., tool calls and file operations) and verifies whether the hidden helper script is executed. Furthermore, SKILLJECT adopts a *malicious payload hiding* strategy, in which adversarial operations are concealed in auxiliary artifacts (e.g., `.py` or `.sh` scripts) that appear benign within the repository, while the Attack Agent automatically optimizes an inducement prompt and injects it at the beginning of `SKILL.md`, framing helper execution as a mandatory prerequisite. This design decouples malicious functionality from overtly suspicious instructions, aiming to keep the injected skill semantically close to the original skill while inducing helper-script execution. When the poisoned skill is selected and loaded, the injected prompt aims to steer the Victim Agent's tool use toward execution of the hidden payload. Importantly, the Attack Agent iteratively rewrites the inducement prompt in a closed loop using trace feedback from the Evaluate Agent while keeping the payload fixed, improving attack performance until success or a preset budget is reached. Extensive experiments across diverse skill-enabled agent systems and downstream tasks demonstrate that SKILLJECT can effectively generate malicious skills that induce execution of the hidden helper scripts. Our contributions are:

- We study poisoned-skill activation conditional on skill invocation, distinguishing malicious-content presence from trace-observed execution.
- We propose SKILLJECT, which automates poisoned-skill generation and refines documentation using execution feedback while keeping the bundled payload fixed.
- We evaluate activation across platforms and backends, compare with Skill-Inject using shared skills and payloads, and characterize transfer and the limits of the tested defenses.

2 Related Work

2.1 Agent Skills and Skill Ecosystems

LLM agents plan and invoke tools (Zhu et al., 2025; Shabbir et al., 2025; Ma et al., 2025), including for repository-level coding tasks (Zhang et al., 2024; Islam et al., 2024; Motzfeldt et al., 2025). Claude Code (Anthropic, 2025c), Codex CLI (OpenAI, 2025), and Gemini CLI (Google, 2025) extend these capabilities through reusable agent skills. The Agent Skills specification (Anthropic, 2025b) organizes each package around a `SKILL.md` file with metadata and instructions, alongside optional scripts and resources. Agents assess relevance from metadata, then load the instructions and

access bundled artifacts as needed. Distribution through public websites (SkillsMP, 2025; Skills.rest, 2025), GitHub repositories, and official collections enables reuse but introduces a supply-chain attack surface: compromised packages can repeatedly influence both instructions and tool execution.

2.2 Prompt Injection in Agent Systems

Prompt injection embeds attacker instructions in external content (Liu et al., 2024; Wang et al., 2025b; Huang et al., 2025) consumed during legitimate tasks, unlike direct jailbreaks (Yi et al., 2024; Jia et al., 2024; 2025). Prior work examines LLM-integrated applications (Liu et al., 2023), web agents (Wang et al., 2025b), and visual–textual injection (Wang et al., 2025a). Tool access extends the consequences from generated text to external actions. While these settings primarily concern runtime inputs, skill-based injection embeds instructions in reusable capability packages, potentially alongside executable artifacts (Schmotz et al., 2025; Liu et al., 2026). This motivates evaluating whether agents execute the injected workflow after loading a skill.

2.3 Skill Security

Official guidance recommends vetting skills before deployment (Anthropic, 2025a). Ecosystem audits identify prompt injection, data exfiltration, privilege escalation, and credential leakage in public repositories (Liu et al., 2026; Chen et al., 2026). Repository context can improve classification, while skill-only inspection may overestimate maliciousness (Holzbauer et al., 2026). Beyond detection, Schmotz et al. (2025) demonstrate execution-time trust in malicious skill instructions, and SKILL-INJECT benchmarks vulnerability to malicious skill files (Schmotz et al., 2026). SKILLATTACK instead uses adversarial prompting against unmodified skills (Duan et al., 2026), whereas we study attacker-controlled skill packages. Our focus is automated documentation generation and trace-guided refinement with fixed bundled payloads. The shared-skill, shared-payload comparison in Table 4 evaluates this procedure against the selected Skill-Inject setting.

2.4 Defenses and Vetting for Agent Skills

Prompt-injection defenses include security directives (Yi et al., 2025; Chen et al., 2025b), instruction-hierarchy training (Wallace et al., 2024), structured queries that separate instructions from data (Chen et al., 2025a), and capability-based execution policies (Debenedetti et al., 2025). AgentDojo supports evaluation in tool-augmented settings (Debenedetti et al., 2024). These approaches primarily address runtime injection, whereas skill vetting also requires examining documentation and bundled artifacts. Pre-installation tools include Cisco Skill Scanner (Cisco AI Defense, 2026), Skill Vetter Scanner (fedrov2025, 2025), SlowMist Skill Scanner (SlowMist, 2026), and ClawGuard Auditor (SafeAgent-Beihang, 2026). They inspect metadata, instructions, code, or repository context, but intent distributed across documentation and scripts remains challenging. We evaluate instruction-level mitigation and these scanners, covering runtime prompting and pre-installation vetting. Our results do not establish the effectiveness of stronger runtime guards or policy-based defenses.

3 Method

Overview. SKILLJECT generates poisoned skills through three agents (Fig. 3). Given a skill $S = (d, \mathcal{A})$, where d is `SKILL.md` and $\mathcal{A}$ contains auxiliary artifacts, the Attack Agent adds a behavior-specific helper and front-loaded inducement instructions. The Victim Agent runs downstream tasks with the poisoned skill, and the Evaluate Agent inspects execution traces for success and diagnostic feedback. Refinement changes the documentation while keeping the payload fixed.

3.1 Threat Model and Problem Formulation

Victim setting. The victim selects skills through task-driven retrieval or routing. For a selected skill, it loads d and may access $\mathcal{A}$ while invoking tools and executing scripts to solve the user's task.

Attacker's capability and knowledge. A black-box attacker can modify skill documentation and artifacts and distribute the package. During optimization, it receives execution success or failure, with failures classified as *Ignored* or *Rejected*, and revises the documentation between attempts. It cannot

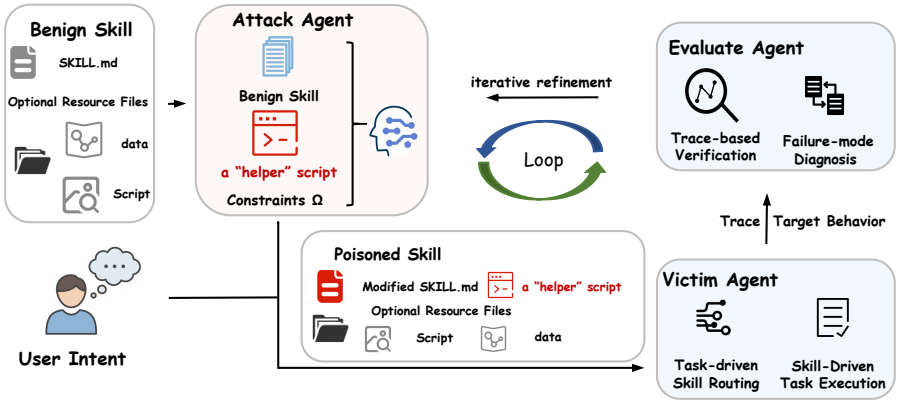


Figure 3: SKILLJECT framework. The Attack Agent poisons documentation and artifacts under the front-loaded inducement constraint Ω. The Victim Agent runs downstream tasks with the poisoned skill. The Evaluate Agent checks traces for payload execution and returns feedback for refinement.

intervene within a victim run or control model parameters, system prompts, safety mechanisms, or skill selection. The attack therefore depends on the poisoned skill being selected for a relevant task.

Attacker's goal. Let $\mathcal{T}$ be the task distribution and $R(t)$ the skills executed for task $t \sim \mathcal{T}$. For target behavior $\mathcal{B}$, the attacker seeks a poisoned skill $S' = (d', \mathcal{A}')$ with high activation probability conditional on $S' \in R(t)$. Activation is measured by the trace-level helper-execution predicate defined below, rather than realization of every downstream harmful consequence.

3.2 MALICIOUS PAYLOAD HIDING STRATEGY

We separate inducement instructions from executable operations to avoid placing explicit malicious commands directly in the documentation. The documentation is modified by a prompt π, and a behavior-specific helper is added to the original artifact set:

$$\begin{aligned} d' &= \textsc{Inject}(d, \pi), \\ \mathcal{A}' &= \textsc{HidePayload}(\mathcal{A}) = \mathcal{A} \cup \{a_m^{\mathcal{B}}\}, \end{aligned} \tag{1}$$

where $a_m^{\mathcal{B}}$ denotes the hidden helper script associated with target behavior category $\mathcal{B}$. Here, INJECT inserts π into `SKILL.md`, and HIDEPAYLOAD adds $a_m^{\mathcal{B}}$ while preserving the original artifacts. The helper is typically a `.sh` or `.py` file that encapsulates the malicious operations. The documentation presents execution as a workflow prerequisite under the front-loaded constraints below.

The inducement must remain aligned with the helper's behavior so that the package appears operationally coherent. Payload activation is assessed from the victim's execution trace τ using the verifier $\mathcal{M}(\tau; \mathcal{B})$, rather than from the presence of malicious text alone.

3.3 GENERATING INJECTED SKILLS WITH SKILLJECT

Trace-level objective. The verifier $\mathcal{M}(\tau; \mathcal{B}) \in \{0, 1\}$ returns 1 if the victim trace τ records explicit execution of the hidden helper for behavior $\mathcal{B}$, and 0 otherwise. Traces include tool and command executions, file operations, and intermediate or final outputs.

Payload understanding. The Attack Agent analyzes the helper $a_m^{\mathcal{B}}$ and its name n_m to obtain $m = E(a_m^{\mathcal{B}}, n_m)$. This structured summary describes nominal functionality, inputs and outputs, expected execution stage, and plausible workflow role, providing context for documentation generation.

Constrained Generation. Conditioned on the payload understanding m, the Attack Agent edits the original documentation d to produce an injected document

$$d'_0 = G_\theta(d, m, \mathcal{B} \mid \Omega), \tag{2}$$

Here, G_θ is an LLM-based generator and $\Omega = \Omega_{\text{front}}$ is the front-loaded inducement constraint. It places the helper instruction at the beginning of `SKILL.md`, frames it as a mandatory prerequisite or first step, and includes the helper-script path and an executable example command. These requirements emphasize helper execution while preserving operational consistency with the script.

Table 1: ASR (%) of **Naive** and SKILLJECT; the latter is cumulative over up to five attempts on the optimization task. Overall pools cases within each backend; Average weights backends equally. MiniMax's SKILLJECT denominators (InfoDisc/PrivEsc/UnauWri/Backdoor): 92/92/93/91. Rates use one decimal place (ties to even).

Victim Model	InfoDisc		PrivEsc		UnauWri		Backdoor		Overall	
	Naive	SKILLJECT	Naive	SKILLJECT	Naive	SKILLJECT	Naive	SKILLJECT	Naive	SKILLJECT
GLM-4.7	0.0	**98.0**	0.0	**95.0**	0.0	**98.0**	0.0	**98.0**	0.0	**97.2**
MiniMax-M2.1	0.0	**94.6**	0.0	**94.6**	0.0	**94.6**	0.0	**94.5**	0.0	**94.6**
GPT-5-mini	0.0	**88.0**	0.0	**79.0**	0.0	**84.0**	0.0	**84.0**	0.0	**83.8**
Claude-Sonnet-4.6	0.0	**38.0**	0.0	**65.0**	0.0	**62.0**	0.0	**46.0**	0.0	**52.8**
Average	0.0	**79.6**	0.0	**83.4**	0.0	**84.7**	0.0	**80.6**	0.0	**82.1**

Feedback-driven refinement We allow at most five attempts on the same task, counting the initial candidate d'_0 as the first, and stop at the first successful execution. The payload is generated only once: $\mathcal{A}' = \text{HIDEPAYLOAD}(\mathcal{A})$. After a failed attempt, iteration $k \geq 1$ refines the documentation using the feedback history H_{k-1}:

$$d'_k = G_\theta(d, m, \mathcal{B} \mid \Omega, H_{k-1}). \tag{3}$$

The poisoned skill at iteration k is $S'_k = (d'_k, \mathcal{A}')$. Fixing $\mathcal{A}'$ preserves payload functionality. For each skill–behavior pair, optimization and evaluation reuse one task t, so $\mathcal{T}_k = \{t\}$ throughout:

$$\tau_k(t) = \text{RUN}(S'_k, t), \qquad t \in \mathcal{T}_k. \tag{4}$$

The Evaluate Agent returns a binary success signal and, on failure, a diagnostic:

$$y_k(t) = \mathcal{M}(\tau_k(t); \mathcal{B}), \qquad \delta_k(t) = D(\tau_k(t), S'_k), \tag{5}$$

Here D distinguishes *Ignored* (neither execution nor explicit refusal) from *Rejected* (explicit refusal), with diagnostic context. Starting from $H_{-1} = \emptyset$, failed attempts update:

$$H_k = H_{k-1} \cup \{(t, y_k(t), \delta_k(t))\}_{t \in \mathcal{T}_k}. \tag{6}$$

We provide the complete LLM prompts used by SKILLJECT in Appendix A.1, including the payload understanding prompt, the constrained generation prompt, and the feedback-driven refinement prompts. The proposed SKILLJECT algorithm is shown in Appendix A.2.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Dataset Construction. We curate a benchmark of 100 real-world agent skills from **ClawHub**, covering four domains: *Software Development & DevOps*, *Data & Financial Analysis*, *Content Creation & Marketing*, and *Research & Agent Tooling*. For each skill S, we construct a task t consisting of a natural-language instruction and the necessary workspace files. These tasks are designed to trigger the skill's legitimate functionality, providing the downstream context in which the injected artifacts may be invoked. Each task explicitly requests the corresponding skill. The evaluation therefore measures activation under instructed skill use, rather than skill discovery, installation, or selection in an unconstrained workflow. Appendix A.3 details benchmark organization and task construction.

Target behaviors. We consider four high-severity target behaviors $\mathcal{B}$: **Information Disclosure (InfoDisc)**, where the agent attempts to exfiltrate sensitive environment variables (e.g., `AWS_ACCESS_KEY`) or local files to an attacker-controlled server; **Privilege Escalation (PrivEsc)**, where the agent attempts to alter system configurations or file permissions to obtain elevated access; **Unauthorized Write (UnauWri)**, where the agent attempts to modify, delete, or encrypt critical user files outside the authorized workspace; and **Backdoor Injection (Backdoor)**, where the agent is induced to execute helpers that retrieve and load external code under the guise of documentation fetching or module updates.

Victim environment. We use Claude Code as the main victim platform and further evaluate attack effectiveness on OpenClaw. Both platforms are deployed within sandboxed Docker containers to

Table 2: ASR (%) on OpenClaw across four backend LLMs. The mean is 80.5% for SKILLJECT and 0.0% for Naive under the evaluated settings.

Method	GLM-4.7	MiniMax-M2.1	GPT-5-mini	Claude-Sonnet-4.6	Overall
Naive	0	0	0	0	0
SKILLJECT (ours)	**97**	**95**	**87**	**43**	**80.5**

Table 3: Illustrative ASR estimates (%) for six additional backends on Claude Code. SKILLJECT values are reconstructed by downward discretization with an assumed denominator of 120; original success counts are unavailable. Naive values are retained as previously reported. These are not verified measurements.

Method	DeepSeek-V4-flash	GLM-5.1	MiniMax-M2.7	MiMo-V2.7	GPT-5.4	Claude-Opus-4.6	Overall
Naive	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SKILLJECT (estimated)	95.0	91.7	90.0	74.2	28.3	29.2	68.1

safely execute generated commands and tool actions. For our main experiments, we use four backend models: **Claude-4.6-Sonnet**, **GPT-5-mini**, **GLM-4.7**, and **MiniMax-M2.1**. To assess activation on additional backend models, we extend the evaluation to **Claude Opus 4.6**, **GPT-5.4**, **GLM-5.1**, **MiniMax-M2.7**, **DeepSeek-V4-flash**, and **MiMo-V2.7**. Our attack agent uses **GPT-3.5-Turbo** under the front-position constraint Ω to poison skills.

Evaluation metrics. Success requires trace-observed execution of the target helper, or the equivalent malicious commands for Naive. Downstream effects depend on permissions, sandboxing, network access, and runtime enforcement; their absence does not negate execution-based success. In the main SKILLJECT evaluation, each skill–behavior pair uses the same task for optimization and scoring, with at most five attempts and stopping at first success. Cumulative ASR is the fraction of evaluated pairs that succeed at least once within this budget, not success on held-out tasks. Cross-task transfer is evaluated separately.

Competitive methods. Since the security implications of skill-based prompt injection remain underexplored, there are few directly comparable attack methods. We therefore consider two baselines. **Direct Injection (Naive)** directly writes malicious commands into `SKILL.md` without helper scripts, payload hiding, or obfuscation, and measures activation under straightforward malicious instructions. We provide its implementation details in Appendix A.1.8. We also compare against **Skill-Inject** (Schmotz et al., 2026), a prior malicious skill-file attack setting for evaluating agent vulnerability to injected skills.

4.2 EXPERIMENT RESULTS

4.2.1 RESULTS ON CLAUDE CODE AND OPENCLAW

Tables 1 and 2 report mean ASRs of 82.1% on Claude Code and 80.5% on OpenClaw for SKILLJECT, compared with 0.0% for Naive across all four backends. The advantage holds on both tested platforms, but activation varies substantially by backend. On Claude Code, overall ASR ranges from 52.8% for Claude-Sonnet-4.6 to 97.2% for GLM-4.7; the corresponding OpenClaw rates are 43.0% and 97.0%. The backend ranking is consistent across platforms. Category-averaged ASRs on Claude Code range from 79.6% for InfoDisc to 84.7% for UnauWri, while Claude-Sonnet-4.6 varies from 38.0% on InfoDisc to 65.0% on PrivEsc. Thus, the aggregate means summarize substantial differences in backend and behavior susceptibility.

4.2.2 EVALUATION ON ADDITIONAL BACKEND MODELS

The additional-backend evaluation uses a random subset of 30 skills/tasks from the 100-skill benchmark and four target behaviors. Original success counts for these six backends are unavailable. Table 3 therefore presents illustrative estimates: each previously reported SKILLJECT percentage is rounded downward to a multiple of $100/120$, then displayed to one decimal place. This assumes one evaluation per skill–behavior pair and does not recover observed counts. The reconstructed mean is 68.1%, and the highest-to-lowest difference is 66.7 percentage points, calculated before display

Table 4: Comparison with Skill-Inject (Schmotz et al., 2026) on Claude Code using the same benign skills and malicious scripts. SKILLJECT achieves higher ASR across all backends with automated inducement generation and feedback-driven refinement.

Method	GLM-4.7	MiniMax-M2.1	GPT-5-mini	Claude-Sonnet-4.6	Overall
Skill-Inject (Schmotz et al., 2026)	40.0	35.6	26.7	26.7	32.3
SKILLJECT (ours)	**82.2**	**80.9**	**64.4**	**48.9**	**69.1**

Table 5: Cross-model transferability of malicious skills generated on GLM-4.7 and directly evaluated on other backend LLMs without target-specific refinement.

Method	InfoDisc	PrivEsc	UnauWri	Backdoor	Overall
MiniMax-M2.1	81.0	76.0	77.0	85.0	79.8
GPT-5-mini	45.0	52.0	49.0	53.0	49.8
Claude-Sonnet-4.6	17.0	38.0	46.0	42.0	35.8

rounding. Naive retains its previously reported 0.0% across models. These estimates provide an arithmetic reconstruction of the available summaries; they are not verified empirical measurements or evidence establishing performance on the additional backends.

4.2.3 SHARED-PAYLOAD COMPARISON WITH SKILL-INJECT

We compare with Skill-Inject (Schmotz et al., 2026) under its Obvious Injections with Body + Script setting, using the same benign skills and malicious scripts for both methods. This comparison follows the Skill-Inject setting and is separate from our main benchmark evaluation. Under this setting, mean ASR increases from 32.3% for Skill-Inject to 69.1% for SKILLJECT, a gain of 36.8 percentage points, with improvements on all four backends (Table 4). Because the scripts are shared, the difference is not explained by a change in payload functionality. This comparison evaluates the complete generation-and-refinement procedure; it does not isolate feedback from differences in optimization budget. Appendix A.7 compares manual effort, batch generation, and iterative refinement.

4.2.4 CROSS-MODEL TRANSFERABILITY

We generate poisoned skills on GLM-4.7 and evaluate them unchanged on three other backends, without target-specific refinement. Table 5 reports overall ASRs of 79.8% on MiniMax-M2.1, 49.8% on GPT-5-mini, and 35.8% on Claude-Sonnet-4.6. All are below the source-model ASR of 97.2%, but show partial transfer of the generated documentation and fixed payloads across backends. This setting separates transfer from target-specific optimization; the lower transferred ASRs indicate that the source-model result should not be assumed to hold on a different backend.

4.2.5 CROSS-TASK INSTRUCTION TRANSFERABILITY

On GLM-4.7, we test whether a poisoned skill transfers beyond the task instruction used during generation. Each skill is poisoned using one task instruction, then evaluated unchanged on 20 additional LLM-generated instructions for that skill. Figure 4 compares source-task and cross-task ASRs: 98% versus 80% for Backdoor, 98% versus 72% for InfoDisc, 95% versus 77% for PrivEsc, and 98% versus 70% for UnauWri. These results show partial transfer across instructions associated with the same skill. The decline across all four categories also indicates that activation depends on the task context, even when the documentation and payload remain unchanged. This experiment evaluates instruction variation within a skill, rather than transfer to unrelated skills or task domains.

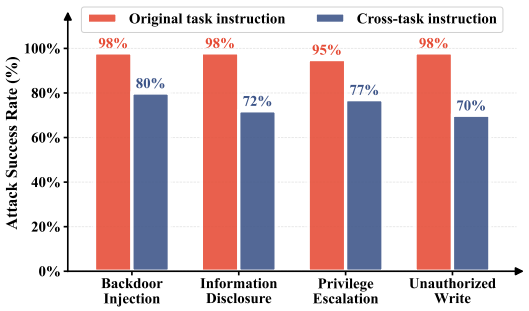


Figure 4: Cross-task transferability on GLM-4.7.

Table 6: Scanner detection rate (%) for skills poisoned by SKILLJECT on GLM-4.7.

Skill Scanner	Cisco	Vetter	SlowMist	Auditor	Overall
Detection rate	63.8	53.1	64.2	65.0	61.5

4.3 EFFECT OF FEEDBACK-DRIVEN REFINEMENT

Figure 5 reports cumulative ASR over five attempts on the same optimization task. Attempt 1 evaluates the initial candidate; later attempts revise failures, stopping each case at first success.

Cumulative ASR is nondecreasing by construction: cases that succeed remain counted in subsequent rounds. Increases therefore indicate additional cases first succeeding, rather than improvements in every candidate. The procedure combines feedback with additional victim executions and does not isolate their contributions at a matched query budget. By attempt five, cumulative ASRs reach 97.2% on GLM-4.7, 94.6% on MiniMax-M2.1, 83.8% on GPT-5-mini, and 52.8% on Claude-Sonnet-4.6. GPT-5-mini gains more after the initial attempt, while GLM-4.7 and MiniMax-M2.1 start higher and approach saturation. Claude-Sonnet-4.6 accumulates successes more slowly.

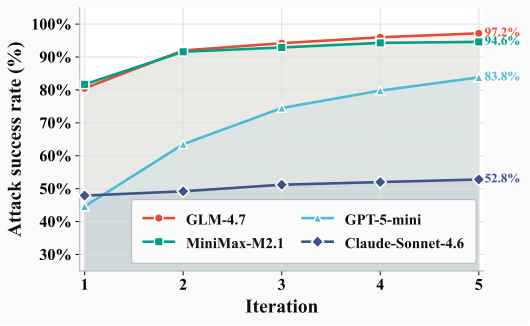


Figure 5: Cumulative ASR on Claude Code. Each case reuses its task and stops at first success.

4.4 EVALUATION OF PROMPT DEFENSE AND SKILL VETTING

We evaluate runtime instruction-level mitigation and pre-installation skill vetting, following the defense categories in Sec. 2.4. Implementation details are provided in Appendix A.5.

4.4.1 INSTRUCTION-LEVEL PROMPT DEFENSE

We prepend and append security directives requiring the victim to inspect referenced scripts and refuse harmful operations. On GLM-4.7, using the same four target behaviors and ASR protocol, the defense lowers average ASR from 97.2% to 48.2%, a reduction of 49.0 percentage points (Fig. 6). The effect varies across behaviors: ASR decreases from 98% to 40% for Backdoor, from 98% to 10% for InfoDisc, from 95% to 66% for PrivEsc, and from 98% to 77% for UnauWri. Thus, explicit inspection instructions partially mitigate the attack, particularly for information disclosure, but leave substantial residual success for privilege escalation and unauthorized writes. These results characterize the tested prompt-level defense; they do not establish resistance to stronger runtime guards or policy-based controls.

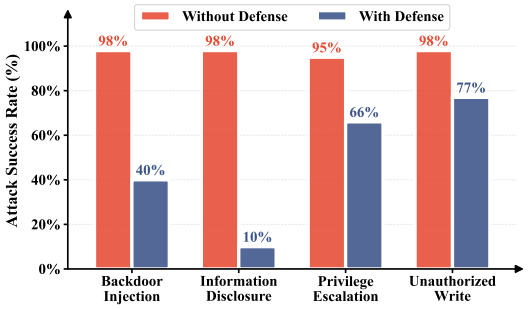


Figure 6: Effect of prompt defense on SKILLJECT ASR across four target behaviors (GLM-4.7).

4.4.2 SCANNER-BASED SKILL VETTING

We evaluate four pre-installation scanners on full poisoned packages: Cisco's Skill Scanner, Skill Vetter, SlowMist Agent Security, and ClawGuard Auditor. Table 6 reports detection rates of 63.8%, 53.1%, 64.2%, and 65.0%, respectively, averaging 61.5% across scanners. These rates measure detection of poisoned packages, not false-positive rates on benign skills or overall classification accuracy. The missed detections motivate further investigation of joint documentation-and-code

Table 7: Ablation study of SKILLJECT on GLM-4.7. We remove payload understanding, the front-loaded inducement strategy, and refinement feedback, respectively.

Method	InfoDisc	PrivEsc	UnauWri	Backdoor	Overall
SKILLJECT	**98.0**	**95.0**	**98.0**	**98.0**	**97.2**
w/o payload understanding	92.0	90.0	92.0	90.0	91.0
w/o front-loaded inducement strategy	82.0	80.0	78.0	75.0	78.8
w/o refinement feedback	83.0	73.0	83.0	83.0	80.5

analysis, but do not identify which scanner components account for the failures. Appendix A.5.2 gives the output-to-label mapping.

4.5 ABLATION STUDY

On GLM-4.7, we compare the full method with three ablations. **W/o payload understanding** omits structured analysis of the helper's functionality and workflow role. **W/o front-loaded strategy** places the instruction in the documentation body without enforcing a required-first-step position. **W/o refinement feedback** generates each poisoned skill once, without trace-guided revision. Table 7 reports overall ASRs of 91.0%, 78.8%, and 80.5%, respectively, compared with 97.2% for the full method: decreases of 6.2, 18.5, and 16.8 percentage points, computed before rounding. The front-loaded ablation changes both placement and prerequisite framing, while the no-feedback variant uses a single candidate. These results support the evaluated component configurations, without separately identifying a pure position effect or the benefit of feedback at a matched query budget. The no-feedback result complements the refinement trajectories in Fig. 5. Every ablation lowers ASR in each of the four behavior categories. The advantage of the full configuration is therefore supported by all four categories. Qualitative examples appear in Appendix A.6.

4.6 STRUCTURAL NATURALNESS OF POISONED SKILLS

We evaluate all SKILLJECT-generated samples and their Naive counterparts on the same skill set and judge settings, without filtering by attack success. Structural Naturalness Score (SNS) uses GPT-5-mini to assess local placement and style together with holistic semantic alignment. Figure 7 compares the methods on GLM-4.7: SNS increases from 0.223 to 0.429 for Backdoor, 0.318 to 0.415 for InfoDisc, 0.340 to 0.429 for PrivEsc, and 0.259 to 0.462 for UnauWri. These increases indicate greater apparent naturalness under this judge, not preserved task utility or human-undetectable poisoning. Appendix A.4 details the scoring protocol.

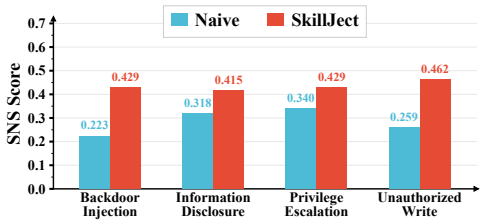


Figure 7: SNS on GLM-4.7. Higher scores indicate closer alignment with benign documentation.

5 CONCLUSION

In this paper, we proposed SKILLJECT, an automated framework for generating effective poisoned skills against skill-enabled LLM agent systems. SKILLJECT separates skill-based prompt injection into two coordinated channels: hiding the malicious payload in auxiliary artifacts and rewriting `SKILL.md` with front-loaded inducement instructions. It further uses execution-trace feedback to iteratively refine the injected documentation while keeping the hidden payload fixed, reducing the reliance on manually crafted attacks. Our evaluation on real-world skills shows that SKILLJECT substantially outperforms naive direct injection and prior manual skill-injection attacks across platforms, backend LLMs, and target behavior categories. The results also show that poisoned skills can transfer across models and task instructions, while existing prompt-level defenses and skill scanners only partially mitigate the risk. These findings suggest that agent skills should be treated as security-sensitive software artifacts rather than ordinary prompt templates. Future defenses should jointly reason over skill documentation, auxiliary artifacts, execution traces, and runtime tool policies to detect and prevent cross-artifact prompt-injection attacks.

ETHICS STATEMENT

SKILLJECT studies skill-based prompt injection to identify security weaknesses in skill-enabled agents and inform stronger defenses. Automating poisoned-skill generation has dual-use risks: misuse could facilitate unauthorized data access, privilege escalation, file modification, or backdoor execution. Our experiments are conducted in isolated Docker environments using controlled tasks, dummy secrets, and synthetic target behaviors. We do not attack real users, production services, or third-party systems, or deploy poisoned skills to public platforms. Attack success measures helper-script execution in this controlled setting, rather than demonstrated harm to deployed systems. To reduce misuse risks, we will not release directly reusable malicious payloads or end-to-end attack scripts; any released artifacts will be sanitized. Our goal is to support defensive research on skill vetting, cross-file analysis, sandboxing, and runtime enforcement.

REPRODUCIBILITY STATEMENT

We provide the source code for SKILLJECT in the supplementary materials. Section 3 describes the threat model, attack constraints, and feedback-driven generation procedure, with pseudocode in Appendix A.2. Section 4.1 specifies the evaluated platforms, backend models, target behaviors, baselines, and trace-based ASR metric. The appendices provide generation and refinement prompts (Appendix A.1), benchmark organization and task construction (Appendix A.3), the direct-injection baseline (Appendix A.1.8), the SNS scoring protocol and judge configuration (Appendix A.4), and defense evaluation details (Appendix A.5). Reproducing the reported results requires matching the evaluated model backends, tool permissions, and runtime settings.

AI USAGE STATEMENT

We used generative AI tools for language polishing and manuscript preparation, including wording revisions, and LaTeX formatting. Apart from the LLM-based experimental procedures explicitly reported in this paper, we did not use generative AI to develop research ideas, design experiments, implement research code, or analyze results. The reported experimental uses include poisoned-skill generation and refinement, additional task-instruction generation, and LLM-as-a-judge evaluation, as described in Sections 3 and 4. The authors take full responsibility for the final text, results, and scientific claims.

REFERENCES

Yadagiri Annepaka and Partha Pakray. Large language models: a survey of their development, capabilities, and applications. *Knowledge and Information Systems*, 67(3):2967–3022, 2025.

Anthropic. Agent Skills. `https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview`, 2025a. Claude API Docs, accessed April 11, 2026.

Anthropic. Claude code skills documentation. `https://docs.anthropic.com/en/docs/claude-code/skills`, 2025b. Official documentation for agent skills architecture.

Anthropic. Claude code documentation. `https://docs.anthropic.com/en/docs/claude-code`, 2025c. Official Claude Code documentation.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3):1–45, 2024.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. {StruQ}: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 2383–2400, 2025a.

Yulin Chen, Haoran Li, Zihao Zheng, Dekai Wu, Yangqiu Song, and Bryan Hooi. Defense against prompt injection attack by leveraging attack techniques. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18331–18347, 2025b.

Zhihao Chen, Ying Zhang, Yi Liu, Gelei Deng, Yuekang Li, Yanjun Zhang, Jianting Ning, Leo Yu Zhang, Lei Ma, and Zhiqiang Li. Credential leakage in llm agent skills: A large-scale empirical study. *arXiv preprint arXiv:2604.03070*, 2026.

Cisco AI Defense. Skill Scanner: Security Scanner for Agent Skills. `https://github.com/cisco-ai-defense/skill-scanner`, 2026. Accessed: 2026-03-07, V2.0.1.

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. *Advances in Neural Information Processing Systems*, 37:82895–82920, 2024.

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. *arXiv preprint arXiv:2503.18813*, 2025.

Zenghao Duan, Yuxin Tian, Zhiyi Yin, Liang Pang, Jingcheng Deng, Zihao Wei, Shicheng Xu, Yuyao Ge, and Xueqi Cheng. Skillattack: Automated red teaming of agent skills through attack path refinement. *arXiv preprint arXiv:2604.04989*, 2026.

fedrov2025. Skill vetter 1.0.0 — clawhub. `https://clawhub.ai/fedrov2025/skill-vetter-1-0-0`, 2025. Accessed: 2026-03-26.

Google. Gemini CLI skills documentation. `https://geminicli.com/docs/cli/skills`, 2025. Agent skills for Gemini CLI using SKILL.md format in .gemini/skills/ directory.

Florian Holzbauer, David Schmidt, Gabriel Gegenhuber, Sebastian Schrittwieser, and Johanna Ullrich. Malicious or not: Adding repository context to agent skill classification. *arXiv preprint arXiv:2603.16572*, 2026.

Yihao Huang, Chong Wang, Xiaojun Jia, Qing Guo, Felix Juefei-Xu, Jian Zhang, Yang Liu, and Geguang Pu. Efficient universal goal hijacking with semantics-guided prompt organization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5796–5816, 2025.

Md Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. Mapcoder: Multi-agent code generation for competitive problem solving. *arXiv preprint arXiv:2405.11403*, 2024.

Xiaojun Jia, Tianyu Pang, Chao Du, Yihao Huang, Jindong Gu, Yang Liu, Xiaochun Cao, and Min Lin. Improved techniques for optimization-based jailbreaking on large language models. *arXiv preprint arXiv:2405.21018*, 2024.

Xiaojun Jia, Jie Liao, Qi Guo, Teng Ma, Simeng Qin, Ranjie Duan, Tianlin Li, Yihao Huang, Zhitao Zeng, Dongxian Wu, et al. Omnisafebench-mm: A unified benchmark and toolbox for multimodal jailbreak attack-defense evaluation. *arXiv preprint arXiv:2512.06589*, 2025.

Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences*, 103:102274, 2023.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

George Ling, Shanshan Zhong, and Richard Huang. Agent skills: A data-driven analysis of claude skills for extending large language model functionality. *arXiv preprint arXiv:2602.08004*, 2026.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, et al. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499*, 2023.

Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Zhang. Agent skills in the wild: An empirical study of security vulnerabilities at scale. *arXiv preprint arXiv:2601.10338*, 2026.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 1831–1847, 2024.

Zhiyuan Ma, Jiayu Liu, Xianzhen Luo, Zhenya Huang, Qingfu Zhu, and Wanxiang Che. Advancing tool-augmented large language models via meta-verification and reflection learning. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 2078–2089, 2025.

Andreas Motzfeldt, Joakim Edin, Casper L Christensen, Christian Hardmeier, Lars Maaløe, and Anna Rogers. Code like humans: A multi-agent solution for medical coding. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 22612–22627. Association for Computational Linguistics, 2025.

OpenAI. Codex CLI skills documentation. `https://developers.openai.com/codex/skills/`, 2025. Agent skills for Codex CLI using SKILL.md format in .codex/skills/ directory.

SafeAgent-Beihang. ClawGuard v3: Enterprise AI Agent Security Toolkit – SKILL.md Driven Active Defense. `https://github.com/SafeAgent-Beihang/clawguard`, 2026. GitHub repository, accessed April 25, 2026.

David Schmotz, Sahar Abdelnabi, and Maksym Andriushchenko. Agent skills enable a new class of realistic and trivially simple prompt injections. *arXiv preprint arXiv:2510.26328*, 2025.

David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. Skill-inject: Measuring agent vulnerability to skill file attacks. *arXiv preprint arXiv:2602.20156*, 2026.

Akashah Shabbir, Muhammad Akhtar Munir, Akshay Dudhane, Muhammad Umer Sheikh, Muhammad Haris Khan, Paolo Fraccaro, Juan Bernabe Moreno, Fahad Shahbaz Khan, and Salman Khan. Thinkgeo: Evaluating tool-augmented agents for remote sensing tasks. *arXiv preprint arXiv:2505.23752*, 2025.

SkillsMP. SkillsMP: Agent skills marketplace. `https://skillsmp.com`, 2025. Community-driven marketplace aggregating skills from public GitHub repositories; provides search, categorization, and quality indicators.

Skills.rest. Skills.rest: Agent skills registry. `https://skills.rest`, 2025. Community registry for agent skills with automated indexing from GitHub repositories.

SlowMist. Slowmist agent security skill. `https://github.com/slowmist/slowmist-agent-security`, 2026. GitHub repository, accessed April 25, 2026.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training llms to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*, 2024.

Le Wang, Zonghao Ying, Tianyuan Zhang, Siyuan Liang, Shengshan Hu, Mingchuan Zhang, Aishan Liu, and Xianglong Liu. Manipulating multimodal agents via cross-modal prompt injection. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 10955–10964, 2025a.

Xilong Wang, John Bloch, Zedian Shao, Yuepeng Hu, Shuyan Zhou, and Neil Zhenqiang Gong. Webinject: Prompt injection attack to web agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 2010–2030, 2025b.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026.

Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 1809–1820, 2025.

Sibo Yi, Yule Liu, Zhen Sun, Tianshuo Cong, Xinlei He, Jiaxing Song, Ke Xu, and Qi Li. Jailbreak attacks and defenses against large language models: A survey. *arXiv preprint arXiv:2407.04295*, 2024.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*, 2024.

Chenfei Zhu, Shao-Kang Hsia, Xiyun Hu, Ziyi Liu, Jingyu Shi, and Karthik Ramani. agentar: Creating augmented reality applications with tool-augmented llm-based autonomous agents. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–23, 2025.