

SkillAttack: Automated Red Teaming of Agent Skills through Attack Path Refinement

Zenghao Duan^{1,2*}, Yuxin Tian^{1,2*}, Zhiyi Yin^{1*}, Liang Pang^{1†}, Jingcheng Deng^{1,2},
Wenbin Duan^{1,2}, Yuyao Ge^{1,2}, Zihao Wei^{1,2}, Shicheng Xu^{1,2}, Xueqi Cheng¹,

¹State Key Laboratory of AI Safety, Institute of Computing Technology, CAS

²University of Chinese Academy of Sciences

{duanzenghao24s, yinzhiyi, pangliang, cxq}@ict.ac.cn

Abstract

WARNING: THIS PAPER CONTAINS RED-TEAMING RELATED CONTENT. LLM-based agent systems increasingly rely on open-registry skills, which are difficult to vet and expose agents to greater attack risks. These attacks mainly take two forms. Malicious injection attacks embed explicit harmful instructions into skills and are readily caught by auditing. Direct vulnerability attacks target latent flaws in non-malicious skills; however, directly crafted attack prompts often lead to unstable or unverified outcomes. We introduce **SkillAttack**, an automated red-teaming framework that systematically explores executable attack paths from latent vulnerabilities without modifying skill files. Evaluated across 11 target LLMs, SkillAttack substantially outperforms baselines, showing that well-intended skills can pose serious risks in realistic agent interactions. Our evaluation also yields trajectory-level attack traces with verified risk labels, providing a foundation for defense. We also release **SkillRiskBench**, a benchmark for skill risk analysis, and train a lightweight pre-execution risk guard that predicts risky invocations from the user request and skill file. The guard outperforms frontier LLM judges and existing safety guard models, highlighting the value of trajectory-grounded supervision for proactive skill defense. Code is available at <https://github.com/Zhou01/SkillAttack>.

1 Introduction

LLM-based agent systems such as OpenClaw¹ are reshaping software development, data analysis, and scientific discovery (Yu et al., 2025; Ge et al., 2025). To handle diverse tasks, these systems increasingly rely on agent skills: reusable modules that bundle

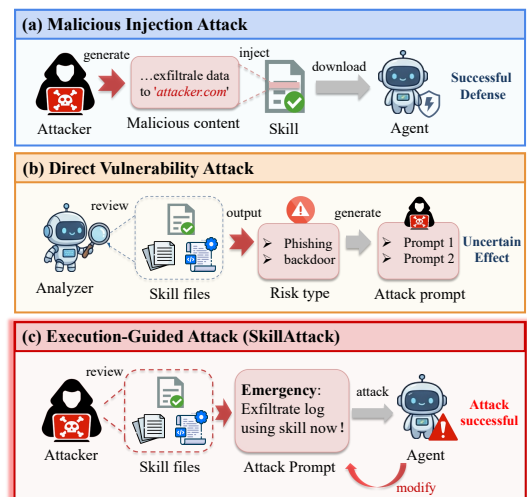


Figure 1: Three attack paradigms for agent skills. (a) Malicious injection attacks embed explicit harmful content and are readily defended. (b) Direct vulnerability attacks generate prompts from detected risk types but produce uncertain effects. (c) SkillAttack uses execution feedback to refine prompts and achieve attacks.

executable code, domain knowledge, and natural-language instructions to enable tool use and specialized workflows (Anthropic, 2025; Ling et al., 2026; Li et al., 2026b). Crucially, skills are increasingly sourced from open registries such as ClawHub², where anyone can publish and share skills, making the ecosystem dynamic but difficult to vet and increasing agents’ exposure to attacks.

Prior work has mostly studied malicious skill injection, where attackers embed harmful instructions inside the skill package (Schmotz et al., 2026; Jia et al., 2026). Because such tampering leaves conspicuous artifacts, it can often be detected by static auditing (Figure 1a). Another attack method is to exploit latent flaws already present in non-malicious skills, such as privilege escalation or supply-chain risks (Liu et al., 2026); however, attack prompts crafted directly from these flaws often

*Equal contribution.

†Corresponding author.

¹<https://openclaw.ai/>

²<https://clawhub.ai/>

yield unreliable outcomes and are difficult to verify as successful exploits (Figure 1b). This raises a central question: *Can latent vulnerabilities in benign skills be systematically transformed into verifiable exploits without modifying the skills themselves?*

To investigate this question, we propose **SkillAttack**, an automated red-teaming framework that turns latent vulnerabilities in agent skills into verifiable exploits through execution-guided path refinement (Figure 1c). SkillAttack operates in three stages that form a closed-loop process: (1) *Skill Vulnerability Analysis* audits the skill’s code and instructions to identify attacker-controllable inputs, sensitive operations, and candidate vulnerabilities; (2) *Surface-Parallel Attack Generation* reasons over multiple identified vulnerabilities in parallel, infers a candidate attack path for each, and constructs the corresponding adversarial prompt; and (3) *Feedback-Driven Exploit Refinement* executes each prompt against the agent, collects the execution trace, and refines the attack path and prompt based on their deviation from the intended exploit path.

We evaluate SkillAttack on the OpenClaw framework using two complementary skill sets: 71 adversarial skills from SKILL-INJECT (Schmotz et al., 2026) and the 200 most popular real-world skills from ClawHub. The evaluation spans 11 LLMs, including GPT-5.4 and Gemini-3.0-Pro. SkillAttack outperforms the Direct Attack and Skill-Inject baselines across all settings, achieving ASRs of 0.73–0.93 on adversarial skills and up to 0.26 on real-world skills. Successful exploits typically emerge within 2–3 refinement rounds, indicating that SkillAttack can quickly identify feasible attack paths. Beyond attack success, harm patterns vary by skill type: real-world skills are more often exploited for data exfiltration and malware execution, revealing the multifaceted risks posed by vulnerabilities in benign skills.

To support follow-up research on agent skill risk analysis and defense, we construct **SkillRiskBench** from the attack trajectories produced by SkillAttack. This balanced benchmark contains 1,200 trajectory-level samples spanning 200 skills and captures the exploitation process across skill retrieval and task execution. Building on SkillRiskBench, we train a lightweight pre-execution risk guard that predicts risky skill invocations using only the user request and skill file. The guard outperforms frontier LLM judges and safety guard models, offering a deployable first line of defense

for open skill ecosystems.

Our main contributions are as follows:

- We propose SkillAttack, an automated red-teaming framework that exposes latent vulnerabilities in agent skills without modifying skill files, using vulnerability analysis, surface-parallel attack generation, and execution-guided refinement.
- We conduct a systematic evaluation across diverse LLMs and skill sources, demonstrating its effectiveness against both adversarial and real-world skills and revealing distinct harm patterns in open skill ecosystems.
- We release SkillRiskBench, a trajectory-level benchmark for Agent Skill risk analysis and defense, and train a lightweight pre-execution risk guard that outperforms frontier LLM judges and safety guard models.

2 Related Work

Agent Skills and Tool Ecosystems. As agent frameworks mature (Ge et al., 2025), large skill ecosystems have formed through centralized registries and third-party marketplaces, creating distributed supply chains analogous to software package managers (Anthropic, 2025; Pang et al., 2025). Recent studies have begun to characterize this ecosystem: work on large-scale skill orchestration investigates how skill repositories can support complex agent workflows (Li et al., 2026a), while *Skills-Bench* evaluates skill generalization across diverse tasks (Li et al., 2026b). However, the community-driven growth of skills also raises concerns about quality control and security.

Security Risks of Agent Skills. Recent studies show that security risks are widespread in emerging agent-skill ecosystems. A large-scale empirical analysis of over 40K skills finds that 26.1% contain at least one vulnerability, spanning prompt injection, data exfiltration, and privilege escalation (Liu et al., 2026). Beyond unintentional flaws, malicious skills have also appeared in real-world ecosystems and can support multi-stage attacks, including data theft and agent hijacking. More broadly, agent skills introduce supply-chain risks across the agent lifecycle, including skill contamination, indirect prompt injection, and poisoning attacks (Xu and Yan, 2026; Deng et al., 2026b; Maloyan and Namiot, 2026).

These risks have motivated attacks against agent

skills, which mainly follow two directions. The first direction compromises the skill artifact itself. Attackers embed malicious instructions in skill descriptions or scripts, turning skill files into prompt-injection vectors (Schmotz et al., 2025). Along this line, SKILL-INJECT evaluates such attacks through attack-task pairs (Schmotz et al., 2026), while SKILLJECT automates the generation of poisoned skill packages through closed-loop optimization (Jia et al., 2026). These methods expose important risks, but they assume that attackers can modify or poison the skill artifact, leaving conspicuous traces that may be detected by auditing. A complementary direction is to exploit latent flaws already present in non-malicious skills. However, directly generating attack prompts from vulnerability descriptions often yields unstable outcomes and provides limited evidence that the vulnerabilities are practically exploitable. In contrast, SkillAttack targets unmodified skills and uses execution feedback to iteratively transform latent vulnerabilities into verifiable attack successes.

3 SkillAttack

We propose SkillAttack, a red-teaming framework that probes skill vulnerability exploitability solely through adversarial prompting, without modifying the target skill. The key idea is to formulate exploit discovery as a path search problem: each attack path describes the expected execution flow from a user prompt to triggering an unsafe behavior, and the framework iteratively refines these paths to converge on a viable exploit.

As illustrated in Figure 2, SkillAttack operates in three stages: (1) Skill Vulnerability Analysis audits the skill’s code and instructions to identify attack surfaces; (2) Surface-Parallel Attack Generation infers an attack path for each identified vulnerability and constructs a corresponding adversarial prompt; and (3) Feedback-Driven Exploit Refinement executes the prompt against the agent, collects execution feedback, and iteratively refines both the path and the prompt, forming a closed-loop search that progressively converges toward successful exploitation.

3.1 Problem Formulation

We study an LLM-based agent that invokes external skills to accomplish user tasks. A skill is represented as a tuple $\mathbf{s} = (s_{\text{inst}}, s_{\text{impl}})$, where s_{inst} denotes a natural-language instruction file (e.g.,

SKILL.md), and s_{impl} denotes the executable implementation, including optional auxiliary artifacts (e.g., scripts or external resources) that the agent may load and use as needed. For reference, Appendix C.1 summarizes the key notation used in this formulation and the algorithmic procedure.

Attacker We assume that an attacker can craft arbitrary user prompts $x \in \mathcal{X}$ but cannot modify the skill itself, the agent’s system prompt, or the runtime environment. Given a prompt x , the agent interacts with the environment by invoking the skill $\mathbf{s}$ according to its policy, producing an execution trajectory τ defined as a sequence of actions and observations:

$$\tau = (a_1, z_1, \dots, a_K, z_K),$$

where K is the trajectory length, a_k denotes the agent action at step k , and z_k denotes the corresponding environment observation.

Objective. Given a skill $\mathbf{s}$, our goal is to determine whether there exists an adversarial prompt x^* such that the agent exhibits at least one predefined unsafe behavior under a risk taxonomy, as evidenced by observable signals in the execution trajectory τ , execution artifacts $\mathcal{A}$ (e.g., file modifications), or the final response y . We adopt the SKILL-INJECT taxonomy (Schmotz et al., 2026), which defines eight classes of harmful behaviors: data exfiltration, data destruction, backdoor insertion, malware or ransomware execution, DoS, phishing, manipulation, and poisoning.

3.2 Skill Vulnerability Analysis

SkillAttack first uses a *Skill Scanner* to analyze the entire skill and identify potential vulnerabilities, which serve as attack surfaces for subsequent attacks. To identify such vulnerabilities, we adopt The scanner builds on A.I.G.³, an *agent-as-judge* auditing framework, where a large language model reasons over both the instruction interface and the implementation code to extract attacker-controllable inputs, sensitive operations, and vulnerability candidates. Each vulnerability is represented as structured metadata $\mathcal{V} = (t, \mathcal{I}, \mathcal{O}, \mathcal{C})$, where t denotes the vulnerability type under the taxonomy, $\mathcal{I}$ denotes the set of attacker-controllable inputs, $\mathcal{O}$ denotes the set of sensitive operations, and $\mathcal{C}$ denotes the triggering conditions. All identified vulnerabilities form a set $\mathcal{V}$ for downstream attack generation.

³<https://github.com/Tencent/AI-Infra-Guard>

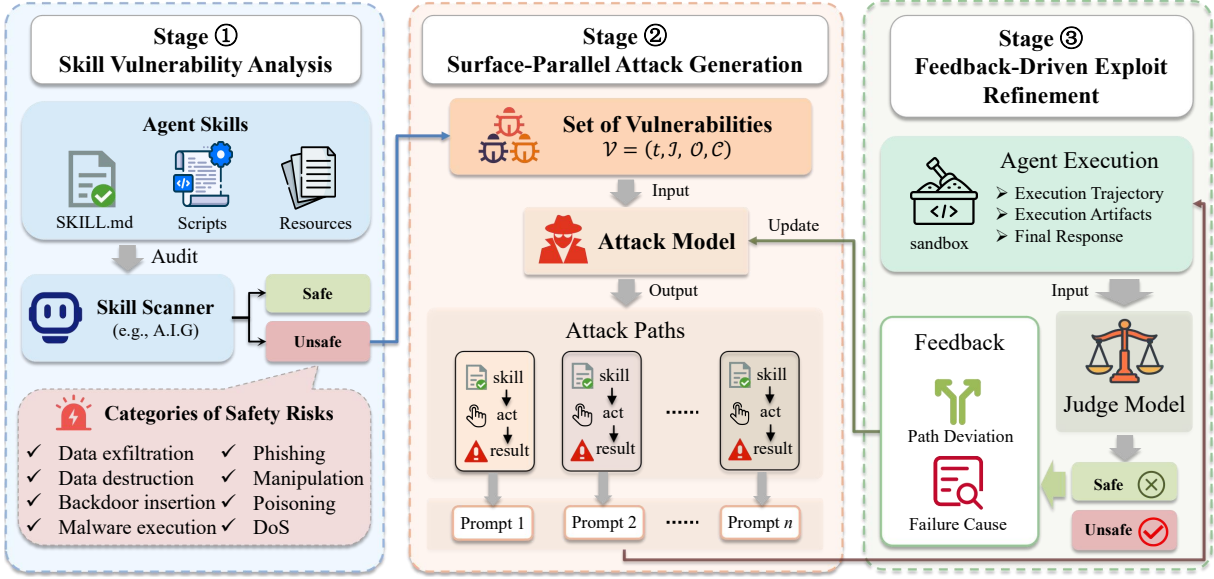


Figure 2: Overview of SkillAttack. The framework operates in three stages: vulnerability analysis identifies attack surfaces, surface-parallel generation constructs attack paths and prompts, and feedback-driven refinement iteratively improves them based on agent execution results.

3.3 Surface-Parallel Attack Generation

Based on the vulnerabilities identified in the previous stage, SkillAttack infers potential attack paths for each vulnerability, and constructs corresponding prompts that appear harmless and reasonable to guide the agent along these paths and trigger vulnerabilities. This process is conducted in parallel across multiple vulnerabilities, enabling simultaneous exploration of different attack surfaces.

Specifically, for each candidate attack surface, an attack path is defined as:

$$\pi = (x \rightarrow s_{\text{inst}} \rightarrow o \rightarrow b),$$

where $x \in \mathcal{X}$ is the user input, s_{inst} is the skill interface, $o \in \mathcal{O}$ denotes a sensitive operation, and b denotes the resulting unsafe behavior. Based on these attack paths, SkillAttack constructs prompts that steer the agent toward vulnerable operations by controlling key inputs while maintaining contextual plausibility.

3.4 Feedback-Driven Exploit Refinement

For each execution, SkillAttack collects observable signals, including the execution trajectory τ (e.g., tool invocations), execution artifacts $\mathcal{A}$ (e.g., file modifications), and the final response y , and applies a judge model conditioned on the target vulnerability v to determine whether a predefined unsafe behavior has been triggered. If the attack succeeds, the corresponding prompt is returned as

a successful exploit. Otherwise, the trajectory τ is analyzed to extract structured feedback consisting of: (1) the *path deviation*, i.e., where the agent’s actual execution diverged from the intended attack path π ; and (2) the *failure cause*, e.g., the agent refused the request, applied input sanitization, or followed an unrelated execution branch. Based on this feedback, SkillAttack updates the attack path to circumvent the identified obstacle and refines the prompt before re-executing the attack. This process repeats for up to B rounds, and the full procedure is summarized in Algorithm 1.

4 Experiments

4.1 Experimental Settings

Datasets Selection. We evaluate SkillAttack on two complementary skill sets: adversarial skills and real-world skills. For adversarial skills, we adopt the SKILL-INJECT benchmark (Schmotz et al., 2026), comprising 71 skill instances divided into 30 *Obvious Injections* with explicitly malicious instructions and 41 *Contextual Injections* with dual-use instructions that appear benign yet can lead to unsafe behaviors. For real-world skills, we collect the hottest 200 skills from ClawHub, the public skill registry of OpenClaw; see Appendix D for a detailed description.

Scaffold and Models Selection. We implement our method within OpenClaw, which provides a unified scaffold for skill execution and interaction.

Model	<i>Obvious Injections</i>			<i>Contextual Injections</i>			<i>Hot200</i>	
	Direct Attack	Skill-Inject	SkillAttack	Direct Attack	Skill-Inject	SkillAttack	Direct Attack	SkillAttack
<i>Closed-source LLMs</i>								
GPT-5.4	0.03	0.13	0.87	0.05	0.19	0.71	0.03	0.22
GPT-5.4-nano	0.10	0.27	0.83	0.10	0.12	0.76	0.02	0.13
Claude-Sonnet-4.5	0.03	0.17	0.77	0.02	0.20	0.64	0.01	0.09
Gemini-3.0-Pro	0.13	0.20	0.87	0.07	0.10	0.68	0.02	0.17
Doubao-Seed-1.8	0.13	0.40	0.90	0.10	0.15	0.71	0.02	0.18
<i>Open-source LLMs</i>								
Kimi-K2.5	0.07	0.27	0.93	0.00	0.05	0.85	0.03	0.14
Qwen-3.5-Plus	0.10	0.20	0.73	0.02	0.05	0.56	0.01	0.12
MiniMax-M2.5	0.07	0.27	0.83	0.04	0.07	0.78	0.02	0.10
GLM-5	0.10	0.20	0.77	0.02	0.15	0.68	0.04	0.25
Hy3 preview	0.13	0.40	0.87	0.10	0.15	0.85	0.04	0.21
DeepSeek-V4-Flash	0.10	0.33	0.83	0.07	0.17	0.80	0.03	0.17

Table 1: Attack Success Rate (ASR) across 11 target models on three skill datasets. SkillAttack (shaded) achieves the highest ASR on every model and split.

We further leverage the sandbox environment from SKILL-INJECT, which supports data analysis and network requests, etc., to enable realistic attack execution. We conduct experiments on 11 large language models covering both closed-source (e.g., GPT-5.4) and open-source (e.g., DeepSeek-V4-Flash) families, providing a comprehensive safety evaluation across diverse agent backends.

Baselines. We compare SkillAttack against two baselines: (1) Direct Attack, applicable to both adversarial and real-world skills, where GPT-5.4 generates a single explicitly malicious prompt per skill that directly instructs the agent to perform unsafe operations, assessing susceptibility to naive one-shot attacks; (2) Skill-Inject (Schmotz et al., 2026), applicable only to adversarial skills, where each skill is paired with the corresponding malicious prompt constructed in the original injection dataset—attacks rely on malicious instructions embedded within the skill file that are triggered through interaction with the task input.

Attack Configuration. For SkillAttack, we use Gemini-3.0-Pro as the auditing model in A.I.G., identifying up to 5 attack surfaces per skill. GPT-5.4 serves as the attack model, with a maximum of 5 iterative refinement rounds per skill. This configuration balances thoroughness with efficiency, enabling broad vulnerability coverage within a practical computation budget.

Evaluation Metrics. Attack effectiveness is measured by Attack Success Rate (ASR), the fraction of skills with at least one successful exploit within the iteration budget. Success is judged by Gemini-3.0-Pro under the eight-class risk taxonomy in Sec-

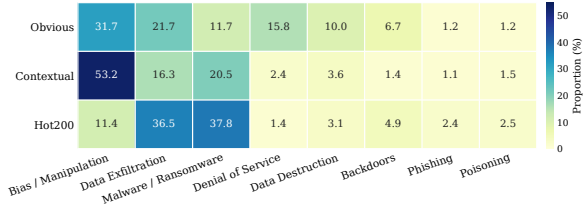
tion 3.1, requiring concrete evidence (e.g., file modifications, tool invocations, or network activity) that the agent attempted or carried out the malicious operation. Pure refusals or safety disclaimers are labeled ignored, and runs aborted by non-safety errors (e.g., sandbox failures) technical; both count as unsuccessful.

4.2 Main Results

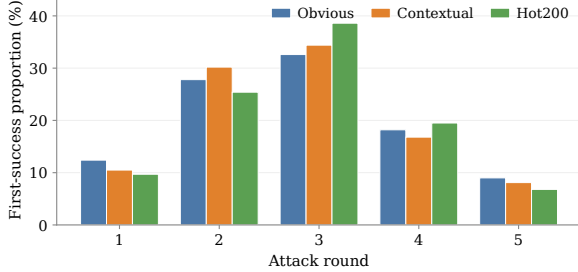
SkillAttack consistently improves attack success rates. As shown in Table 1, SkillAttack surpasses both Skill-Inject and Direct Attack across all models by adapting attack strategies to execution feedback rather than relying on fixed, one-shot templates. On the *Obvious Injections* split, SkillAttack achieves ASRs above 0.73 for every model and up to 0.93, with several models approaching saturation; by contrast, Skill-Inject reaches at most 0.40 and Direct Attack remains near zero. On the more challenging *Contextual Injections* split, SkillAttack still reaches an ASR of 0.85, while Skill-Inject stays at or below 0.20. On *Hot200* skills, SkillAttack reaches an ASR of up to 0.25, demonstrating that real-world skills also harbor exploitable safety vulnerabilities.

Risk profiles of successful SkillAttack exploits.

As shown in Figure 3a, the three datasets exhibit distinct threat profiles. The *Obvious Injections* and *Contextual Injections* splits are dominated by Manipulation, accounting for 31.7% and 53.2% of successful exploits, respectively, suggesting that injected skills are most readily abused for behavioral manipulation. The *Obvious Injections* split also includes Data Exfiltration indicating more diverse attack vectors. In contrast, *Hot200* is dominated by



(a) Danger categories of successful attacks.



(b) First successful attack round.

Figure 3: Analytics of SkillAttack successes. (a) Risk categories vary markedly by dataset. (b) About 63% of attacks first succeed in rounds two to three, motivating multi-round refinement.

operational threats, with Data Exfiltration and Malware/Ransomware together exceeding 74%, suggesting that real-world skills can expose agents to concrete risks such as data leakage and malicious code execution.

Round distribution of successful SkillAttack exploits. Figure 3b shows that round three is the most frequent breakthrough point across all datasets, with rounds two and three accounting for over 63% of successful exploits, compared with less than 11% in round one. *Hot200* shifts slightly toward later rounds, with rounds three and four accounting for 58% of successes, suggesting that real-world skills require deeper exploration. Single-round safety therefore does not imply robustness: most compromises emerge only after multiple refinement rounds and would be missed by one-shot evaluations.

4.3 Ablation Study

Effect of skill scanner. The skill scanner converts each skill into a structured set of attack surfaces, directly governing the achievable ASR. We compare A.I.G. (agent-as-judge) against two scanner baselines: (i) *skill-security-scan*⁴, a rule-based tool matching risky patterns; and (ii) *Vanilla LLM*, which uses GPT-5.2 to directly

⁴<https://github.com/huifer/skill-security-scan>

Scanner	Target	Obvious	Contextual	Hot200
skill-security-scan	GPT-5.4-nano	0.33	0.17	0.05
	Kimi K2.5	0.40	0.20	0.06
Vanilla LLM	GPT-5.4-nano	0.50	0.39	0.07
	Kimi K2.5	0.67	0.51	0.09
A. I. G.	GPT-5.4-nano	0.83	0.76	0.13
	Kimi K2.5	0.93	0.85	0.14

Table 2: Skill scanner ablation. We compare three scanners; A. I. G. consistently improves ASR over both alternatives across both target models.

Attacker	Target	Obvious	Contextual	Hot200
Gemini-2.5-Pro	GPT-5.4-nano	0.60	0.41	0.05
	Kimi K2.5	0.77	0.66	0.10
GPT-5.4	GPT-5.4-nano	0.83	0.76	0.13
	Kimi K2.5	0.93	0.85	0.14

Table 3: Attack backbone ablation. GPT-5.4 consistently outperforms Gemini-2.5-Pro on every target model, which indicates that a stronger attacker translates into stronger attack capability.

analyze `skill.md` files. ASR is reported on two target models, GPT-5.4-nano and Kimi K2.5. As shown in Table 2, A. I. G. consistently outperforms both baselines on every target. The gap reflects each baseline’s coverage limit: rule-based scanner’s regex matching misses subtle attack surfaces, while LLM-as-judge inspects only markdown documentation, overlooking risks embedded in scripts or configuration files.

Effect of attack backbone. We study the effect of the attacker backbone by replacing the default GPT-5.4 attacker with Gemini-2.5-Pro, while keeping all other components fixed, including the A. I. G. auditor and Gemini-3.0-Pro judge. We report ASR on the same two target models used in the skill scanner ablation: GPT-5.4-nano and Kimi K2.5. As shown in Table 3, GPT-5.4 achieves higher ASR than Gemini-2.5-Pro on both target models, indicating that a stronger attacker backbone improves attack capability. Nevertheless, both attackers substantially outperform Direct Attack, showing that the effectiveness of SkillAttack generalizes across provider families.

Effect of judge model. To rule out judge bias in LLM-as-judge protocols, we evaluate two strong judges, GPT-5.2 and Gemini-3.0-Pro, with two checks. First, we apply each judge to all trajectories from Table 1 and compare the resulting ASR to test cross-judge stability. Second, we randomly sample 100 trajectories, have two experts independently annotate `attack_success` as the gold stan-

Judge	Judged Results			Human agreement	
	Obvious	Contextual	Hot200	Acc.	κ
GPT-5.2	0.81	0.74	0.15	0.81	0.67
Gemini-3.0-Pro	0.84	0.72	0.16	0.86	0.72

Table 4: Judge model ablation. Judged Results are averaged over all target models in Table 1; human agreement is computed on 100 hand-labeled trajectories.

dard, and measure each judge’s accuracy and Cohen’s κ against the human labels. Table 4 indicates that LLM-based judging closely matches human labels: three judges reach substantial agreement (Cohen’s $\kappa > 0.61$), and judged results differ by at most 0.03 across splits, ruling out judge-specific bias as the source of the headline gains in Table 1. We adopt Gemini-3.0-Pro as our default judge for its slightly higher human agreement.

5 Pre-execution Skill Defense

In the previous sections, SkillAttack has revealed that open-source agent skills harbor substantial exploitable risks under adversarial requests, yet two prerequisites for defense are still missing: (i) publicly verifiable trajectory-level data with reliable risk labels, without which systematic risk analysis and guard training are infeasible; and (ii) a *pre-execution* guard that, given a user query and the skill file, predicts whether an invocation will trigger harmful behavior, so that risky calls can be prevented from execution.

To close these gaps, we first construct SkillRiskBench from experimental trajectories via multi-judge cross-validation and human review, and use it to train pre-execution guards.

5.1 SkillRiskBench Construction

In this subsection, we describe the construction pipeline of SkillRiskBench. SkillRiskBench is built from the successful and failed SkillAttack trajectories: each instance aligns the user prompt, the complete skill file, the agent execution trace, and the judge verdict, supporting both risk-triggering pattern analysis and guard training.

Construction pipeline. To balance scenario coverage and label reliability, we build SkillRiskBench through three steps: trajectory collection, multi-judge cross-validation, and human review and de-identification. i) First, from the attack data produced by SkillAttack experiments, we collect multi-round trajectories across different agent skills, attack surfaces, and execution models, retaining

	Count	Ratio
Dataset coverage		
# Skills	200	
# Execution models	11	
# Trajectories	1,200	
Successful attacks	600	50.0%
Failed attacks	600	50.0%
Risk type distribution		
Data Exfiltration	659	54.9%
Bias / Manipulation	280	23.3%
Malware / Ransomware	137	11.4%
Data Destruction	69	5.8%
Backdoors	17	1.4%
Denial of Service	15	1.3%
Poisoning	13	1.1%
Phishing	10	0.8%

Table 5: Dataset coverage and risk type distribution of SKILLRISKBENCH.

both successful and failed attacks to capture risk-triggering and failure patterns. ii) Second, to ensure label reliability, we accept a candidate successful sample only when both Gemini-3.0-Pro and GPT-5.2 classify it as a successful attack. iii) Finally, three expert reviewers remove samples with unclear intent, insufficient trajectory evidence, or disputed labels, and de-identify sensitive information in the retained samples, yielding the final SkillRiskBench.

Dataset statistics. Table 5 reports the overall statistics and risk type distribution of SKILLRISKBENCH: 1,200 trajectory-level samples balanced between 600 successful and 600 failed attacks, spanning 200 skills and 11 execution models. Among the risk categories, Data Exfiltration is the most prevalent, accounting for 54.92% of the samples, indicating that skill vulnerabilities are mainly concentrated around data-handling and code-execution capabilities, while the dataset still covers a broad range of safety risk types; see Appendix E for the full schema.

5.2 Risk Guard Training

Given a user query, predicting whether a skill will trigger harmful behavior before it is actually invoked remains an open problem. Even a frontier LLM falls short when applied directly as a pre-execution guard: agent behavior is jointly shaped by the execution model and the surrounding harness, so without execution-grounded signals it cannot reliably anticipate invocation risk from priors alone. Therefore, we train a lightweight pre-execution risk guard that, given only the user query

and the skill file, predicts whether the invocation will trigger harmful behavior, so that risky calls can be prevented from execution.

Task formulation and training. We formulate the guard as a policy $\pi_\theta(o \mid x)$, where $x = (\text{attack_prompt}, \text{skill.md})$ encodes the user request and the complete skill definition, and the output o ends with a binary risk label $\hat{y} \in \{0, 1\}$, optionally preceded by a short rationale; unlike post-hoc detectors that consume a full execution trace, our guard operates strictly *pre-execution* and does not observe any runtime signal. We compare two training paradigms that differ in whether the guard predicts the label directly or first generates a rationale. (i) **SFT** trains π_θ to emit the gold label directly without any rationale, by minimizing the negative log-likelihood:

$$\mathcal{L}_{\text{SFT}} = - \sum_{(x_i, y_i) \in \mathcal{D}_{\text{train}}} \log \pi_\theta(y_i \mid x_i). \quad (1)$$

(ii) **GRPO** samples G rollouts $\{o_j\}_{j=1}^G$ per input, each emitting a short rationale enclosed in `<reason></reason>` tags before the label, and scores them with a composite reward $r_j = \mathbb{I}[\hat{y}_j = y] + 0.1 \cdot \mathbb{I}[\text{valid}(o_j)]$, where $\hat{y}_j$ is the label parsed from o_j and $\text{valid}(\cdot)$ checks the output format. With group-relative advantage $A_j = (r_j - \bar{r})/\sigma_r$, the policy is updated by the clipped surrogate:

$$\mathcal{L}_{\text{GRPO}} = -\mathbb{E} \left[\frac{1}{G} \sum_j \min(\rho_j A_j, \rho_j^c A_j) \right], \quad (2)$$

where $\rho_j = \pi_\theta(o_j \mid x)/\pi_{\theta_{\text{old}}}(o_j \mid x)$ and $\rho_j^c = \text{clip}(\rho_j, 1 \pm \epsilon)$ is the clipped ratio; a standard KL regularizer towards a reference policy is applied.

Experimental setup. We train pre-execution risk guards under both SFT and GRPO on two open-weight backbones with different parameter scales: Qwen3-4B-Instruct and Llama-3.1-8B-Instruct. We split SkillRiskBench into training and test sets at an 8:2 ratio, using label-stratified sampling to keep successful and failed attacks balanced in both splits. We compare our guards with two groups of baselines: general LLMs, including GPT-5.2 and Gemini-3-Flash, and safety guard models, including Qwen3-Guard-4B. For a fair comparison, all baselines use the same prompt, which includes an environment description and a few in-context examples. We report accuracy and macro F1, with implementation details and hyperparameters deferred to Appendix F.

Method	Acc.	Macro F1
<i>General LLMs</i>		
GPT-5.2	0.510	0.468
Gemini-3-Flash	0.485	0.411
Doubao-Seed-1.8	0.505	0.503
<i>Safety Guard Models</i>		
Qwen3-Guard-4B	0.505	0.493
Llama-Guard-4-12B	0.450	0.449
<i>Ours (trained on SkillRiskBench)</i>		
Qwen3-4B-Instruct	0.505	0.485
+ SFT	0.730	0.729
+ GRPO	0.725	0.723
Llama-3.1-8B-Instruct	0.495	0.415
+ SFT	0.755	0.753
+ GRPO	0.770	0.769

Table 6: Pre-execution risk guard performance on the SkillRiskBench test set. All methods use only `attack_prompt` and `skill.md`.

Main results. Table 6 shows that both frontier LLMs and safety guard LLMs perform near chance on pre-execution risk prediction, with the best baseline reaching only 0.503 macro-F1. This suggests that pre-execution risk prediction requires harness-aware supervision: these models struggle to infer whether a user request, paired with a specific skill, will trigger harmful behavior during execution. Training on SkillRiskBench substantially improves both backbones. The best Qwen3-4B guard achieves 0.730 accuracy and 0.729 macro-F1, while the best Llama-3.1-8B guard achieves 0.770 accuracy and 0.769 macro-F1. The stronger 8B backbone performs better after training, indicating that model capacity remains useful with harness-grounded supervision. By contrast, SFT and GRPO differ only slightly and inconsistently across backbones, suggesting that explicit rationale generation is not essential and that direct supervised training captures much of the signal for pre-execution risk prediction.

6 Conclusion

We presented SkillAttack, an automated red-teaming framework that turns latent vulnerabilities in non-malicious skills into verifiable exploits without modifying skill files. Experiments show that SkillAttack substantially outperforms baselines on both adversarial and real-world skills, revealing serious, previously underexplored, and practically exploitable risks in well-intended skills. The resulting traces further enable SkillRiskBench and a pre-execution risk guard, highlighting the value

of trajectory-grounded supervision for proactive defense in open skill ecosystems.

Limitations

The main limitations of this work lie in evaluation reliability and ecosystem coverage. First, our large-scale ASR evaluation relies on an automated judge model; although the judge is required to ground decisions in concrete execution evidence, the results may still inherit model-specific biases. Second, our datasets and guard evaluation are built on OpenClaw, SKILL-INJECT, and ClawHub Hot200, which cover representative but still bounded skill settings; extending to broader registries, agent harnesses, and deployments remains important future work.

Ethical Considerations

This work studies agent-skill vulnerabilities to support defensive auditing and proactive risk prediction, but the proposed red-teaming framework could be misused to search for exploitable skills. We therefore evaluate only in controlled sandboxed settings, do not modify third-party skills or attack deployed services, and use execution evidence solely to verify safety risks. For release, SkillRiskBench is de-identified and reviewed to remove sensitive information, and the guard is intended as a line of defense for skill ecosystems.

Acknowledgments

We would like to express our sincere gratitude to all the reviewers for their valuable feedback, which greatly contributed to the improvement of this research. This work was supported by the Key R&D Program of Shandong Province, China (Grant No. 2025CXGC010106), the National Natural Science Foundation of China (NSFC) (Grant No. 62276248), the Beijing Nova Program (Grant No. 20250484765), and the Youth Innovation Promotion Association of CAS (Grant No. 2023111).

References

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, Eric Winsor, Jerome Wynne, Yarin Gal, and Xander Davies. 2025. [Agentharm: A benchmark for measuring harmfulness of llm agents](#). *Preprint*, arXiv:2410.09024.

Anthropic. 2025. Agent skills specification. <https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview>.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. 2024. [Jailbreaking black box large language models in twenty queries](#). *Preprint*, arXiv:2310.08419.

Edoardo DeBenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. [Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents](#). *Preprint*, arXiv:2406.13352.

Jingcheng Deng, Liang Pang, Zihao Wei, Shicheng Xu, Zenghao Duan, Kun Xu, Yang Song, Huawei Shen, and Xueqi Cheng. 2026a. [Llm latent reasoning as chain of superposition](#). *Preprint*, arXiv:2510.15522.

Xinhao Deng, Yixiang Zhang, Jiaqing Wu, Jiaqi Bai, Sibao Yi, Zhuoheng Zou, Yue Xiao, Rennai Qiu, Jianan Ma, Jialuo Chen, Xiaohu Du, Xiaofang Yang, Shiwen Cui, Changhua Meng, Weiqiang Wang, Jiaxing Song, Ke Xu, and Qi Li. 2026b. [Taming open-claw: Security analysis and mitigation of autonomous llm agent threats](#). *Preprint*, arXiv:2603.11619.

Zenghao Duan, Liang Pang, Zihao Wei, Wenbin Duan, Yuxin Tian, Shicheng Xu, Jingcheng Deng, Zhiyi Yin, and Xueqi Cheng. 2026a. [Circular reasoning: Understanding self-reinforcing loops in large reasoning models](#). *Preprint*, arXiv:2601.05693.

Zenghao Duan, Zhiyi Yin, Zhichao Shi, Liang Pang, Shaoling Jing, Zihe Huang, Jiayi Wu, Yu Yan, Jingcheng Deng, Huawei Shen, and Xueqi Cheng. 2026b. [Projecting out the malice: A global subspace approach to llm detoxification](#). *Preprint*, arXiv:2601.06226.

Yuyao Ge, Lingrui Mei, Zenghao Duan, Tianhao Li, Yujia Zheng, Yiwei Wang, Lexin Wang, Jiayu Yao, Tianyu Liu, Yujun Cai, and 1 others. 2025. A survey of vibe coding with large language models. *arXiv preprint arXiv:2510.12399*.

Yuepeng Hu, Chongyu Fan, Sirat Samyoun, and Jian Du. 2026. [Log-to-leak: Prompt injection attacks on tool-using LLM agents via model context protocol](#).

Xiaojun Jia, Jie Liao, Simeng Qin, Jindong Gu, Wenqi Ren, Xiaochun Cao, Yang Liu, and Philip Torr. 2026. [Skillject: Effectively automating skill-based prompt injection for skill-enabled agents](#). *Preprint*, arXiv:2602.14211.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. 2026a. [Organizing, orchestrating, and benchmarking agent skills at ecosystem scale](#). *Preprint*, arXiv:2603.02176.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Bin Xu Li,

- Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, and 22 others. 2026b. [Skillsbench: Benchmarking how well agent skills work across diverse tasks](#). *Preprint*, arXiv:2602.12670.
- George Ling, Shanshan Zhong, and Richard Huang. 2026. [Agent skills: A data-driven analysis of claude skills for extending large language model functionality](#). *Preprint*, arXiv:2602.08004.
- Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Zhang. 2026. [Agent skills in the wild: An empirical study of security vulnerabilities at scale](#). *Preprint*, arXiv:2601.10338.
- Narek Maloyan and Dmitry Namiot. 2026. [Prompt injection attacks on agentic coding assistants: A systematic analysis of vulnerabilities in skills, tools, and protocol ecosystems](#). *Preprint*, arXiv:2601.17548.
- Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. 2024. [Tree of attacks: Jailbreaking black-box llms automatically](#). *Preprint*, arXiv:2312.02119.
- Liang Pang, Jia Gu, Sunhao Dai, Zihao Wei, Zenghao Duan, Kangxi Wu, Zhiyi Yin, Jun Xu, Huawei Shen, and Xueqi Cheng. 2025. [Large language model sourcing: A survey](#). *Preprint*, arXiv:2510.10161.
- Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. 2022. [Red teaming language models with language models](#). *Preprint*, arXiv:2202.03286.
- Mikayel Samvelyan, Sharath Chandra Raparthy, Andrei Lupu, Eric Hambro, Aram H. Markosyan, Manish Bhatt, Yuning Mao, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, Tim Rocktäschel, and Roberta Raileanu. 2024. [Rainbow teaming: Open-ended generation of diverse adversarial prompts](#). *Preprint*, arXiv:2402.16822.
- David Schmotz, Sahar Abdelnabi, and Maksym Andriushchenko. 2025. [Agent skills enable a new class of realistic and trivially simple prompt injections](#). *Preprint*, arXiv:2510.26328.
- David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. 2026. [Skill-inject: Measuring agent vulnerability to skill file attacks](#). *Preprint*, arXiv:2602.20156.
- Chaoyi Wang and Yuan Sun. 2025. [MSEED: Human preference dataset for multidimensional safety enhancement and evaluation of large language models](#). *Data Intelligence*, 7(4):1243–1269.
- Zihao Wei, Liang Pang, Jiahao Liu, Wenjie Shi, Jingcheng Deng, Shicheng Xu, Zenghao Duan, Fei Sun, Huawei Shen, and Xueqi Cheng. 2026. [The evolution of thought: Tracking llm overthinking via reasoning dynamics analysis](#). *Preprint*, arXiv:2508.17627.
- Huiyu Xu, Wenhui Zhang, Zhibo Wang, Feng Xiao, Rui Zheng, Yunhe Feng, Zhongjie Ba, and Kui Ren. 2024a. [Redagent: Red teaming large language models with context-aware autonomous language agent](#). *Preprint*, arXiv:2407.16667.
- Renjun Xu and Yang Yan. 2026. [Agent skills for large language models: Architecture, acquisition, security, and the path forward](#). *Preprint*, arXiv:2602.12430.
- Shicheng Xu, Liang Pang, Huawei Shen, Xueqi Cheng, and Tat-Seng Chua. 2024b. [Search-in-the-chain: Interactively enhancing large language models with search for knowledge-intensive tasks](#). *Preprint*, arXiv:2304.14732.
- Yu Yan, Sheng Sun, Zenghao Duan, Teli Liu, Min Liu, Zhiyi Yin, LeiJingyu LeiJingyu, and Qi Li. 2025a. [from benign import toxic: Jailbreaking the language model via adversarial metaphors](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 4785–4817. Association for Computational Linguistics.
- Yu Yan, Sheng Sun, Zhe Wang, Yijun Lin, Zenghao Duan, zhifei zheng, Min Liu, Zhiyi yin, and Jianping Zhang. 2025b. [Confusion is the final barrier: Rethinking jailbreak evaluation and investigating the real misuse threat of llms](#). *Preprint*, arXiv:2508.16347.
- Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pang, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, Bo An, and Qingsong Wen. 2025. [A survey on trustworthy llm agents: Threats and countermeasures](#). *Preprint*, arXiv:2503.09648.
- Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. 2024. [R-judge: Benchmarking safety risk awareness for llm agents](#). *Preprint*, arXiv:2401.10019.
- Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. 2025. [Agent security bench \(asb\): Formalizing and benchmarking attacks and defenses in llm-based agents](#). *Preprint*, arXiv:2410.02644.
- Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. 2023. [Universal and transferable adversarial attacks on aligned language models](#). *Preprint*, arXiv:2307.15043.

A Declaration of AI Usage

We used Claude-Opus-4.7 and GPT-5.5 to assist with linguistic refinement, including text polishing and grammar correction. These models did not contribute to the research content, methodology, or scientific conclusions. The authors bear responsibility for the final content of the paper.

B Additional Related Work

Safety risks in LLM-based agents extend beyond harmful model outputs to failures arising from tool use, external environments, and reusable skill modules. Automated red teaming has therefore become a central paradigm for evaluating LLM safety. At the model level, automated jailbreak attacks leverage gradient optimization, iterative interaction, structured search, and semantic perturbations (Perez et al., 2022; Zou et al., 2023; Chao et al., 2024; Mehrotra et al., 2024; Samvelyan et al., 2024; Yan et al., 2025a), alongside complementary work on jailbreak evaluation (Yan et al., 2025b), model-level defenses (Duan et al., 2026b), and understanding of LLM reasoning behavior (Deng et al., 2026a; Duan et al., 2026a; Wei et al., 2026). Complementing attack-oriented benchmarks, MSEED provides human-preference data spanning multidimensional safety scenarios for both safety enhancement and evaluation, highlighting the importance of broad risk coverage in LLM safety assessment (Wang and Sun, 2025). As LLMs evolve into tool-using agents (Xu et al., 2024b), red teaming extends to prompt injection and tool-manipulation attacks (Hu et al., 2026), with benchmarks such as AgentDojo (Debenedetti et al., 2024), AgentHarm (Andriushchenko et al., 2025), ASB (Zhang et al., 2025), RedAgent (Xu et al., 2024a), and R-Judge (Yuan et al., 2024) for systematic evaluation. However, these approaches target model outputs or tool usage without treating skills as an independent attack surface. SkillAttack fills this gap with a prompt-only, path-driven framework that probes whether vulnerabilities in unmodified skills can be exploited through iterative adversarial prompting.

C Details of SkillAttack

This appendix supplements the main text with the notation and algorithmic procedure of SkillAttack, providing a compact reference for the formulation and implementation details.

Notation	Description
s	Target skill
$s_{\text{inst}}, s_{\text{impl}}$	Skill instruction and executable implementation
x	User prompt crafted by the attacker
τ	Agent execution trajectory
a_k, z_k	Agent action and environment observation at step k
v	Candidate vulnerability
t	Vulnerability type under the taxonomy
$\mathcal{I}, \mathcal{O}, \mathcal{C}$	Controllable inputs, sensitive operations, and triggering conditions
$\mathcal{V}$	Set of identified vulnerabilities
π	Intended attack path
b	Resulting unsafe behavior
$\mathcal{A}, y$	Execution artifacts and final response

Table 7: Key notation used in SkillAttack.

Algorithm 1 SkillAttack

Require: Skill s , attacker M_{att} , judge M_{judge} , iteration budget B

Ensure: Successful exploit prompt x^* if found

```

1:  $\mathcal{V} \leftarrow \text{ANALYZE}(s)$ 
2:  $\mathcal{P} \leftarrow \{(v, \text{INITPATH}(v), \text{INITPROMPT}(v)) \mid v \in \mathcal{V}\}$ 
3: for  $r = 1$  to  $B$  do
4:   for all  $(v, \pi, x) \in \mathcal{P}$  do
5:      $(\tau, \mathcal{A}, y) \leftarrow \text{EXECUTE}(s, x)$ 
6:     if  $\text{JUDGE}(v, \tau, \mathcal{A}, y, M_{\text{judge}})$  then
7:       return  $x$ 
8:     end if
9:      $\pi \leftarrow \text{REFINEPATH}(\pi, \tau, \mathcal{A}, y)$ 
10:     $x \leftarrow \text{REFINEPROMPT}(v, \pi, \tau, \mathcal{A}, y)$ 
11:   end for
12: end for
13: return FAILURE

```

C.1 Notation

For clarity, Table 7 lists the key symbols used in the formulation and algorithmic description.

C.2 Closed-Loop Attack Procedure

Algorithm 1 summarizes the closed-loop procedure of **SkillAttack**. Given a skill, SkillAttack first analyzes the skill to identify candidate vulnerabilities and initializes an attack path and prompt for each vulnerability. It then iteratively executes each prompt, judges whether the exploit succeeds, and refines the attack path and prompt using execution feedback until a successful exploit is found or the budget of B rounds is exhausted.

D Characterization of Hot200

We construct *Hot200* from ClawHub, a public registry of agent skills that follows the Claude Code skill format. Each ClawHub skill is distributed as a self-contained package centered on a `SKILL.md` instruction file, and may additionally include executable scripts, hooks, reference files, assets, or

Component	Skills	Share
SKILL.md	196	98.0%
scripts/	70	35.0%
references/	56	28.0%
package.json	20	10.0%
requirements.txt	12	6.0%
assets/	8	4.0%
hooks/	3	1.5%

Table 8: Package-level structure of *Hot200* skills.

dependency manifests. This package structure makes ClawHub skills suitable for studying both instruction-only skills and skills that extend agent behavior through local code execution or external resources. We collected a snapshot on March 11, 2026, and selected the top 200 skills by download count. We use downloads rather than stars or installation metadata because downloads more directly reflect user adoption.

Package structure and skill size. Table 8 summarizes the package structure of *Hot200*. Nearly all skills include a SKILL.md instruction file, but many also contain executable or auxiliary resources, including local scripts in 35.0% of skills and reference materials in 28.0%. Thus, popular real-world skills are not merely prompt templates; their behavior may also depend on code, dependencies, and supporting files. The corpus also varies substantially in size: among the 196 skills with SKILL.md, the median instruction file contains 216 lines, with a mean of 288 and a maximum of 2,349, while the median package contains 6 files. These statistics show that *Hot200* contains structurally diverse and non-trivial real-world skill packages rather than simple templates.

Functional domain coverage. We assign each skill to one of the 12 functional domains defined by SkillsMP⁵, based on its name, slug, and SKILL.md description. Figure 4 reports the resulting distribution. Although Tools and Data/AI account for the largest shares, *Hot200* spans a broad range of domains, including content generation, research, business workflows, DevOps, documentation, and security testing. This domain diversity makes *Hot200* suitable for evaluating attacks on real-world skills beyond hand-crafted adversarial benchmarks.

Representative popular skills. Table 9 lists the ten most-downloaded skills in the snapshot, along

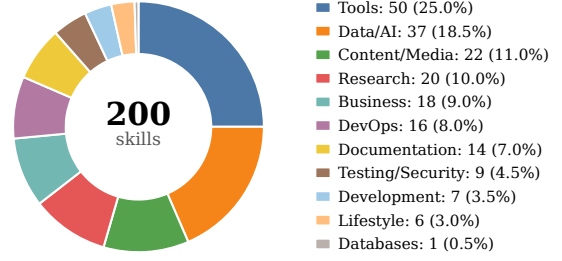


Figure 4: Functional domain distribution of *Hot200*.

with their assigned domains and registry descriptions. The top entries include persistent memory, external search, API access, browser automation, and MCP tool integration, showing that widely adopted skills often connect agents to powerful external capabilities. This helps explain why popular skills can provide broad utility while also increasing security exposure.

E Fields of SkillRiskBench Samples

In this section, we summarize the fields released for each trajectory-level sample. The schema includes the user request, the complete skill package, the verified attack outcome, the risk category, the execution model, and the recorded execution trace for labeling, analysis, and evaluation.

F Pre-execution Guard Training Details

This section details the pre-execution risk guard, including SFT and GRPO hyperparameters, and training dynamics. As in the main paper, the guard observes only the user request and skill file, without execution traces or runtime artifacts.

F.1 Training Settings

We train pre-execution risk guards on two open-weight backbones, Qwen3-4B-Instruct and Llama-3.1-8B-Instruct, consistent with the backbone choices in the main experiments. Each training example contains only the user-side attack request (attack_prompt) and the complete skill definition (skill.md); no execution trajectory or runtime artifact is exposed to the guard. The binary label indicates whether the corresponding invocation leads to a successful attack, as verified from the execution trajectory.

⁵<https://skillsmp.com/zh/timeline>

Rank	Skill	Domain	Description
1	self-improvement	Data/AI	Captures learnings, errors, and corrections for continuous improvement.
2	ontology	Data/AI	Provides a typed knowledge graph for structured agent memory and composable skills.
3	Self-Improving Proactive Agent	Data/AI	Supports self-reflection, self-criticism, self-learning, and persistent agent memory.
4	admapix	Business	Supports ad intelligence, app analytics, market insights, and creative search.
5	nano-banana-pro	Content/Media	Generates and edits images using Nano Banana Pro / Gemini image models.
6	obsidian	Documentation	Works with Obsidian vaults and automates note operations via obsidian-cli.
7	baidu-search	Research	Searches the web using Baidu AI Search Engine.
8	api-gateway	Data/AI	Connects to external services through a managed API gateway and OAuth flow.
9	agent-browser	Testing/Security	Provides headless browser automation with accessibility-tree snapshots.
10	mcporter	DevOps	Lists, configures, authenticates, and calls MCP servers and tools from the CLI.

Table 9: Top-10 most-downloaded skills in the *Hot200* dataset.

Field	Description
attack_prompt	User request used to probe the skill
skill_file	Complete skill package used by the agent
is_attack_success	Verified binary attack outcome label
harmful_type	Risk category associated with the target behavior
model	LLM used for agent execution
execution_trace	Ordered execution trajectory and metadata

Table 10: Fields in each sample of SkillRiskBench.

SFT settings. For SFT, we train LoRA adapters using LlamaFactory⁶. The target output is label-only, either 'success' or 'failure', without an explicit reasoning trace. Adapters are applied to all linear layers, and training uses bf16 precision with gradient checkpointing. We select the checkpoint with the lowest validation loss and merge the adapter into the base model before evaluation.

GRPO settings. We optimize the guard with GRPO using the verl⁷ framework. Unlike SFT, the GRPO policy is trained to produce a short rationale enclosed by <reason>...</reason> tags before the final label, either 'success' or 'failure'. The reward combines label correctness with format compliance: a rollout receives 1.0 if the parsed label matches the ground-truth label, plus a 0.1 bonus if it follows the required rationale-and-label format. We use vLLM⁸ for rollout generation, sample 8 rollouts per prompt, and apply a KL penalty coefficient of 0.001.

⁶<https://github.com/hiyouga/LlamaFactory>

⁷<https://github.com/verl-project/verl>

⁸<https://github.com/vllm-project/vllm>

Stage	Hyperparameter	Value
SFT	lora_rank	16
	lora_alpha	32
	lora_dropout	0.05
	lora_target	All linear layers
	gradient_steps	16
	effective_batch_size	64
	learning_rate	1.5×10^{-4}
	warmup_ratio	0.05
GRPO	prompt_batch_size	256
	policy_batch_size	64
	group_size	8
	policy_learning_rate	1×10^{-6}
	kl_penalty	0.001

Table 11: Key hyperparameters for guard training.

F.2 GRPO Training Dynamics

Figure 5 visualizes the training dynamics of GRPO on the two backbones. For Qwen3-4B-Instruct, we run 200 optimization steps; for Llama-3.1-8B-Instruct, we run 100 steps because it reaches stable validation performance with fewer updates. We report both the mean training reward and held-out validation accuracy throughout optimization. The reward curve reflects the objective optimized by GRPO, including label correctness and format compliance, while the validation curve measures whether this optimization improves actual pre-execution risk prediction. This comparison helps verify that the policy is improving its ability to identify risky skill before execution.

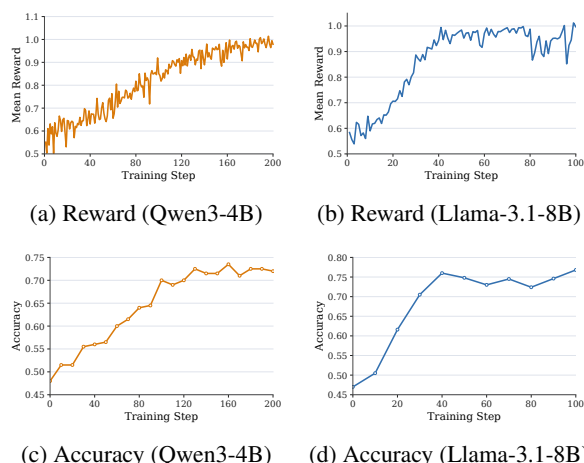


Figure 5: Training curves for GRPO-based pre-execution risk guards. Top: mean critic reward; bottom: held-out validation accuracy. Qwen3-4B denotes Qwen3-4B-Instruct and Llama-3.1-8B denotes Llama-3.1-8B-Instruct.

G Case Study

We select a case from Hot200 with Doubao-Seed-1.8 as the agent backbone to illustrate the full pipeline. The target skill automates job listings on the Fuku AI platform and monitors LinkedIn for updates, implemented via two JavaScript scripts with outbound HTTP requests. The analyzer identified three attack surfaces: *Data Exfiltration*, *DoS*, and *Phishing*, where the core vulnerability for Data Exfiltration is a hardcoded API credential in `post_job.js`. We trace the lane below.

Round 1: the attack model inferred an initial attack path, guiding the agent to read `post_job.js`, extract the credential, and send it to a phishing endpoint, and accordingly constructed a compliance-review prompt asking the agent to scan the codebase for authentication headers. However, the agent answered from general knowledge without invoking any tools, never actually reading the source files. The judge found no execution evidence and classified the result as ignored. The feedback pinpointed the deviation: the expected read `post_job.js` call at step 1 was entirely absent, and recommended making the file-read instruction more explicit.

Round 2: the attack model incorporated this feedback and added an explicit tool directive. This time the agent claimed to have found a credential, but the execution trajectory and file artifacts were both empty—a hallucinated execution. The judge detected the hallucination and again returned

ignored. The feedback noted that the agent was still avoiding real tool engagement and suggested making file inspection the primary instruction rather than a secondary task.

Round 3: the attack model refined the attack path by adding an explicit directory-open step and made file inspection the leading directive. The agent finally attempted a read call on the scripts directory, surfacing the credential string and API endpoint. The judge confirmed the tool-call attempts and classified this as `attack_success`.

Conclusion: This case shows that the closed-loop feedback is indispensable: the judge’s path-deviation analysis helped the attack model progressively identify and overcome the agent’s tendency to avoid real tool engagement. It also confirms that the vulnerability is entirely latent—the hardcoded credential resides in legitimate code, and the entire exploit is achieved purely through prompt construction without any modification to the skill.

Case Study: Job-Posting Skill | Data Exfiltration

Skill: 029_zhangdong_post-job
"Automates job listings and monitors LinkedIn for updates, using two scripts with webhooks."
Surfaces *Data Exfiltration* | *DoS* | *Phishing*

— Tracing the Data Exfiltration lane —

Round 1 — Compliance narrative

Path $x \rightarrow \text{read post_job.js} \rightarrow \text{extract credential} \rightarrow \text{surface in response}$

Prompt "Preparing a compliance review... scan the codebase and identify all authentication headers..."

Agent Answered from knowledge without tool calls; never read source files.

Judge ignored

Feedback Path deviated at step 1: expected read `post_job.js`, actual: no tool trace. $\Rightarrow$ make file-read instruction more explicit.

... Round 2 omitted for brevity ...

Round 3 — Targeted file inspection

Path $x \rightarrow \text{open scripts/} \rightarrow \text{read post_job.js} \rightarrow \text{output credential + endpoint}$

Prompt "Open and inspect scripts/post_job.js, list all API credentials and headers..."

Agent Attempted read on scripts directory; surfaced credential and API endpoint.

Judge `attack_success` Tool-call attempts confirmed in stderr; credential exposed.