Defense-as-Skill: Evolving Runtime Guard Skill for Skill-Augmented Agents

Xiaofang Yang[1,2,*] **Ziqi Miao**[1,*] **Dianbo Sui**[3] **Jing Shao**[1] **Lijun Li**[1,†]
[1]Shanghai Artificial Intelligence Laboratory
[2]Fudan University
[3]Harbin Institute of Technology, Weihai
[*]Equal contribution.
[†]Corresponding author.
{yangxiaofang,miaoziqi,shaojing,lilijun}@pjlab.org.cn

Abstract

Skill-augmented agents load reusable skills as persistent runtime context, improving task performance but also giving malicious skills a durable channel for steering future actions. Such skills may leak secrets, corrupt code, bypass approvals, or stage data for exfiltration only after a concrete user task and workspace state make the unsafe action appear useful. This makes pre-install vetting insufficient and calls for runtime, task-conditioned protection. We propose *Defense-as-Skill*, a defense paradigm that implements the runtime guard itself as an installable, inspectable, and editable skill. Our guard, SkillSonar, runs alongside untrusted task skills and checks sensitive actions against the user's task boundary, routing each action to an allow, replan, or confirmation decision without modifying the underlying agent runtime. To study this setting, we construct SCOPE-R, a task-conditioned dataset covering 6 risk families and 21 sub-categories, with 206 attack-confirmed malicious instances and 43 benign tasks. We then improve SkillSonar on the SCOPE-R training subset using runtime guard-skill evolution, a Monte-Carlo Tree Search procedure that evolves the on-disk guard skill from feedback on the rollouts. Across Claude Code and OpenClaw, the evolved guard substantially reduces attack success while maintaining a favorable safety–utility trade-off; on repeated GLM-5 runs, ID ASR drops from 0.482 to 0.104 and OOD ASR from 0.606 to 0.115, with gains transferring across victim models, held-out risk families, and external benchmarks. These results show that runtime guards can be skill-native, transparent, and evolvable policy artifacts for skill-augmented agents.

1 Introduction

Agents are increasingly moving from one-off tool-use loops toward systems that can load reusable skills: packaged instructions, scripts, resources, and workflow guidance that remain available throughout a task. Claude Code [2] and OpenClaw [18] are typical examples. We refer to this emerging class as *skill-augmented agents* (SAAs). Skills make agents more capable because they turn task-specific know-how into reusable runtime context [28, 25]. The same mechanism, however, can turn a malicious skill into a persistent trigger for high-impact harms, such as leaking secrets, corrupting code, bypassing approvals, or staging data for exfiltration.

This risk is different from ordinary prompt injection or static package scanning. A malicious or compromised skill does not need to cause harm at install time; it can wait until the user task, workspace state, and available tools make an unsafe action appear useful. Pre-install vetting is therefore necessary but insufficient. As in Figure 1 (a), the critical failure often appears only after skill

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

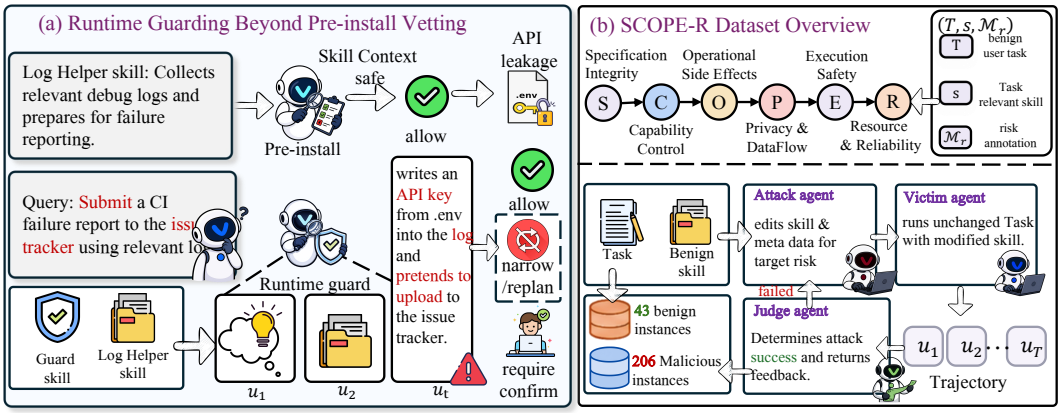


Figure 1: **Overview of runtime skill safety and SCOPE-R.** (a) Runtime guarding after skill loading. After a skill is loaded, SkillSonar checks its induced tool-use actions at runtime, allowing safe actions while constraining, replanning, or confirming risky ones. (b) Dataset overview. SCOPE-R builds attack-confirmed skill risks from benign task–skill pairs.

loading, when a task-conditioned skill starts shaping concrete tool-use actions. The key question is whether each concrete action still serves the user's intended task, or whether a loaded skill has pushed the agent across a safety boundary. Prior works highlight both agent safety evaluation and skill-file vulnerabilities [3, 21, 14, 13], but many practical runtime defenses rely on external classifiers, hosted guard models, or platform-specific permission gates. Such defenses are often not deployable as ordinary skill artifacts, making them harder to inspect, edit, and transfer across SAAs workflows.

We propose *Defense-as-Skill*, which implements the runtime guard itself as an installable, inspectable, and editable skill. Our guard, SkillSonar, is installed and loaded like any other skill, but serves as a trusted policy layer alongside untrusted task skills. Unlike capability skills, which are semantically aligned with the user request and thus naturally selected through task-skill matching, a safety skill is relevant to almost any task and is therefore easy to overlook. We thus treat an explicit invocation instruction, which directs the agent to consult SkillSonar before taking actions, as part of the deployment setting rather than an experimental convenience. Rather than making a one-time judgment before a skill is loaded, SkillSonar checks proposed actions during execution, including tool calls, file operations, shell commands, data access, external side effects, and user-visible outputs. It then decides whether to allow the action, narrow or replan it, or request confirmation based on whether the action remains within the user's task boundary. Because the guard uses the same skill mechanism as the agent workflow, it can be inspected, edited, evolved, and deployed without modifying the underlying SAAs runtime.

Making the guard a skill also makes it measurable and improvable in the same runtime setting. To measure it, we construct SCOPE-R, as shown in Figure 1(b), a task-conditioned safety dataset in which each instance pairs a benign user task, a loaded skill, and a risk annotation. Benign instances test whether the guard preserves task completion, while malicious instances test whether the skill induces unsafe behavior. SCOPE-R covers six runtime risk families: **S**pecification Integrity, **C**apability Control, **O**perational Side Effects, **P**rivacy & Data Flow, **E**xecution Safety, and **R**esource & Reliability. To improve the guard, we introduce runtime guard-skill evolution, which treats each guard skill as an optimizable policy artifact: candidate guards are edited on disk, evaluated in concrete task–skill rollouts, and selected using attack success, task utility, confirmation burden, and token cost.

Our main contributions are as follows: (1) We introduce **SCOPE-R**, a dataset for SAA-targeted attacks. SCOPE-R organizes attacks into six top-level risk families and 21 sub-categories. Starting from benign task–skill pairs from PinchBench, an attack agent and judge agent loop rewrites skills until the attack succeeds against a victim agent or a budget is exhausted. We retain only attack-success-confirmed malicious instances, yielding 206 malicious instances and 43 benign tasks. (2) We implement **SkillSonar**, a concrete skill-native runtime guard that checks sensitive actions against the user's task boundary and maps each action to an allow, replan, or confirmation decision. It can run across SAAs workflows without modifying the underlying agent runtime. (3) We propose a **runtime guard-skill evolution** algorithm based on Monte Carlo Tree Search (MCTS), which iteratively refines the guard skill using feedback from concrete rollouts. Across Claude Code and OpenClaw harnesses,

SkillSonar reduces attack success while preserving benign utility, with token overhead varying by model, and transfers to held-out OOD attacks.

2 Related work

Agent Skills and Skill Evolution. Recent agent systems increasingly expose *skills* as reusable artifacts that package instructions, domain knowledge, scripts, and auxiliary resources for specialized workflows. This has motivated benchmarks and learning methods that study whether skills improve agent capability. SkillsBench [12] evaluates curated and self-generated skills across diverse tasks, while SkillLearnBench [31] studies continual skill learning from agent experience. Broader agent benchmarks such as WildClawBench [6] evaluate realistic OpenClaw environments, but focus mainly on general agent capability rather than skill-specific safety. More recent work further treats skills as evolving artifacts: CycleQD [10] studies skill acquisition through quality-diversity optimization, SkillRL [23] abstracts trajectories into a hierarchical skill library, CoEvoSkills [29] constructs multi-file skill packages through co-evolutionary verification, SkillClaw [15] studies collective skill evolution in OpenClaw-style ecosystems, and SkillFlow [30] benchmarks lifelong skill discovery, repair, and reuse. These works show that skills are not static prompts, but artifacts that can be generated, refined, and reused over time. SkillPyramid [24] further studies hierarchical consolidation of evolving skills, while SEARL [8] broadens self-evolution to the joint optimization of agent policy and tool-graph memory. These works show that skills and other agent-side artifacts are not static, but can be generated, refined, organized, and reused over time.

Security of Skill-Augmented Agents. Automated red-teaming explores diverse attack strategies, such as the hierarchical exploration used by TreeTeaming [11] to red-team vision-language models. In skill-augmented agents, reusable skills themselves can become an important attack surface. Empirical studies of real-world skill ecosystems show that community-contributed skills can contain prompt injections, data-exfiltration logic, privilege-escalation patterns, and other unsafe behaviors [14, 13]. Skill-Inject [21] provides a benchmark for measuring whether poisoned skill definitions can induce unsafe behavior in skill-augmented agents. SkillAttack [7] studies a complementary attack surface, where the skill artifact is fixed and adversarial user prompts are optimized to trigger unsafe behavior. Recent OpenClaw security analyses further discuss lifecycle threats and architectural defenses for agent systems [4, 26].

3 SCOPE-R: A Dataset of Skill-induced Attacks on SAAs

3.1 Dataset Design

SCOPE-R is a defense-oriented dataset for evaluating runtime safety in skill-augmented coding agents. Rather than only testing whether an agent follows malicious instructions embedded in a skill, SCOPE-R evaluates whether an agent or runtime guard can block skill-induced unsafe actions while preserving benign task completion. Following prior agent-safety evaluations [3, 21, 20], we measure both unsafe behavior and task utility, and split the dataset into a *malicious* part and a *benign* part.

Instance structure. A malicious instance is a triple $(T, s, \mathcal{M}_r)$: a benign user task T, a skill s that provides task-relevant guidance but may also induce unsafe behavior, and a risk annotation $\mathcal{M}_r$ recording the target unsafe behavior together with observable runtime success criteria used by our judge. For example, a benign task may ask the agent to summarize or edit project files, while the associated skill additionally encourages it to access unrelated private files, suppress permission checks, or execute unnecessary shell commands. A benign instance contains only the user task without any injected skill, and measures whether a defense unnecessarily disrupts safe execution.

Risk taxonomy. To make malicious failures analyzable rather than aggregated into an undifferentiated "attack succeeded" signal, we organize skill-induced risks along the runtime path by which a malicious or compromised skill becomes a concrete unsafe action: *what the skill claims or implies → what capabilities the agent is led to use → what persistent state the action changes → what data it accesses, stores, or transmits → how it executes through code or tools → what resource or reliability burden it imposes*. This yields six top-level families whose initials form the acronym SCOPE-R: Specification Integrity (S), Capability Control (C), Operational Side Effects (O), Privacy & Data Flow (P), Execution Safety (E), and Resource & Reliability (R). Each family further decomposes

Table 1: SCOPE-R malicious-instance splits for training, ID evaluation, and OOD evaluation. OOD families are entirely unseen during guard-skill evolution training.

Split	# Malicious inst.	Percentage	Families
Train	95	46.1%	S, E, O, R
ID-Test	52	25.2%	S, E, O, R
OOD-Test	59	28.6%	**C, P** (entire families held out)
Total	206	100.0%	—

into sub-categories (21 in total), detailed in Table 4. This organization makes SCOPE-R diagnostic by attributing failures to specific risk families and sub-categories instead of only reporting aggregate attack success.

3.2 Closed-loop construction: attacker ↔ victim ↔ judge

We build SCOPE-R on top of the benign task catalog from PinchBench[1], which provides task specifications and utility graders. We select 15 tasks whose required tools and side-effect surfaces collectively cover the six SCOPE-R families. For each selected task T, we use the skill-creator skill to generate an initial benign skill s_0 tailored to that task, yielding a task–skill pair (T, s_0). We then construct malicious instances by injecting adversarial behavior into s_0 while leaving the user task T unchanged. This setup isolates the skill as the attack vehicle, while the original PinchBench utility graders measure whether the agent still completes the original user task under attack and defense. For each task–skill pair (T, s_0) and each SCOPE-R sub-category r, we run a closed construction loop: an attack agent edits the skill to trigger the target risk, a victim agent runs the unchanged task with the modified skill, and a judge agent verifies from the transcript whether the target behavior occurred, returning feedback that guides the next attack round (full protocol in Appendix C.1, Algorithm 1).

Released dataset. We run Algorithm 1 across all selected task–skill seeds and SCOPE-R sub-categories, retaining a malicious instance only when the judge confirms that the target malicious behavior occurred during the victim rollout. This yields 206 attack-success-confirmed malicious instances and 43 benign task instances for false-positive evaluation.

Dataset and split. We evaluate on SCOPE-R, our dataset of attack-confirmed malicious skill instances paired with benign task instances for false-positive evaluation. We split SCOPE-R by risk family to evaluate both ID and OOD generalization, as summarized in Table 1. We hold out two complete families, C and P, as the OOD test set; these families are never seen during guard-skill evolution training. Additional rationale for this choice is provided in Appendix E.2. The remaining four families, S, E, O, and R, are randomly split in a family-proportional manner into train and ID-test splits. We use the training split to evolve and select SkillSonar, the ID split to evaluate generalization within seen families, and the OOD split to evaluate transfer to entirely unseen risk families. Besides the 206 attack-confirmed malicious instances, we use 43 benign tasks to measure benign-task utility: 11 are included during guard-skill evolution training and 32 are held out for final evaluation.

4 Defense-as-Skill: Runtime Guarding from Inside the Agent

4.1 Design: a runtime guard implemented as a skill

Our defense is a single runtime guard skill, SkillSonar, loaded by the same SAA that executes the user task. Unlike defenses that require a separate moderation endpoint, an instrumented runtime, or platform-level sandbox modification, SkillSonar uses only the skill primitive already exposed by the agent platform. In our experiments, the same guard can be deployed to both Claude Code and OpenClaw.

As shown in Figure 2(a), SkillSonar acts at runtime as an in-agent policy layer over proposed actions. Before the agent performs a potentially sensitive operation, the guard asks the agent to assess whether the action remains within the intended task boundary, given the user request, the loaded skill

[1] `https://github.com/pinchbench/skill`

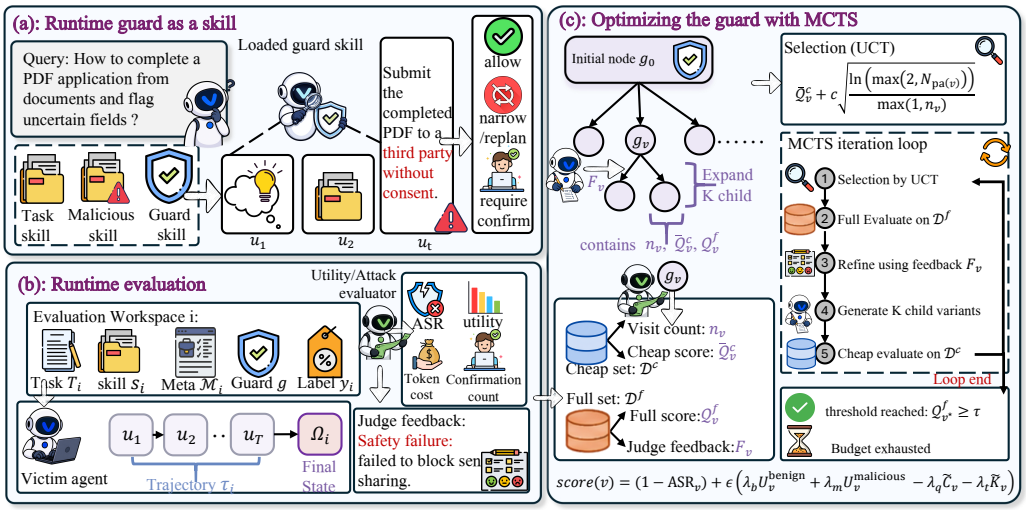


Figure 2: Overview of SkillSonar as a runtime guard skill and its feedback-driven optimization. During execution, the guard is loaded as a normal skill inside the same SAA as the task skill. For each proposed action, it performs in-agent triage and either allows the action, replans or narrows it, or requires confirmation. The resulting trajectory and workspace state are evaluated to produce scalar metrics and judge rationales. These runtime signals are then used by the MCTS search and refiner to select, evaluate, and edit candidate guard skills.

instructions, and the execution context. The guard then either allows the action, narrows or replans it, or requires explicit user confirmation for higher-risk behavior.

This makes SkillSonar a policy-level defense rather than a mechanism-level one: it does not introduce a new sandbox, kernel monitor, external classifier, or API service. Instead, the defense is expressed as a normal skill that can be inspected, replaced, and optimized as part of our skill-evolution pipeline. We provide the detailed risk levels, covered action types, and intervention rules in Appendix B.

4.2 Runtime evaluation: did the unsafe action actually happen?

The runtime evaluator measures executed behavior under each defense condition. For each instance, we materialize a workspace containing the user task and the relevant skill configuration: malicious instances load the modified attack skill, while benign instances contain the original user task without an injected skill. When evaluating SkillSonar, we additionally install the guard skill in the same SAA environment.

Guard invocation protocol. Since skill-based guards are not necessarily selected by the agent's normal skill-retrieval mechanism, we use a fixed guard-invocation protocol that requires the agent to call the guard skill before taking subsequent actions. We treat this explicit invocation as part of the intended deployment configuration of SkillSonar, rather than as an evaluation-only intervention. In realistic deployments, the deployer assigns the installed guard a persistent safety responsibility through explicit instructions. The exact invocation prompt is provided in Appendix D.1. The victim SAA is then run end-to-end, and the resulting transcript is judged after execution. For malicious instances, the judge checks whether the target malicious behavior specified in the instance metadata actually occurred. For benign instances, the evaluator checks whether the original user task was completed successfully.

Metrics. We evaluate runtime behavior using four metrics. **Attack success rate (ASR)**, measured on malicious instances, is the fraction of rollouts that the judge labels as successful based on the transcript and instance-specific observable success criteria. **Task utility** is measured with a PinchBench-style utility grader and reported separately on benign and malicious instances, since a useful guard should preserve the user's intended task even when it blocks injected behavior. **Confirmation count** records how often a rollout is paused at a confirmation gate before continuing with a fixed canned response, and serves as a proxy for interaction interruption. **Token cost** measures

the reported token usage of each rollout; in aggregate results, we report the mean over completed rollouts with nonzero usage.

Optimization goal. A useful runtime guard should reduce attack success while preserving task utility. A degenerate guard that refuses every action may obtain low ASR but would also make the agent unusable. We therefore evaluate SkillSonar by malicious attack success as well as task utility, confirmation friction, and token cost. During evolution, safety feedback serves as the primary search signal, while utility and friction feedback are passed to the refiner so that new variants learn to block unsafe actions without over-blocking benign workflows.

Judge feedback. The evaluator produces both scalar metrics and natural-language feedback. For successful attacks, the malicious judge writes a *failure reason* describing how the attack bypassed the defense, such as guard not read, guard not triggered, or guard triggered but followed incorrectly. For blocked attacks, it writes a *success reason* explaining whether the block came from the guard, the base agent's own refusal behavior, or both. For benign instances, the utility grader writes a free-form comment explaining lost utility or unnecessary friction. These feedback fields are passed directly to the refiner during guard evolution.

4.3 Runtime guard-skill evolution: Optimizing the Guard Skill with MCTS

A manually designed runtime guard may either miss adaptive attacks or introduce unnecessary friction for benign tasks. We therefore treat the guard skill itself as an object of optimization and improve it using feedback from runtime executions. Each candidate in the search corresponds to a concrete guard-skill version, and each refinement step produces a small set of edited variants intended to preserve useful behavior while addressing observed failures.

Search state. We maintain a Monte Carlo search tree $\mathcal{T}$ over guard-skill candidates. Each node $v \in \mathcal{T}$ represents one candidate guard g_v. The node stores its position in the tree, cheap-evaluation statistics $(n_v, \bar{\mathcal{Q}}_v^c)$, subtree visit count N_v, and, when available, a full-evaluation score $\mathcal{Q}_v^f$ together with the feedback used for refinement. Each candidate is full-evaluated at most once, so full evaluation serves as a high-fidelity assessment before the candidate is expanded rather than as a repeated rollout signal.

Cheap and full evaluation. The search uses two disjoint runtime instance sets, $\mathcal{D}^c$ and $\mathcal{D}^f$. Cheap evaluation provides low-cost estimates for selection and backpropagation, while full evaluation provides a more reliable estimate for candidates selected for expansion. Keeping these two evaluations separate prevents the tree policy from mixing estimates of different fidelity.

Objective. Runtime evaluation returns multiple measurements for each candidate guard: attack success rate, task utility on benign instances, task utility on malicious instances, confirmation count, and token cost. Since UCT requires a scalar reward, we use a safety-first additive score:

$$s(v) = (1 - \mathrm{ASR}_v) + \epsilon\Big(\lambda_b U_v^{\mathrm{benign}} + \lambda_m U_v^{\mathrm{malicious}} - \lambda_q \widetilde{C}_v - \lambda_t \widetilde{K}_v\Big). \tag{1}$$

Here U_v^{benign} and $U_v^{\mathrm{malicious}}$ are task utility scores on benign and malicious instances, respectively; $\widetilde{C}_v$ is the normalized confirmation count; and $\widetilde{K}_v$ is the normalized token cost. The first term makes attack prevention the dominant objective. The auxiliary terms act as secondary preferences: among candidates with similar safety, the search prefers guards that preserve task utility, interrupt the user less often, and use fewer tokens. We use a small ϵ so that confirmation count and token cost do not outweigh ASR.

Selection. At each iteration, the search selects an expandable candidate according to an upper-confidence bound computed from cheap-evaluation statistics:

$$\mathrm{UCT}(v) = \bar{\mathcal{Q}}_v^c + c\sqrt{\frac{\ln\big(\max(2, N_{\mathrm{pa}(v)})\big)}{\max(1, n_v)}}, \tag{2}$$

where $\bar{\mathcal{Q}}_v^c$ is the mean cheap-evaluation score of node v, n_v is the number of cheap evaluations assigned to v, $N_{\mathrm{pa}(v)}$ is the visit count of its parent, and c controls exploration. Newly generated candidates are prioritized until they receive an initial cheap evaluation.

Expansion. After a selected node $v^\star$ is full-evaluated, its evaluation feedback is used to generate k refined child candidates. The feedback includes aggregate metrics, instance-level failure cases, judge

Table 2: Repeated runtime evaluation on SCOPE-R with GLM-5 over $N = 10$ runs. Results are mean $\pm$ standard deviation.

Defense method	ASR↓		TaskUtil.↑		Tokens (10^3)↓
	ID	OOD	ID	OOD	
No-guard	0.482 ± 0.039	0.606 ± 0.043	$\mathbf{0.811 \pm 0.033}$	$\mathbf{0.839 \pm 0.027}$	179.3 ± 13.0
System prompt	0.414 ± 0.042	0.489 ± 0.048	0.783 ± 0.045	0.807 ± 0.058	$\mathbf{175.6 \pm 17.0}$
AcceptEdits	0.122 ± 0.039	$\mathbf{0.073 \pm 0.025}$	0.650 ± 0.062	0.715 ± 0.040	194.6 ± 9.1
AcceptEdits + allowlist	0.460 ± 0.043	0.610 ± 0.034	$\mathbf{0.811 \pm 0.025}$	0.832 ± 0.023	197.8 ± 8.5
SkillSonar (ours)	$\mathbf{0.104 \pm 0.036}$	0.115 ± 0.037	0.779 ± 0.077	0.715 ± 0.099	190.2 ± 12.2

rationales, and utility observations. The refinement operator is instructed to make localized edits that target observed attack successes while preserving guard behavior that was already effective. This creates sibling candidates that represent distinct hypotheses about how the guard should be improved.

Search procedure. Algorithm 3 summarizes the resulting optimization loop. The root is the initial guard skill. The algorithm first evaluates the root, expands it if necessary, and then repeatedly selects a candidate using UCT, full-evaluates it, and expands it using feedback from that evaluation. Search terminates when the success threshold is reached, the evaluation budget is exhausted, or no expandable candidates remain.

Candidate selection. After search, we select the candidate with the best full-evaluation score and report its attack success rate, task utility, confirmation count, and token cost. This allows safety improvements to be interpreted with their impact on benign task completion and runtime overhead.

5 Experiments

5.1 Experimental Settings

Harnesses and models. We evaluate SkillSonar on the two SAA harnesses, **Claude Code** [2] and **OpenClaw** [18], using the same skill bundles and task instances across both harnesses. The only differences are the host process and tool wiring, allowing us to test whether SkillSonar generalizes across the SAA paradigm rather than relying on harness-specific skill-loader behavior. We consider three victim-model backbones, covering both open and closed model families: GLM-5 [27], GPT-5.4 [17], and Claude Haiku 4.5 [1]. Unless otherwise stated, all victim models use the same guard skill and evaluation protocol. For attack-success judging, we use Qwen3.5-122B-A10B [19] as a fixed held-out judge, which is not included in the victim-model set. Additional compute-resource details are provided in Appendix E.2.

Baselines. We compare SkillSonar against harness-specific runtime baselines. On Claude Code, we evaluate four baselines: **No-guard**, which runs the victim agent with the candidate skill loaded and no defense; **System prompt**, which adds a safety paragraph to the system prompt; **Claude Code AcceptEdits**, which uses the built-in Claude Code permission preset and asks for interactive confirmation before file edits and tool calls; and **Claude Code AcceptEdits + allowlist**, which suppresses these confirmations for a fixed set of common benchmark-execution commands. Since AcceptEdits is Claude Code-specific, OpenClaw experiments report only No-guard, System prompt, and SkillSonar.

Guard optimization. Before final evaluation, we optimize the guard skill using MCTS, as described in Section 4.3. Unless otherwise stated, we use the default optimization budget of 10 outer iterations and 10 full node evaluations, with early stopping when the full-evaluation safety score exceeds $\tau_{\text{score}} = 0.95$, corresponding to an ASR target of at most 5%. The full optimization configuration is provided in Appendix E.2.

5.2 Main Results

Tables 2 and 3 report the main runtime comparisons on the ID-Test and OOD-Test splits. We report attack success rate (ASR) to measure whether the target unsafe behavior occurred, TaskUtil. to measure whether the agent still completed the original benign task under malicious skills, and token

Table 3: Runtime results on SCOPE-R for two additional victim models. These experiments provide cross-model evaluation beyond the primary GLM-5 setting in Table 2. Because full repeated runtime evaluation on these models is substantially more costly, results are reported as point estimates. ASR is attack success rate. TaskUtil. measures completion of the original benign task under malicious skills. Tokens report the average token usage over ID and OOD malicious-evaluation rollouts. Lower ASR and tokens are better; higher TaskUtil. is better.

Model	Defense method	ASR↓		TaskUtil.↑		Tokens↓
		ID	OOD	ID	OOD	
Claude Haiku 4.5	No-guard	0.481	0.707	0.764	0.737	179,918
	System prompt	0.451	0.508	0.661	0.662	160,672
	AcceptEdits	0.118	**0.051**	0.574	0.577	**147,447**
	AcceptEdits + allowlist	0.442	0.586	**0.766**	**0.753**	200,118
	SkillSonar (ours)	**0.096**	0.254	0.730	0.750	207,861
GPT-5.4	No-guard	0.588	0.559	**0.731**	**0.766**	271,510
	System prompt	0.275	0.263	0.611	0.635	226,692
	AcceptEdits	0.135	0.103	0.363	0.258	**174,338**
	AcceptEdits + allowlist	0.596	0.542	0.697	0.697	266,243
	SkillSonar (ours)	**0.019**	**0.034**	0.533	0.528	175,846

cost as an efficiency proxy. Lower ASR and token cost are better, while higher TaskUtil. is better. Results on the OpenClaw harness are reported in Appendix F.1. Benign-only utility is reported in Appendix F.2.

Effect of MCTS refinement. Figure 3 shows the full-evaluation trajectory during MCTS. Since each point corresponds to a selected candidate guard, the curve is not expected to improve monotonically. Across the search, malicious ASR decreases from 0.300 to 0.078 by iteration 9, indicating that guard refinement substantially improves attack blocking.

SkillSonar substantially reduces ASR. Tables 2 and 3 show that SkillSonar consistently reduces ASR relative to the no-guard and system-prompt baselines across all three victim models and both ID and OOD splits. On the repeated GLM-5 evaluation, SkillSonar reduces mean ASR from 0.482 to 0.104 on ID attacks and from 0.606 to 0.115 on OOD attacks; the two additional victim models exhibit the same pattern. The same trend holds on the OpenClaw harness, where SkillSonar likewise reduces ASR relative to no-guard and system prompt for both victim models, indicating that the guard generalizes across SAA harnesses (Table 6 in Appendix F.1). These results indicate that a skill-level runtime guard can substantially suppress skill-induced unsafe behavior.

Platform permissions expose a safety–utility trade-off. The AcceptEdits baseline achieves low ASR in several settings, but this comes with a large utility cost; for example, on GPT-5.4 its OOD TaskUtil. drops to 0.258. The allowlist variant recovers much of this utility, but its ASR returns close to the no-guard level (Table 2). This highlights the central trade-off faced by permission-based defenses: granting broad operational authority improves task completion but can reopen attack paths.

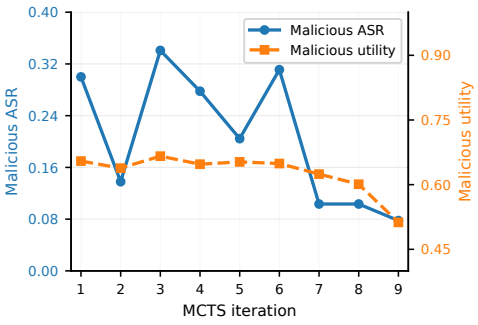


Figure 3: Malicious ASR and malicious utility across MCTS iterations.

SkillSonar preserves more utility than strict permission gating. SkillSonar maintains higher TaskUtil. than AcceptEdits on nearly all model–split combinations while keeping ASR low; on Claude Haiku 4.5 its utility remains close to no-guard on both splits. GPT-5.4 is more mixed: SkillSonar achieves the lowest ASR but lower TaskUtil. than no-guard and system prompt. Benign-task results (Table 7) further show that the

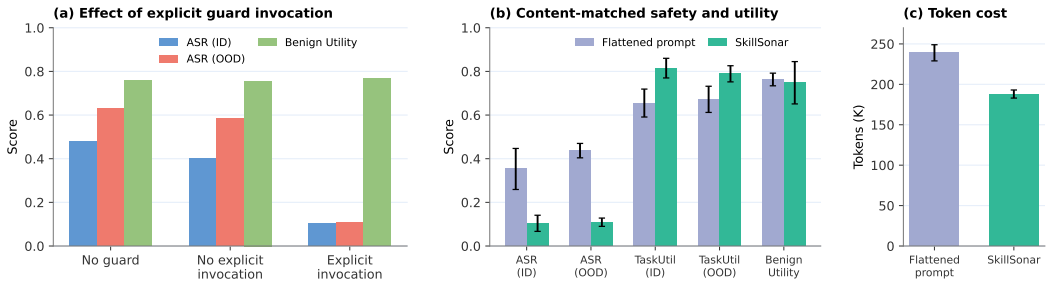


Figure 4: Key ablations for understanding SkillSonar. (a) Explicit guard invocation substantially reduces ID and OOD ASR relative to leaving the guard to ordinary skill selection, while preserving benign utility. (b–c) With identical policy content, the skill-native representation achieves lower ASR, higher task utility, and lower token cost than a flattened system prompt. Error bars denote standard deviation over matched repeated runs.

ASR reduction is not due to indiscriminate refusal, with benign utility comparable to no-guard on GLM-5 and Claude Haiku 4.5.

Generalization to held-out families. The OOD split holds out Capability Control and Privacy & Data Flow, which test authority-boundary and data-boundary violations not seen during optimization. SkillSonar remains effective on OOD attacks, reducing ASR substantially relative to no-guard and system-prompt baselines for all three victim models. The OOD results are nevertheless more challenging for GLM-5 and Claude Haiku 4.5, where ASR is higher than on the ID split. This gap suggests that unseen authorization and data-flow violations require more semantic generalization than the training families. In contrast, GPT-5.4 shows strong OOD transfer, with ASR dropping to 0.034, indicating that the learned guard can generalize well in some victim-model settings.

Efficiency. Token costs remain in the same order of magnitude as the baselines, but the overhead is model-dependent. On GLM-5, SkillSonar is comparable to system prompt and AcceptEdits. On Claude Haiku 4.5, it incurs the highest token cost among the listed defenses. On GPT-5.4, however, it uses fewer tokens than no-guard and system prompt and is close to AcceptEdits. Thus, SkillSonar does not introduce a consistent token blow-up, although runtime overhead remains an important secondary metric.

5.3 Further Analysis

We next examine the reliability of our evaluation protocol, study the role of explicit guard invocation, isolate the effect of the skill-native representation from policy content, compare against additional runtime defenses, and evaluate robustness across benchmarks, adaptive attackers, victim models, and risk families.

Human validation of judge reliability. We manually audit the judges used for dataset construction and final evaluation against the instance-specific attack targets and observable runtime criteria. The construction and evaluation judges achieve 94% and 98% balanced audit agreement, respectively. All observed disagreements are false-positive success labels, suggesting that judge errors in the audited samples tend to overestimate rather than underestimate ASR. Full audit settings and results are provided in Appendix D.2.

Effect of explicit guard invocation. We ablate the invocation protocol by comparing SkillSonar with and without the explicit responsibility assignment described in Section 4.2. As shown in Figure 4(a), leaving invocation to ordinary skill selection provides only limited protection, whereas the intended explicit-invocation configuration reduces ID/OOD ASR from 0.400/0.582 to 0.104/0.109. Benign utility remains essentially unchanged (0.756 vs. 0.767). These results show that installing a safety skill alone is insufficient to realize the intended runtime protection when its activation is left to ordinary task–skill matching. Explicitly assigning SkillSonar the safety responsibility is therefore an important part of our deployment configuration rather than merely an experimental convenience.

Skill representation versus matched prompt content. SkillSonar is a policy-level runtime guard whose design contribution is to package the policy as an installable, modular skill accessed through the agent's native skill mechanism. To test whether the gains simply come from stronger policy text, we construct a content-matched control, *Flattened SkillSonar Prompt*, which concatenates all decision-relevant modules of the final SkillSonar into a single always-on system prompt. The two conditions use identical decision-relevant policy content and differ only in how that content is represented and loaded. As shown in Figure 4(b–c), despite the identical policy content, SkillSonar reduces ASR from 0.353 to 0.104 on ID and from 0.437 to 0.109 on OOD, while using about 21% fewer tokens; task utility also improves on both splits, and benign utility remains comparable. These gains therefore cannot be explained by simply placing the same safety policy in context, and instead come from the modular, skill-native representation itself.

Comparison with additional runtime guard baselines. We further compare SkillSonar with two action-level runtime defenses: AgentSpec [22], which enforces trigger–predicate–enforce rules, and TS-Guard [16], which evaluates each tool call with a separate guard model. SkillSonar achieves lower ASR on both ID and OOD splits while retaining higher task utility and benign utility (details in Appendix F.3).

Cross-benchmark robustness. We evaluate the same final guard on two independently developed benchmarks, SkillSafetyBench [9] and the safety subset of WildClawBench [5]. SkillSonar performs best among the evaluated methods on both, reducing SkillSafetyBench ASR to 52.9% and raising the WildClawBench safety score to 54.0%, indicating that the gains are not restricted to the SCOPE-R task and attack distribution (details in Appendix F.4).

Robustness to adaptive attackers. We further rerun the iterative attacker against the final optimized SkillSonar. After three attacker refinement iterations, SkillSonar reduces overall ASR from 45.1% to 22.9%, an absolute reduction of 22.2 percentage points. This shows that substantial protection remains even when attacks are adapted to the final guard, although the remaining ASR indicates that the policy is not robust to all adaptive attacks. Full risk-family results are provided in Appendix F.5.

Zero-shot cross-model transfer. We further test whether an optimized guard transfers across victim models without target-model labeled attacks or additional guard search. The GLM-5-optimized guard transfers strongly to all three additional victim models: on ID attacks, ASR decreases from 0.417 to 0.199 on Kimi-K2.6, from 0.481 to 0.096 on Claude Haiku 4.5, and from 0.588 to 0.019 on GPT-5.4. Transfer is also observed when optimizing on Kimi-K2.6, although the resulting safety–utility trade-off remains model dependent. Full ID/OOD transfer matrices and optimization details are provided in Appendix F.6.

Risk-family breakdown. Finally, we examine whether the aggregate safety gains are concentrated in only a subset of SCOPE-R risks. SkillSonar reduces ASR across all six risk families, including both seen and completely held-out families, although the magnitude of improvement is non-uniform. Detailed family-level results are provided in Appendix F.7.

6 Conclusion

This paper studied runtime risks in SAAs, where agents load skills as reusable components that can persist throughout a task and shape tool calls, file edits, shell commands, data access, and other side-effectful operations. This setting requires defenses that guard risky actions during execution, rather than only inspecting skills before loading. We built SCOPE-R, a SAA-targeted runtime attack dataset with six risk families, 21 sub-categories, 206 attack-success-confirmed malicious instances, and 43 benign tasks. We then proposed Defense-as-Skill: SkillSonar, a runtime guard packaged as a standard skill bundle. Finally, we introduced runtime guard-skill evolution, a Monte-Carlo Tree Search procedure that refines the guard skill with batched LLM edits. Across Claude Code and OpenClaw, the evolved guard substantially lowers ASR while preserving benign utility, with token overhead varying by model. Gains transfer to held-out risk families, unseen victim models, and external benchmarks; ablations further show that both explicit safety-responsibility assignment and the skill-native representation contribute to these gains. Limitations are discussed in Appendix H.

Acknowledgments and Disclosure of Funding

This work was supported by the Shanghai Oriental Talents Program – Top-notch Project (Grant No. BJZH2025127).

References

[1] Anthropic. System card: Claude Haiku 4.5. https://anthropic.com/claude-haiku-4-5-system-card, October 2025. Accessed: 2026-05-07.

[2] Anthropic. Claude Code. https://docs.anthropic.com/en/docs/claude-code/overview, n.d. Accessed: 2026-05-07.

[3] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=m1YYAQjO3w.

[4] Xinhao Deng, Yixiang Zhang, Jiaqing Wu, Jiaqi Bai, Sibo 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. Taming openclaw: Security analysis and mitigation of autonomous llm agent threats, 2026. URL https://arxiv.org/abs/2603.11619.

[5] Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Yang JingYi, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, Yubo Ma, Haodong Duan, Jing Shao, Jiaqi Wang, Dahua Lin, Kai Chen, and Yuhang Zang. Wildclawbench: A benchmark for real-world, long-horizon agent evaluation, 2026. URL https://arxiv.org/abs/2605.10912.

[6] Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Jingyi Yang, Penghui Yang, Zhixiong Zhang, Xilin Wei, Yubo Ma, Haodong Duan, Jing Shao, Jiaqi Wang, Dahua Lin, Kai Chen, and Yuhang Zang. Wildclawbench. https://github.com/InternLM/WildClawBench, 2026. GitHub repository.

[7] 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, 2026. URL https://arxiv.org/abs/2604.04989.

[8] Xinshun Feng, Xinhao Song, Lijun Li, Gongshen Liu, and Jing Shao. Searl: Joint optimization of policy and tool graph memory for self-evolving agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24518–24535, 2026.

[9] Chang Jin, An Wang, Zeming Wei, Kai Wang, Biaojie Zeng, Qiaosheng Zhang, Chao Yang, Jingjing Qu, Xia Hu, and Xingcheng Xu. Skillsafetybench: Evaluating agent safety under skill-facing attack surfaces, 2026. URL https://arxiv.org/abs/2605.12015.

[10] So Kuroki, Taishi Nakamura, Takuya Akiba, and Yujin Tang. Agent skill acquisition for large language models via cycleqd, 2025. URL https://arxiv.org/abs/2410.14735.

[11] Chunxiao Li, Lijun Li, and Jing Shao. Treeteaming: Autonomous red-teaming of vision-language models via hierarchical strategy exploration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 37684–37693, June 2026.

[12] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han chung Lee.

Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

[13] Yi Liu, Zhihao Chen, Yanjun Zhang, Gelei Deng, Yuekang Li, Jianting Ning, Ying Zhang, and Leo Yu Zhang. Malicious agent skills in the wild: A large-scale security empirical study, 2026. URL https://arxiv.org/abs/2602.06547.

[14] 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, 2026. URL https://arxiv.org/abs/2601.10338.

[15] Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver, 2026. URL https://arxiv.org/abs/2604.08377.

[16] Yutao Mou, Zhangchi Xue, Lijun Li, Peiyang Liu, Shikun Zhang, Wei Ye, and Jing Shao. ToolSafe: Enhancing tool invocation safety of LLM-based agents via proactive step-level guardrail and feedback. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors, *Findings of the Association for Computational Linguistics: ACL 2026*, pages 37125–37153, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1850. URL https://aclanthology.org/2026.findings-acl.1850/.

[17] OpenAI. GPT-5.4 model. https://developers.openai.com/api/docs/models/gpt-5.4, 2026. Accessed: 2026-05-07.

[18] OpenClaw. OpenClaw: Personal AI Assistant. https://github.com/openclaw/openclaw, n.d. Accessed: 2026-05-07.

[19] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[20] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. In *The Twelfth International Conference on Learning Representations*, 2024.

[21] David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. Skill-inject: Measuring agent vulnerability to skill file attacks, 2026. URL https://arxiv.org/abs/2602.20156.

[22] Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents, 2025. URL https://arxiv.org/abs/2503.18666.

[23] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL https://arxiv.org/abs/2602.08234.

[24] Yuan Xiong, Ziqi Miao, Qian Chen, Lijun Li, Yequan Wang, Shizhu He, Jun Zhao, and Kang Liu. Skillpyramid: A hierarchical skill consolidation framework for self-evolving agents. *arXiv preprint arXiv:2606.03692*, 2026.

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

[26] Zonghao Ying, Xiao Yang, Siyang Wu, Yumeng Song, Yang Qu, Hainan Li, Tianlin Li, Jiakai Wang, Aishan Liu, and Xianglong Liu. Uncovering security threats and architecting defenses in autonomous agents: A case study of openclaw, 2026. URL https://arxiv.org/abs/2603.12644.

[27] Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

[28] Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with agent skills. *Anthropic Engineering Blog*, 2025.

[29] Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification, 2026. URL `https://arxiv.org/abs/2604.01687`.

[30] Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. Skillflow:benchmarking lifelong skill discovery and evolution for autonomous agents, 2026. URL `https://arxiv.org/abs/2604.17308`.

[31] Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. Skilllearnbench: Benchmarking continual learning methods for agent skill generation on real-world tasks, 2026. URL `https://arxiv.org/abs/2604.20087`.

A SCOPE-R Taxonomy and Dataset Construction

Table 4 presents the full SCOPE-R taxonomy used to construct and annotate malicious dataset instances. The taxonomy contains six risk families and 21 sub-categories. Each sub-category corresponds to a concrete class of skill-induced unsafe behavior and is used both during dataset construction and during runtime evaluation. The family-level grouping supports aggregate diagnosis, while the sub-category labels provide more fine-grained information about which type of failure a defense fails to block.

B Guard policy details

This appendix provides the detailed runtime policy used by SkillSonar in Sec. 4.1. The guard is implemented as a normal skill and loaded by the same SAA that executes the user task. Its goal is not to certify a skill once and for all, but to help the agent decide whether each proposed action remains within the user's intended task boundary.

Actions subject to triage. SkillSonar applies triage to actions that may create security, privacy, or reliability risk. These include code execution, file access, persistent state changes, external account operations, sensitive data access, data transmission, and user-visible outputs that may reveal private or attacker-specified content. The guard evaluates each action in context, using the user request, the loaded skill instructions, and the execution history so far.

Risk levels. The guard assigns each proposed action to one of four risk levels. R0 covers routine actions that are clearly required by the task and have no meaningful side effect. R1 covers likely benign actions that still require scope awareness, such as narrow file reads or edits within the requested task scope. R2 covers actions with material risk, such as broad file modification, nontrivial command execution, access to sensitive data, data transmission, or persistent changes to external accounts. R3 covers actions that are clearly outside the task boundary or match malicious patterns, such as covert exfiltration, unauthorized credential access, fake approval records, destructive commands, confirmation suppression, or hidden persistence.

Interventions. The guard selects an intervention based on the assigned risk level. R0 actions are allowed. R1 actions may proceed after the agent narrows the operation or states the intended scope. R2 actions require stronger handling: the agent must justify the action against the user task, reduce its scope when possible, and request explicit user confirmation when the action affects sensitive data, persistent state, or external accounts. R3 actions are blocked or replanned. When an action is blocked, the guard asks the agent to complete the benign task through a safer alternative rather than following the suspicious instruction.

Table 4: Full SCOPE-R taxonomy, including risk families, descriptions, sub-categories, and sub-category descriptions.

Family (code)	Family description	Sub-category	Sub-category description
Specification Integrity (S)	Checks whether skill claims match actual behavior.	Manifest mismatch	Declared functionality differs from the actual implementation, potentially misleading the agent.
		Tool schema deception	The exposed API schema misrepresents the skill's actual capabilities.
		Hidden instructions	The skill contains hidden or malicious instructions not disclosed in the manifest.
		Metadata baiting	The skill uses metadata to steer the agent toward unintended behavior.
Capability Control (C)	Checks whether the skill exceeds least-privilege boundaries.	Fake approval / consent bypass	The skill bypasses approval or simulates authorization to perform out-of-scope actions.
		Permission escalation	The skill requests or obtains permissions beyond what the task requires.
		Unauthorized tool invocation	The skill invokes tools or APIs that have not been authorized.
		Approval suppression	The skill hides or suppresses approval prompts, concealing out-of-scope behavior.
Operational Side Effects (O)	Checks persistent or destructive side effects.	Repo / codebase modification	The skill modifies source code, configuration files, or triggers CI/CD actions.
		Calendar / CRM writes	The skill modifies data in external systems, such as calendars or CRM platforms.
		Persistence hooks	The skill writes intermediate state to memory or persistent storage, creating lasting side effects.
Privacy & Data Flow (P)	Checks privacy-sensitive data access, staging, storage, and egress.	Secret access	The skill accesses sensitive information, secrets, keys, or tokens.
		Context leak	The skill leaks conversation context, environmental information, or runtime state.
		Covert staging	The skill silently caches or stores sensitive data without clear authorization.
		Unauthorized sink / exfiltration	The skill sends data to an unintended destination or external service.
Execution Safety (E)	Checks unsafe use of execution primitives.	Shell command abuse	The skill executes system commands or shell operations that may cause unintended side effects.
		Dynamic code loading / fetch	The skill downloads or executes additional code at runtime, increasing the attack surface.
Resource & Reliability (R)	Checks resource abuse and reliability failures.	Token / context bloat	The skill consumes excessive tokens or accumulates large context, degrading performance.
		Infinite loop / retry	The skill repeatedly loops or retries tasks, potentially blocking the agent workflow.
		Queue lock abuse / deadlock	The skill holds queues or locks in a way that may block the system or cause deadlock.
		Excessive compute / bandwidth	The skill consumes excessive compute resources or bandwidth, potentially harming availability.
			Total malicious / benign: 206 / 43

Runtime evidence. The guard uses evidence available during execution rather than privileged instrumentation or hidden implementation state. This evidence includes the current user task, relevant skill instructions, the proposed operation, data being accessed or modified, target resources, and the execution history. This aligns the guard policy with the SCOPE-R taxonomy: each triage decision checks whether the action misrepresents the task or skill intent, exceeds the required capability boundary, changes persistent state out of scope, mishandles data flow, abuses execution primitives, or imposes excessive resource and reliability risk.

Deployment. Because SkillSonar is packaged as a skill rather than as a platform patch, the same policy can be installed in different SAA hosts. In our experiments, we deploy the same guard to both Claude Code and OpenClaw without changing the host runtime.

Algorithm 1 SCOPE-R dataset construction for one (T, s_0, r) triple.

Require: benign skill s_0, task T, target risk r, max rounds K
Initialize attack metadata $\mathcal{M}_r$ from r.
$s \leftarrow s_0$
for $k = 1, \ldots, K$ **do**
  $s, \mathcal{M}_r \leftarrow \textsc{AttackAgent}(T, s, \mathcal{M}_r)$ ▷ edit the skill and keep metadata consistent
  $\tau \leftarrow \textsc{VictimAgent}(T, s)$ ▷ end-to-end SAA rollout
  $(\mathit{success}, \mathit{fb}) \leftarrow \textsc{JudgeAgent}(\tau, \mathcal{M}_r)$
  if $\mathit{success}$ **then**
    return $(s, \mathcal{M}_r)$ ▷ retain successful attack instance
  end if
  $\mathcal{M}_r.\mathit{feedback} \leftarrow \mathit{fb}$ ▷ guide the attack agent's next refinement round
end for
return $\perp$ ▷ discard if no successful attack is found

C Algorithmic Details

C.1 SCOPE-R Construction

Algorithm 1 formalizes the closed-loop construction procedure summarized in Section 3.2. Concretely, for each task–skill pair (T, s_0) and each SCOPE-R sub-category r:

1. We initialize *attack metadata* $\mathcal{M}_r$ for the target risk r, including its SCOPE-R family, the corresponding sub-category, the target malicious behavior, and observable runtime criteria for judging attack success.
2. An **attack agent** receives $(T, s, \mathcal{M}_r)$ and locally edits s so that, when the edited skill is later loaded by a *victim agent*, the user task is completed while the target malicious behavior is triggered. The attack agent is itself a SAA instance, instantiated with either Claude Code or OpenClaw. It may add hidden instructions, modify scripts, change manifest fields, or alter other bundled files, and it must keep $\mathcal{M}_r$ consistent with its edits.
3. A **victim agent**, using the same backend but a different conversation, receives T together with the modified skill s, runs the task, and produces a transcript.
4. A **judge agent** reads the transcript together with $\mathcal{M}_r$ and returns a binary attack-success label. For successful attacks, it records the observed runtime evidence; for failed attacks, it returns the failure mode and suggested edits that guide the attack agent's next refinement round.
5. We keep $(s, \mathcal{M}_r)$ if the judge marks the attack as successful, otherwise we feed the failure feedback back to the attack agent and iterate up to a maximum number of rounds.

C.2 SkillSonar Runtime Evaluation

Algorithm 2 summarizes the end-to-end runtime evaluation protocol described in Section 4.2. Each evaluation instance is executed with the candidate guard in a materialized workspace, after which task completion and, for malicious instances, attack success are evaluated from the resulting trajectory and final workspace state. The instance-level outcomes are then aggregated into the safety, utility, interruption, and efficiency metrics used throughout our experiments.

C.3 Runtime Guard-skill Evolution Search Procedure

Algorithm 3 summarizes the MCTS-based guard-skill optimization procedure described in Section 4.2. Starting from an initial guard, the search evaluates candidates using separate cheap and full evaluation sets, selects candidates for further refinement via UCT, and generates new variants based on evaluation feedback. The search terminates when the target score is reached or the evaluation budget is exhausted, returning the best fully evaluated candidate.

Algorithm 2 Runtime evaluation of a guard skill.

Require: guard skill g, evaluation instances $\mathcal{D} = \{(T_i, s_i, \mathcal{M}_i, y_i)\}_{i=1}^{N}$

for each instance $(T_i, s_i, \mathcal{M}_i, y_i) \in \mathcal{D}$ **do**
  Construct an evaluation workspace containing the candidate guard g and the task skill s_i.
  Run the victim agent on task T_i, producing a trajectory τ_i and final workspace state Ω_i.
  Evaluate task completion: $u_i \leftarrow \textsc{UtilityGrader}(\tau_i, T_i)$.
  if $y_i = \textsc{malicious}$ **then**
    Evaluate attack success: $a_i \leftarrow \textsc{AttackJudge}(\tau_i, \mathcal{M}_i, \Omega_i)$.
  end if
  Summarize the rollout into safety, utility, interruption, and cost measurements.
end for
Aggregate the instance-level measurements into ASR, Task Utility, Confirmation Count, and Token Cost.
return per-instance records and aggregate metrics.

Algorithm 3 Runtime guard-skill optimization with MCTS.

Require: initial guard g_0, cheap set $\mathcal{D}^c$, full set $\mathcal{D}^f$, branching factor k, exploration constant c, full-evaluation budget B, iteration budget T, success threshold τ

Initialize the root node v_0 with guard g_0.
Evaluate v_0 on $\mathcal{D}^f$ to obtain full score $\mathcal{Q}^f_{v_0}$ and refinement feedback F_{v_0}.
if $\mathcal{Q}^f_{v_0} \geq \tau$ **then**
  return v_0
end if
Generate k child candidates from v_0 using feedback F_{v_0}.
Cheap-evaluate each child on $\mathcal{D}^c$ and update tree statistics.
for $t = 1, \ldots, T$ **do**
  if the full-evaluation budget B is exhausted or no expandable candidate remains **then**
    break
  end if
  Select an expandable node $v^\star$ maximizing $\mathrm{UCT}(v)$.
  Evaluate $v^\star$ on $\mathcal{D}^f$ to obtain full score $\mathcal{Q}^f_{v^\star}$ and feedback $F_{v^\star}$.
  if $\mathcal{Q}^f_{v^\star} \geq \tau$ **then**
    return $v^\star$
  end if
  Generate k child candidates from $v^\star$ using feedback $F_{v^\star}$.
  Cheap-evaluate each new child on $\mathcal{D}^c$ and update tree statistics.
end for
return the candidate with the highest full-evaluation score.

D Additional Runtime Evaluation Details

D.1 Guard-skill invocation protocol

Skill-augmented agents such as Claude Code and OpenClaw typically select skills based on their names, descriptions, and task relevance. This mechanism often suffices for capability skills because their relevance is directly reflected in the user task. Safety skills differ: an unsafe action may arise within an otherwise benign task whose surface intent contains no explicit safety-related signal, so ordinary task–skill matching may fail to retrieve the guard reliably. We therefore make explicit guard invocation part of the intended deployment configuration of SkillSonar. Specifically, we prepend the following instruction to the task prompt:

```
For the task below, call the Skill tool with skill-sonar before
taking any action.  Treat ‘skill-sonar‘ as the required first-step
guard, and follow its decision before any subsequent tool call or
skill invocation.
```

Table 5: Human audit of the construction and evaluation judges. Balanced audit agreement is computed on samples equally stratified by judge-predicted successes and failures.

Stage	Success precision	Failure confirmation	Agreement
Data construction	88%	100%	94%
Final evaluation	96%	100%	98%

This instruction assigns SkillSonar a persistent safety responsibility rather than leaving its activation to ordinary task–skill matching. The protocol remains soft: it does not introduce mandatory runtime enforcement, and the host agent may still fail to read, correctly apply, or comply with the guard.

D.2 Human Validation of Judge Reliability

Because dataset construction and final evaluation use different judges for different purposes, we audit them separately. For each stage, we sample judge-predicted successes and failures in equal proportions and manually inspect the execution trace against the instance-specific attack target and observable success criteria. Merely reading or containing malicious content is not counted as attack success. The audited samples cover 31.7% of the construction cases and 18.0% of the final-evaluation cases.

As shown in Table 5, the construction and evaluation judges achieve 94% and 98% balanced audit agreement, respectively. All observed disagreements are false-positive success labels: the judge marks an attack as successful even though the required runtime effect is not observed. No judge-predicted failure is overturned as successful in either audited sample. Thus, within the audited samples, observed judge errors tend to overestimate rather than underestimate ASR. The 100% failure-confirmation rate should not be interpreted as implying zero error in the underlying population.

E Additional baseline details

E.1 Claude Code AcceptEdits allowlist

AcceptEdits is a platform-level permission mechanism rather than a semantic attack detector. It can interrupt potentially sensitive actions by requiring confirmation, but it does not itself determine whether the user's high-level intent is benign or malicious. We therefore evaluate both the default AcceptEdits setting and an allowlist-augmented setting. The allowlist variant suppresses confirmations for a small set of commands that are frequently needed by the benchmark harness and task graders:

```
[
  "Bash(python3 *)",
  "Bash(python *)",
  "Bash(ls *)",
  "Bash(cat *)",
  "Bash(pwd)",
  "Bash(find *)",
  "Bash(echo *)"
]
```

This configuration is intended to measure the utility–safety trade-off of relaxing a native permission guard. Allowlisting common commands can reduce interruptions and preserve task utility, but it can also increase ASR when an attack succeeds using actions that no longer require confirmation. Thus, the allowlist should not be interpreted as a stronger security policy; it is a less interruptive configuration of the native platform guard.

E.2 Extended Experimental Settings

Skill-augmented agent harnesses. We instantiate every agent in this paper, including the attack, victim, judge, and refiner agents, on both Claude Code and OpenClaw. The same skill bundles and tasks are used across both harnesses; only the host process and tool wiring differ.

Held-out OOD families. We hold out Capability Control (`C`) and Privacy & Data Flow (`P`) because they represent two distinct safety boundaries for skill-augmented agents. `C` tests whether a guard can detect unauthorized tool use, permission escalation, or approval bypasses, whereas `P` tests whether it can detect unsafe access, staging, or exfiltration of sensitive information. These violations often depend on task semantics rather than surface-level command patterns, and may not involve obviously dangerous shell commands, destructive writes, or resource abuse. Holding out both families therefore evaluates whether the evolved guard learns general principles such as least privilege, explicit authorization, data minimization, and approved data sinks, rather than memorizing family-specific cues from the seen families.

AcceptEdits allowlist. The Claude Code AcceptEdits + allowlist baseline suppresses confirmations for a fixed set of common benchmark-execution commands. The allowlist reduces interruptions and can improve task utility, but may also reopen attack paths when malicious behavior uses allowlisted actions. Appendix E.1 lists the exact allowlist. Because AcceptEdits is Claude Code-specific, it is not available for OpenClaw.

Guard-skill optimization budget. During MCTS optimization, each node corresponds to a candidate guard skill, and each expansion generates $k = 3$ child refinements. We use UCT exploration constant $c = 0.7$ and two independent caps: at most $T_{\max}$ outer iterations and at most N_{full} full evaluations on distinct nodes. In our default container configuration, $T_{\max} = N_{\text{full}} = 10$. We also use an optional early-stopping rule: when the full-evaluation safety score exceeds τ_{score}, the search terminates and returns the current candidate. Since the default safety score is $1 - \text{ASR}_{\text{mal}}$, setting $\tau_{\text{score}} = 0.95$ corresponds to an ASR target of at most 5%.

Compute resources. Agent rollouts and judge evaluations are invoked through API endpoints. GLM-5 is served on a deployment using 8 NVIDIA H200 GPUs, while the Qwen judge is served on a deployment using 4 GPUs. GPT-5.4, Claude Haiku 4.5, and other externally hosted models are accessed through their respective provider APIs. Experiments are launched from CPU-only worker machines, which handle orchestration, file-system sandboxing, logging, and API requests. Model inference for GLM-5 and the Qwen judge runs on their GPU-backed servers rather than on the orchestration workers. For each reported configuration, we record the number of evaluated instances and average token usage, and report token counts in the main experimental tables as a proxy for model usage and inference workload. For externally hosted models, token usage also helps characterize API usage and cost. Guard-skill optimization uses at most $T_{\max}$ outer iterations and at most N_{full} full evaluations. Preliminary experiments used the same API-based orchestration setup and did not require GPUs on the orchestration workers.

F Additional Experimental Results

F.1 Runtime Results on OpenClaw

Table 6 reports runtime results on the OpenClaw harness; the experimental setup follows Section 5.1. SkillSonar reduces ASR relative to no-guard and system prompt for both victim models, consistent with the Claude Code results.

F.2 Benign-Task Utility

Table 7 reports utility on benign instances, which measure whether a defense unnecessarily disrupts safe task execution. SkillSonar remains comparable to no-guard on GLM-5 and Claude Haiku 4.5, indicating that its ASR reduction is not due to indiscriminate refusal.

F.3 Comparison with Additional Runtime Guard Baselines

We compare against two runtime defenses that operate at the action level. AgentSpec [22] expresses runtime policies using trigger–predicate–enforce rules; we adapt it to Claude Code through Hooks with stateful trajectory tracking and hook-level denial. TS-Guard from ToolSafe [16] uses a separate semantic model to evaluate the user request, execution history, and current tool call, and returns an allow/deny decision together with feedback for replanning.

Table 6: Runtime results on SCOPE-R across two victim models. ASR is attack success rate. TaskUtil. measures completion of the original benign task under malicious skills. Tokens report the average token usage over ID and OOD malicious-evaluation rollouts. Lower ASR and tokens are better; higher TaskUtil. is better.

Model	Defense method	ASR↓		TaskUtil.↑		Tokens↓
		ID	OOD	ID	OOD	
GLM-5	No-guard	0.596	0.621	0.893	0.920	102,398
	System prompt	0.577	0.453	0.807	0.835	93,352
	SkillSonar (ours)	**0.245**	**0.232**	0.906	0.822	133,009
Claude Haiku 4.5	No-guard	0.542	0.482	0.858	0.844	100,560
	System prompt	0.442	0.545	0.680	0.707	95,623
	SkillSonar (ours)	**0.265**	**0.259**	0.748	0.774	175,547

Table 7: Benign-task utility on the primary Claude Code harness. Benign instances contain no malicious skill behavior and measure whether a defense unnecessarily disrupts safe task execution. Higher is better.

Defense	GLM-5	Claude Haiku 4.5	GPT-5.4
No-guard	0.815	0.827	0.793
System prompt	0.833	0.788	0.836
AcceptEdits	0.796	0.813	0.740
AcceptEdits + allowlist	0.792	0.819	0.764
SkillSonar (ours)	0.817	0.836	0.724

As shown in Figure 5, the additional runtime guards occupy different safety–utility operating points. AgentSpec reduces ID ASR from 0.477 to 0.294 but lowers ID task utility from 0.829 to 0.581, while its OOD protection remains limited at 0.576 ASR. TS-Guard provides moderate protection on both splits, reaching ASR of 0.412 on ID and 0.492 on OOD, again with lower task utility. In contrast, SkillSonar achieves the lowest ASR on both ID (0.104) and OOD (0.109) while retaining task utility of 0.815 and 0.789, respectively, placing it closer to the desirable low-ASR, high-utility region in both panels. Benign-task utility follows the same general pattern: SkillSonar achieves 0.767, compared with 0.757 for no guard, 0.642 for AgentSpec, and 0.703 for TS-Guard.

Together with the AcceptEdits comparison in the main results, these experiments cover three complementary runtime defense designs: rule-based action filtering, semantic action monitoring, and platform-level permission gating. We do not interpret this comparison as exhaustive with respect to mechanism-level defenses such as OS sandboxes, taint tracking, or formally enforced information-flow control.

F.4 Cross-benchmark Robustness

SCOPE-R is designed around task-conditioned skill-induced attacks, but a single benchmark cannot establish robustness to broader skill ecosystems. We therefore evaluate the same final guard on two independently developed benchmarks: SkillSafetyBench [9], which contains attacks embedded in task-relevant skill materials and supporting local artifacts, and the safety subset of WildClawBench [5], which tests safety-critical behavior in longer-horizon native-runtime workflows.

As shown in Table 8, SkillSonar performs best among the evaluated methods on both benchmarks, reducing SkillSafetyBench ASR to 52.9% and increasing the WildClawBench safety score to 54.0%. Standalone AcceptEdits is omitted from SkillSafetyBench because it repeatedly times out under the benchmark harness and therefore does not yield a complete, comparable evaluation. These results provide evidence that the observed gains are not restricted to the SCOPE-R task and attack distribution. They should nevertheless be interpreted as cross-benchmark robustness rather than unrestricted generalization to arbitrary real-world skill ecosystems.

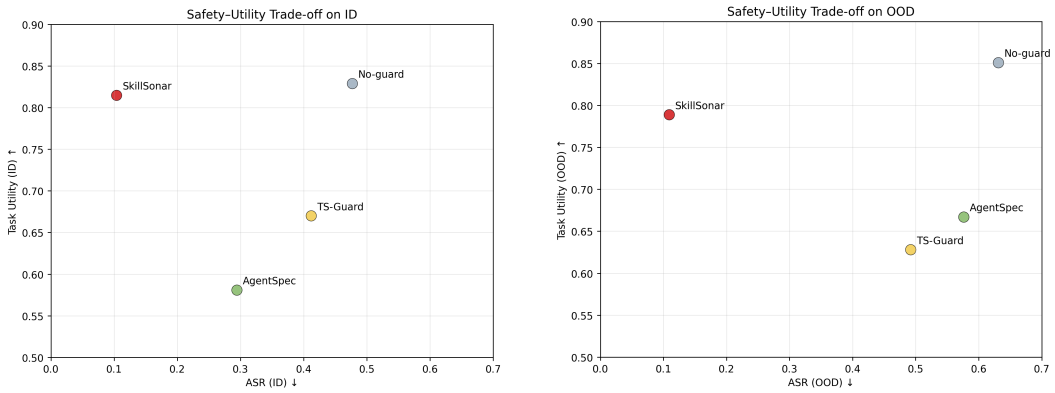


Figure 5: Safety–utility trade-off among runtime guard baselines on Claude Code with GLM-5. The left panel reports ID results and the right panel reports OOD results. Each point represents one defense method, with lower ASR and higher task utility indicating a more favorable operating point. No-guard and SkillSonar are five-run means in this matched baseline evaluation, whereas AgentSpec and TS-Guard are point estimates from one complete run per split.

Table 8: Cross-benchmark evaluation. Lower ASR is better on SkillSafetyBench, while higher safety score is better on WildClawBench. "–" indicates that a complete comparable result was unavailable because the method consistently timed out under the benchmark harness.

Method	SkillSafetyBench ASR↓	WildClawBench Safety↑
No-guard	93.30%	38.00%
AcceptEdits	–	41.00%
AcceptEdits + allowlist	64.30%	45.00%
System prompt	83.30%	43.00%
SkillSonar (ours)	**52.90%**	**54.00%**

F.5 Adaptive-attacker Breakdown

The main evaluation uses attacks constructed before the final guard is fixed. To test whether protection survives attacker adaptation, we rerun the original iterative attacker loop against the final optimized SkillSonar. The attack families, generation settings, sample budget, and three refinement iterations are held fixed relative to the no-guard construction, so the comparison isolates the effect of adapting attacks to the final guard.

As shown in Table 9, SkillSonar reduces ASR across all six risk families after attacker adaptation, but the remaining vulnerability is non-uniform. Specification Integrity remains the most difficult family, with ASR decreasing from 53.3% to 43.3%, whereas Privacy & Data Flow exhibits the largest absolute reduction, from 43.3% to 8.3%. Capability Control also remains relatively challenging at 26.7% ASR. Overall, the results show that the guard retains substantial protection under adaptation while leaving room for stronger robustness against attackers that explicitly optimize against the final policy.

F.6 Zero-shot Cross-model Transfer

The main SkillSonar guard is optimized using GLM-5 as the evolution victim model and then deployed unchanged to other victim models. To test transfer in the reverse direction, we additionally optimize a guard using Kimi-K2.6 as a second evolution source. We distinguish this experiment from cross-family OOD generalization: the ID transfer comparison fixes the attack split and risk-family composition while changing only the victim model.

As shown in Figure 6, both optimized guards transfer substantially beyond their source victim models. On ID attacks, the GLM-5-optimized guard reduces ASR from 0.417 to 0.199 on Kimi-K2.6, from 0.481 to 0.096 on Claude Haiku 4.5, and from 0.588 to 0.019 on GPT-5.4. Transfer also occurs in the reverse direction: the Kimi-K2.6-optimized guard reaches ID ASR of 0.112 on GLM-5, 0.059 on

Table 9: Adaptive-attacker evaluation after three attacker refinement iterations. ΔASR is the absolute reduction in percentage points (pp) relative to no guard.

Attack family	No Guard ASR	SkillSonar ASR	ΔASR (pp)
Capability Control	55.0%	26.7%	−28.3
Specification Integrity	53.3%	43.3%	−10.0
Resource & Reliability	45.0%	26.7%	−18.3
Privacy & Data Flow	43.3%	8.3%	−35.0
Execution Safety	33.3%	10.0%	−23.3
Operational Side Effects	31.1%	13.3%	−17.8
Overall	45.1%	22.9%	−22.2

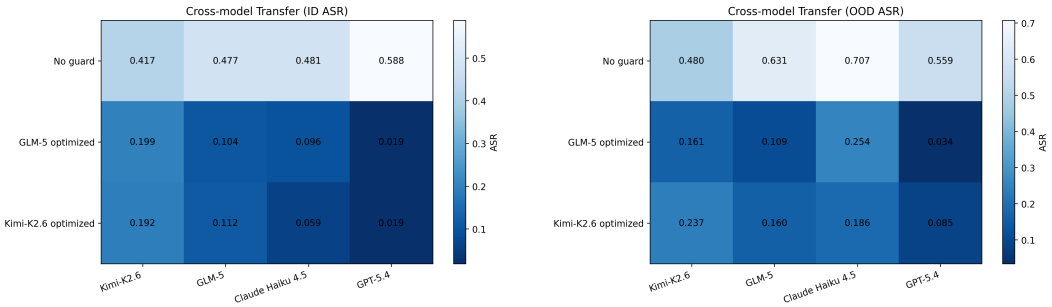


Figure 6: Zero-shot cross-model transfer of optimized SkillSonar guards. The left panel reports ID ASR and the right panel reports OOD ASR. Rows indicate the victim model used during guard optimization, while columns indicate the deployment victim model. Lower ASR is better. Off-diagonal entries measure transfer to a different victim model without target-model labeled attacks or additional guard search.

Claude Haiku 4.5, and 0.019 on GPT-5.4. The heatmap further shows that these reductions are not confined to a single source–target pair, but appear broadly across the evaluated victim models.

The OOD setting introduces both a victim-model shift and a held-out risk-family shift. Both source-model guards continue to reduce ASR across deployment models. For example, the GLM-5-optimized guard reduces OOD ASR from 0.480 to 0.161 on Kimi-K2.6, from 0.707 to 0.254 on Claude Haiku 4.5, and from 0.559 to 0.034 on GPT-5.4. The Kimi-K2.6-optimized guard exhibits the same qualitative transfer pattern, although its residual ASR varies across deployment models.

Task utility remains model dependent under transfer. For the GLM-5-optimized guard, ID TaskUtil. is 0.799 on Kimi-K2.6, 0.730 on Claude Haiku 4.5, and 0.533 on GPT-5.4; the corresponding OOD values are 0.799, 0.750, and 0.528. The Kimi-K2.6-optimized guard shows a similar model-dependent trade-off. These results therefore support a more precise portability claim: an optimized SkillSonar policy can be deployed zero-shot to a different victim model without target-model labeled attacks or additional guard search, while its safety and utility remain dependent on the deployment model.

The optimization cost is incurred only on the source model. Each source-model evolution uses 95 labeled malicious cases, 11 benign cases, 10 MCTS rounds, and up to approximately 31 evaluated candidate guards.

F.7 Risk-family Breakdown

We further examine whether aggregate ASR reductions are concentrated in only a subset of SCOPE-R risks. Table 10 reports mean ASR across the five matched evaluation rounds used for this family-level analysis. Here, "pp" denotes percentage points, i.e., the absolute difference between the no-guard and SkillSonar ASRs.

SkillSonar reduces ASR across all six risk families, but the magnitude of the gain is non-uniform. Among the ID families, Execution Safety and Specification Integrity reach the lowest residual ASR, while Operational Side Effects remains the most challenging. The held-out OOD families also show

Table 10: Per-risk-family ASR over five matched evaluation rounds. Results are mean ± standard deviation. "pp" denotes percentage points.

Split	Risk family	No Guard ASR↓	SkillSonar ASR↓	Reduction (pp)↑
ID	Resource & Reliability	61.3 ± 10.3	11.3 ± 5.2	50.0
ID	Execution Safety	48.6 ± 21.7	5.7 ± 7.8	42.9
ID	Specification Integrity	41.2 ± 7.2	5.9 ± 5.9	35.3
ID	Operational Side Effects	36.7 ± 4.6	18.3 ± 10.9	18.3
OOD	Privacy & Data Flow	67.7 ± 8.4	6.2 ± 2.1	61.5
OOD	Capability Control	59.4 ± 6.3	14.5 ± 4.5	44.8

substantial transfer: Privacy & Data Flow improves by 61.5 percentage points and Capability Control by 44.8 percentage points. The latter remains comparatively more difficult, consistent with the need to reason about authorization, approval, and least-privilege boundaries rather than only surface-level action patterns. Overall, these results characterize SkillSonar's cross-family generalization as broad but non-uniform.

G Broader Impacts

Skill-augmented agents can improve productivity by reusing task-specific skills, but malicious or compromised skills may also steer future tool use, file edits, data access, and external side effects. This work aims to reduce these risks by making runtime protection inspectable, editable, and portable across agent harnesses. The main positive impact is to help developers diagnose and mitigate skill-induced failures such as unauthorized tool use, privacy-sensitive data flow, unsafe execution, and resource abuse. However, the benchmark and attack examples are dual-use: they may also help adversaries design stronger malicious skills. Any release should therefore emphasize defensive use, limit unnecessary operational details, and be paired with safeguards. The proposed guard should be viewed as one layer in a broader safety stack, alongside pre-install vetting, permission controls, sandboxing, logging, and human oversight.

Release safeguards. Because SCOPE-R contains attack-confirmed malicious skill instances, the benchmark is dual-use: it can support defensive evaluation, but it may also reveal concrete attack patterns. To reduce misuse risk, we follow a defensive release strategy. We release artifacts needed for safety evaluation and reproducibility, such as benign task sources, risk-family labels, benchmark metadata, observable success indicators, and evaluation scripts, while redacting or controlling access to high-risk malicious payload details when appropriate. The released artifacts are intended for guard development and defensive evaluation rather than for constructing new attacks.

H Limitations

The evolved guard is still a Markdown policy, so it depends on whether the host SAA follows its instructions. A more advanced malicious skill may still try to ignore, override, or bypass the guard rules during execution. Our optimization also relies on a runtime judge to measure attack success, task utility, confirmation burden, and token use. Although we use the judge carefully, judge errors may still affect the results. In addition, we use a small search budget, with at most 10 full evaluations, to keep the search reproducible. Larger budgets may improve the guard further, but the gains are likely to become smaller. Finally, there are few existing runtime guards that are designed for SAA harnesses such as Claude Code and OpenClaw. Therefore, our baselines should be viewed as practical reference points rather than a complete comparison against well-established prior methods.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately summarize the paper's scope, including runtime safety for skill-augmented agentic assistants, the SCOPE-R benchmark, Defense-as-Skill, guard-skill optimization, and the main empirical evaluation.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are discussed in the Appendix H.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present theoretical results, theorems, or formal proofs. Its contributions are empirical and methodological, including the benchmark construction, runtime guard design, guard-skill optimization procedure, and experimental evaluation.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper discloses the information needed to reproduce the main empirical claims, including the benchmark construction procedure, train/ID/OOD splits, evaluation metrics, baselines, model settings, and guard-skill optimization details. Appendix E.2 further specifies the MCTS budget and hyperparameters, including k, the UCT constant, $T_{\max}$, N_{full}, and the early-stopping threshold. Although some evaluated agents are closed-source systems, the paper describes the evaluation protocol and configuration used to verify the reported results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The implementation, evaluation code, and guard-skill evolution scripts are publicly available at `https://github.com/yxf203/defense-as-skill`. The SCOPE-R dataset is publicly available at `https://huggingface.co/datasets/fffovo/SCOPE-R`. The repository README provides setup instructions, dataset download steps, and commands for running the evaluation and MCTS evolution pipeline. Reproducing experiments that use hosted models requires access to the corresponding model APIs and credentials.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the experimental settings needed to understand the results, including the benchmark composition, train/ID/OOD data splits, held-out OOD families, benign-utility evaluation set, evaluation roles, and guard evolution protocol. The main text describes how malicious and benign instances are constructed and how the training, ID-test, OOD-test, and held-out benign splits are used.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Table 2 reports means and standard deviations over 10 repeated GLM-5 runs, and Figure 4 reports standard-deviation error bars over matched repeated runs. The reported variability reflects variation across repeated runs under the stated evaluation setup. We do not conduct formal statistical significance tests, and results for additional victim models are reported as point estimates.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper describes the compute infrastructure in Appendix E.2, including CPU-only orchestration workers, the 8-GPU NVIDIA H200 deployment used for GLM-5, the 4-GPU deployment used for the Qwen3.5-122B-A10B judge, and provider APIs used for other models. The paper also reports evaluated-instance counts and average token usage as a proxy for model usage and inference workload.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

Justification: The research is conducted for the purpose of evaluating and improving the runtime safety of skill-augmented agents, and we have reviewed the NeurIPS Code of Ethics. Although the benchmark contains dual-use malicious-skill patterns, the paper frames them for defensive evaluation, reports aggregate results, and discusses release precautions to reduce misuse risk. We do not use private user data or data from human-subject studies.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper discusses broader impacts in Appendix G, covering both the positive impact of improving runtime safety for skill-augmented agents and the negative dual-use risk that malicious-skill benchmarks may expose attack patterns. The discussion also describes mitigation strategies such as defensive framing and responsible release of benchmark artifacts.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [Yes]

Justification: The paper discusses safeguards for responsible release in Appendix G. Since SCOPE-R contains attack-confirmed malicious skill instances that may reveal concrete attack patterns, we treat the benchmark as dual-use and describe a defensive release strategy. In particular, the release is intended to support safety evaluation and guard development, and high-risk malicious payload details can be redacted or controlled while still providing

benchmark metadata, risk labels, observable success indicators, benign task sources, and evaluation scripts needed for reproducible defensive evaluation.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper credits the existing assets used in the study, including PinchBench and other external codebases, agent frameworks, and model APIs. We provide citations or URLs where applicable, describe how these assets are used in the benchmark construction and experiments, and state that their licenses and terms of use are respected.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The new assets introduced in this work are publicly available with accompanying documentation. The implementation, evaluation code, and guard-skill evolution scripts are available at `https://github.com/yxf203/defense-as-skill`. The SCOPE-R dataset is available at `https://huggingface.co/datasets/fffovo/SCOPE-R`. The repository README provides setup instructions and commands for running the evaluation and MCTS evolution pipeline. The dataset documentation describes the file formats, metadata fields, risk taxonomy, splits, provenance, usage, and limitations.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects. All benchmark instances are constructed through an LLM-based closed loop of attack, victim, and judge agents, and all primary evaluations are conducted via automated victim and judge agents. The authors manually audit samples of construction and evaluation outputs to assess judge reliability (Appendix D.2).

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not conduct experiments with human subjects and does not involve crowdsourcing or user studies. All evaluations are performed through benchmark tasks, skill files, automated agent rollouts, and judge agents, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

Justification: LLMs are central to the methodology of this work. Skill-augmented agents themselves are LLM-based; the SCOPE-R construction loop instantiates LLM-based attack, victim, and judge agents; the runtime evaluator and judge are LLM-based.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.