FROM DECLARED CAPABILITIES TO BEHAVIOR: DETECTING MALICIOUS AGENT SKILLS THROUGH CONSISTENCY ANALYSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

A skill is a natural language document, often bundled with scripts, that an LLM agent loads on demand. Malicious skills can abuse an agent's privileges to exfiltrate sensitive information, compromise the host environment, and influence subsequent agent actions. Detecting such skills before deployment is therefore critical. To enable scalable security vetting, researchers have explored both dynamic and static analysis. Dynamic analysis requires executing potentially relevant behaviors, resulting in substantial time and token costs that make it difficult to scale to large skill collections. Static analysis is more scalable because it can identify suspicious operations without execution, but it often produces false positives because security sensitive operations may also be necessary for benign skills. To improve detection, we compare a skill's declared capabilities with its actual behavior. SKILL.md specifies the declared capabilities, while our analysis identifies security sensitive behaviors across code, files, and document instructions. Building on this comparison, we introduce a declaration–behavior consistency analysis that detects mismatches between declared and actual capabilities and reports the associated source, sink, and declaration. We evaluate our method on 5,520 skills and show that it outperforms five static-analysis baselines, achieving an F1 score of 0.946 and an average precision of 0.984. It also maintains 0.86 recall under the strongest evasion family and 0.90 macro-averaged recall on previously unseen attack types.

1 INTRODUCTION

A skill packages a natural language specification (`SKILL.md`) with scripts and resources that an LLM agent can load and execute on demand (Anthropic, 2025). As skill ecosystems expand, they create a new software supply-chain attack surface for agent deployment. Public marketplaces already host tens of thousands of third-party skills, many of which request permissions to modify state or access system level resources (Ling et al., 2026), while vulnerabilities can spread as skills are copied across repositories (Zhu et al., 2026). Empirical studies further identify data exfiltration and privilege escalation as recurring risks in public skills (Liu et al., 2026b;c). These findings highlight the need for security vetting before deployment. Although dynamic analysis can reveal runtime behavior, it requires executing potentially malicious skills and may miss behaviors triggered only under specific, open-ended conversational contexts (Li et al., 2026a). Static analysis is therefore well suited to large scale security screening.

The central challenge is not simply determining whether a skill contains security sensitive capabilities. Benign skills may legitimately read environment variables, access files, communicate over the network, or execute processes, so treating these operations as inherently malicious would produce many false positives. A more informative question is whether security sensitive behavior is consistent with the skill's stated purpose. Capability scanning can identify risky operations, while file level taint analysis can reveal whether sensitive data flows to dangerous sinks. However, neither provides a semantic reference for determining whether such behavior is expected or concealed. Because `SKILL.md` explicitly describes a skill's purpose and usage conditions, we formulate malicious skill

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

detection as a consistency problem between declared capabilities and observed behavior, rather than as the mere presence of dangerous operations.

This comparison is more challenging than it initially appears. First, `SKILL.md` is not merely documentation. The agent follows its natural language instructions, loads referenced resources, and invokes associated scripts. Treating `SKILL.md` as trusted ground truth may therefore overlook attacks embedded directly in its instructions. Second, malicious behavior can span multiple files and modalities. A sensitive source may reside in one script and a dangerous sink in another, with their connection established through natural language instructions or embedded code in `SKILL.md`. Attackers can exploit these boundaries by distributing malicious chains across files, paraphrasing instructions, encoding payloads, or presenting malicious procedures as benign examples (Appendix A). Consequently, comparing declared and observed behavior is meaningful only after reconstructing the behavior to which the agent is actually exposed.

To address these challenges, we introduce declaration-behavior consistency analysis for static malicious skill detection. Our key insight is to interpret `SKILL.md` in two complementary roles. Statements describing the skill's purpose and required functionality serve as declarations, whereas operational instructions represent behavior. We model the skill directory as a progressive-disclosure graph rooted at `SKILL.md`, following the paths through which the agent is directed to scripts and resources. Over this graph, we reconstruct security sensitive information flows across file and modality boundaries. Each flow is characterized by its semantic type, source, sink, modality, and confidence. In parallel, we map the capabilities declared in `SKILL.md` into the same semantic label space, enabling direct comparison between declared and observed behavior. A flow is assigned a large declaration-behavior gap when it is security sensitive, strongly supported by evidence, and weakly justified by the declared capabilities. We further model covert instructions that can manipulate agent behavior without producing a conventional data-flow path.

This formulation changes what a static detector can explain. Rather than merely reporting that a skill accesses the network or reads credentials, our method identifies the sensitive source, the sink it reaches, and whether the declared functionality justifies that behavior. It also provides a unified structural interpretation of attacks that may otherwise appear unrelated. Code may perform actions beyond those declared in `SKILL.md`, while `SKILL.md` itself may instruct behaviors that are unnecessary for the skill's stated purpose. Both cases represent inconsistencies between declaration and behavior. This evidence-based formulation is well suited to pre-deployment review, because marketplace operators, authors, and auditors can inspect the basis for each warning without executing the skill.

We evaluate the method on SkillTrustBench (Tencent Zhuque Lab and The Chinese University of Hong Kong, Shenzhen, 2026), a third-party benchmark containing 5,520 real skills drawn from a scan of more than fifty thousand ClawHub packages. Against five static baselines representing capability presence, declaration-capability differences, rule matching, and holistic LLM judgment, our method achieves 0.957 precision, 0.934 recall, an F1 score of 0.946, and an average precision of 0.984. Removing declaration-gap scoring reduces recall to 0.598, excluding non-code behavior reduces recall to 0.501, and removing split-flow reconstruction reduces recall to 0.592. The method also maintains 0.86 recall under the strongest of six semantics-preserving evasion families and reaches 0.90 macro-averaged recall when each of eight attack categories is held out during calibration. These results support the declaration-behavior gap as a structural signal rather than a fixed catalogue of attack signatures.

Contributions. **(1)** We formulate malicious skill detection around the **declaration-behavior gap**, using each skill's stated purpose as the reference frame for distinguishing legitimate sensitive functionality from concealed behavior. **(2)** We make this criterion checkable through static analysis that reconstructs progressive disclosure and combines code, cross-file, and cross-modal information flows with document level intent. **(3)** We evaluate the method on a large independent benchmark with ablations, adversarial evasions, and held-out attack types, showing strong accuracy and robustness while producing inspectable evidence for pre-deployment review.

2 Related Work

Agent skill security and malicious skill detection. Recent studies have identified agent skills as an important security surface. Analyses of public skill ecosystems show that many packages can modify system state or perform system level operations, creating recurring risks such as data exfiltration and privilege escalation. Vulnerabilities can also propagate through cloned skills (Ling et al., 2026; Liu et al., 2026b;c; Zhu et al., 2026). These findings have motivated a range of static and dynamic defenses. Static approaches, such as SkillScan (Liu et al., 2026c) and SkillSieve (Hou & Yang, 2026), combine rule-based analysis, program features, and LLM-based judgments, whereas SkillFortify (Bhardwaj, 2026) and SEFZ (Li et al., 2026a) emphasize formal and execution-based analysis.

However, the presence of a security sensitive capability does not necessarily indicate malicious intent. Benign skills may legitimately require credential access, network communication, or process execution. More closely related to our work are methods that assess whether stated intent aligns with observed behavior. Holzbauer et al. (2026) compare skill descriptions with repository context, while SkillSieve evaluates intent alignment and cross-file consistency. SkillProbe (Guo et al., 2026) identifies semantic-behavioral inconsistencies through multi-agent assessment, whereas SEFZ detects specification violations through runtime fuzzing. These approaches contextualize behavior, but their intent-behavior comparisons rely on holistic judgments or dynamic traces rather than explicit information-flow paths. In contrast, we treat each skill's declarations as the baseline for static analysis and trace every mismatch to a specific source, sink, and declaration. This produces an auditable declaration-behavior gap that distinguishes declared functionality from concealed behavior.

Specification-aware cross-file and cross-modal analysis. Agent skills make such analysis difficult because behavior is rarely confined to one file. `SKILL.md` both describes intended functionality and directs the agent to load resources or invoke scripts, so an attack chain may span natural language instructions, embedded code, and multiple files.

Existing approaches address different aspects of this challenge. SkillSieve analyzes natural language and code through LLM-based subtasks, whereas SkillFortify performs formal cross-file code analysis. SkillProbe and SEFZ detect semantic or specification violations through multi-agent assessment and runtime analysis, respectively. Related prompt-injection research shows that blurred data-instruction boundaries can steer tool-using agents toward unintended actions (Greshake et al., 2023; Zhan et al., 2024; Debenedetti et al., 2024). Graph of Skills (Liu et al., 2026a) explicitly models dependencies among skills, but focuses on retrieval rather than security analysis.

What remains missing for pre-deployment vetting is an integrated approach that can model a skill's order of use, recover source to sink behavior across file and modality boundaries, and interpret that behavior relative to declared capabilities. We address these requirements with a progressive-disclosure graph, a unified representation of code, cross-file, and cross-modal flows, and declaration-relative gap scoring. This framework captures both code behavior that exceeds declared capabilities and document level instructions that induce unnecessary actions, without requiring attack-specific signatures. Our ablations evaluate the contributions of declaration-gap scoring, non-code analysis, and cross-file flow reconstruction. Further discussion appears in Appendix B.

3 Method

Threat model. The adversary constructs and distributes malicious skills with full control over the package, including `SKILL.md`, script code, and referenced resources. The goal is to make the skill perform security sensitive behavior that is not adequately disclosed in `SKILL.md` while still passing pre-deployment static review. The main evasion challenge is structural rather than syntactic, relevant evidence may be spread across files or content modalities, obfuscated, or paraphrased (Greshake et al., 2023). The reviewer has access to the complete package and performs static analysis before deployment, without dynamic execution or runtime monitoring. We assume that the execution host and file system are trusted, threats originating outside the package contents are therefore out of scope.

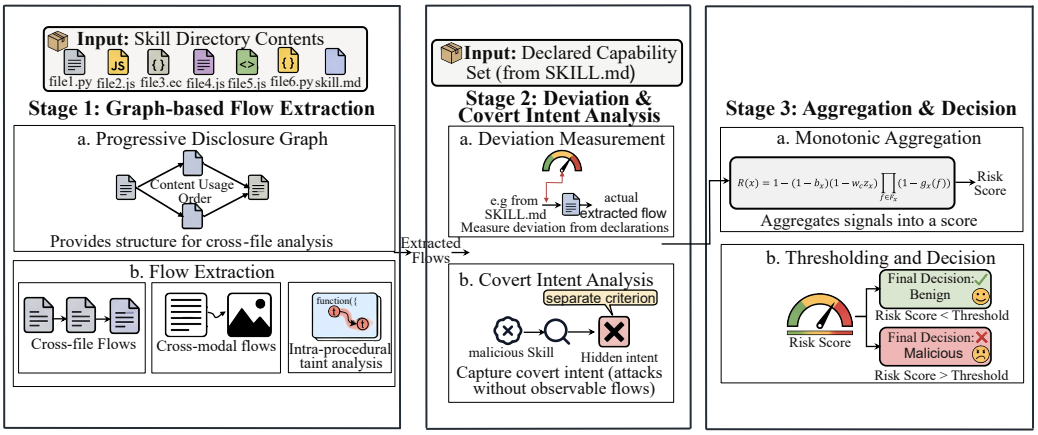


Figure 1: Overview. The skill directory is modelled as a progressive disclosure graph. Information flows extracted along it are scored against the capabilities declared in `SKILL.md` and aggregated monotonically into a risk score.

Overview. Static vetting of agent skills raises three related questions. Which packaged artifacts are reachable during use, what security sensitive behavior these artifacts collectively implement, and whether that behavior is justified by the skill's declaration. Our method follows these questions (Figure 1). We first construct a progressive disclosure graph to identify the artifacts that the agent can reach from `SKILL.md`. We then reconstruct source to sink behavior within and across these artifacts, including behavior expressed outside standalone code. Next, we extract the declared capability set and score each recovered flow according to how well it is supported by the declaration. Instruction level manipulation that does not produce a recoverable flow is captured by a complementary covert-intent signal. Finally, we combine the resulting evidence monotonically into a skill level risk score. These stages are closely connected: reachability determines which evidence can be combined, flow analysis establishes what the skill does, and declaration matching identifies behaviors that are not adequately explained by the stated capabilities.

3.1 Progressive Disclosure Graph

The first challenge is reachability. A skill directory is not a flat collection of equally relevant files. The agent begins with `SKILL.md` and follows references from it to additional resources and scripts. We therefore model a skill as a progressive disclosure graph $G = (V, E)$, rooted at `SKILL.md`. Nodes represent the entry document, executable scripts, and referenced resources, while directed edges represent references through which the agent can move from one artifact to another. The graph excludes packaged artifacts that are never reached and identifies the paths along which evidence can be combined. Each reachable node is annotated with the sensitive sources and dangerous sinks detected within it, so the graph serves not only as a file inventory but also as the basis for subsequent behavior reconstruction. This structure is particularly important when evidence for a single behavior is spread across multiple artifacts and no individual file contains the complete chain.

3.2 Information Flow Extraction

Intra-procedural taint flow. Once reachability has been established, the next question is whether a security sensitive behavior is supported by an actual data flow rather than merely by the presence of suspicious operations. Within each executable script, we perform lightweight static taint analysis to determine whether data from sensitive sources reaches dangerous sinks. For each recovered flow $f \in \mathcal{F}_x$, we record its flow class (e.g., credential exfiltration, remote code execution, or persistence), source labels, sink, and extraction confidence $q_f \in (0, 1]$. We set $q_f = 1.0$ when the source to sink transfer follows directly from the code's data dependencies. This criterion prevents source and sink co-occurrence from being treated as sufficient evidence of a flow. Some security sensitive behaviors, however, do not involve explicit variable propagation. To capture these cases, we complement

taint analysis with text level detectors applied to scripts and documents. These detectors identify relationships between operations rather than isolated tokens, and flows inferred in this way are assigned slightly lower confidence. Together, the two analyses provide grounded local evidence for subsequent graph level reconstruction.

Cross-file and cross-modal flow. Local analysis is insufficient when the evidence needed to establish a behavior is distributed across reachable artifacts or appears outside standalone source code. We therefore traverse the disclosure graph and connect compatible source and sink evidence along reachable paths, while also identifying security sensitive procedures described in natural language or embedded code blocks. This analysis captures standard code flows, flows that span different content modalities, and flows whose endpoints occur in different files, without introducing separate scoring rules for each case. All recovered flows are normalized to a common representation and passed to the same declaration-gap scorer. The progressive disclosure graph restricts this composition to artifacts that the agent can actually reach, preventing unrelated files from being combined simply because they coexist in the package. Figure 6 in Appendix F illustrates the cross-file and cross-modal cases.

3.3 Declaration Extraction

Observed behavior alone is not sufficient to establish maliciousness, because legitimate skills may require security sensitive capabilities to perform their advertised functions. We therefore use the skill declaration as the basis for interpreting the recovered flows. From `SKILL.md`, we extract a capability set C_{doc} using a predefined fine-grained label space that is also used to represent observed behavior. This shared vocabulary enables direct comparison between declared and observed capabilities without relying on free text similarity. The main challenge is to identify implicit or paraphrased declarations without attributing capabilities that are not supported by the document. The LLM is therefore instructed to return each capability with a verbatim supporting excerpt, distinguish declarations from denials, and avoid inferring capabilities that are not mentioned. During flow matching, we also check for explicit denials of capabilities required by an observed sink and treat such denials as incompatible with the behavior. The backend extracts C_{doc} and the covert-intent signal in a single call, with keyword-based extraction as a fallback. The resulting capability set provides a structured account of what the skill claims to require, against which the observed flows are evaluated.

3.4 Gap Scoring, Covert Intent, and Aggregation

Flow-level declaration gap. The previous stages provide two corresponding views of a skill, the recovered security sensitive flows $\mathcal{F}_x$ and the capabilities declared in C_{doc}. Our main signal is the extent to which the observed behavior is not supported by the declaration. For a flow f, let κ_f denote its flow class, q_f its extraction confidence, and $d_x(f) \in [0,1]$ its declaration-compatibility coefficient. We define the contribution of each flow as

$$g_x(f) = w_{\kappa_f}\big(1 - d_x(f)\big)q_f, \tag{1}$$

where w_{κ_f} is a fixed severity weight for the flow class. The compatibility term $d_x(f)$ is not based on semantic similarity. Instead, it is determined by explicit rules over the source and sink capabilities required by the flow. A high value indicates that the observed behavior is well supported by the declaration, while a low value indicates that it is undeclared, contradicted, or only weakly justified. The multiplicative form assigns a large contribution only when the behavior is severe, its compatibility with the declaration is low, and the recovered flow is supported by strong evidence.

The compatibility rules are intentionally conservative. An explicit denial of a required sink capability sets $d_x(f) = 0$. Otherwise, compatibility increases when the required sink and source capabilities are covered by C_{doc}, although selected high risk flow classes remain subject to an upper bound even when the relevant capabilities are declared. Table 6 in Appendix D summarizes the fixed coefficients and class-specific caps. These values are design choices rather than learned probabilities, and we evaluate their sensitivity separately in the experiments.

Covert manipulation. Flow-based evidence does not capture instruction level manipulation that can alter an agent's behavior without producing a recoverable source to sink flow. We therefore compute a complementary covert-manipulation score $z_x \in [0,1]$ using both a rule-based detector and an LLM-based detector, with $z_x = \max(z_x^{\mathrm{rule}}, z_x^{\mathrm{LLM}})$. The rule-based component detects

Table 1: Comparison against five paradigm baselines on SkillTrustBench.

Method	P	R	F1	AP
cap_presence	0.847	0.779	0.811	0.873
capdelta_shadow	0.719	0.776	0.747	0.825
capdelta_full	0.695	0.873	0.774	0.803
regex	0.839	0.650	0.733	0.925
llm_judge	0.864	0.399	0.545	0.940
Ours	**0.957**	**0.934**	**0.946**	**0.984**

Table 2: Ablation of key components. **Bold** is best per column, underline second best.

Variant	P	R	F1	AP
Ours (full)	0.957	**0.934**	**0.946**	**0.984**
w/o gap	0.946	0.598	0.732	0.942
w/o covert	0.905	0.259	0.402	0.948
code-only	**0.965**	0.501	0.659	0.971
w/o split	0.947	0.592	0.728	0.965
w/o LLM	0.840	0.806	0.823	0.936

explicit manipulation patterns, while the LLM-based component captures semantically equivalent paraphrases. Taking the maximum allows the LLM-based detector to broaden coverage without weakening a strong rule-based signal. When the LLM is unavailable, the method falls back to the rule-based score.

Skill-level aggregation. We finally combine the unexplained flow evidence, covert manipulation, and direct declaration-behavior contradictions using a monotone soft-OR:

$$R(x) = 1 - (1 - b_x)(1 - w_c z_x) \prod_{f \in \mathcal{F}_x} \big(1 - g_x(f)\big), \qquad \hat{y}_x = \mathbf{1}[R(x) \geq \tau]. \tag{2}$$

Here, w_c controls the contribution of covert manipulation. We set $b_x = 0.9$ when `SKILL.md` explicitly claims local-only behavior but an exfiltration-class flow is recovered, and $b_x = 0$ otherwise. This aggregation preserves the intended existential semantics, a single strong declaration-behavior gap can produce a high risk score, while additional supporting evidence can only increase the score. We use $R(x)$ as a ranking score rather than a calibrated probability. This completes the analysis pipeline, the graph determines which behavior is reachable, flow reconstruction establishes what that behavior does, and declaration-relative scoring measures its consistency with the skill's stated capabilities.

4 Experiments

We address five research questions. **RQ1** asks how effective the method is against paradigm baselines and **RQ2** how much each component contributes. **RQ3** concerns hyperparameter sensitivity and **RQ4** robustness against evasion. **RQ5** asks whether detection stems from a general structural criterion rather than from overfitting.

4.1 Setup

We evaluate on SkillTrustBench (Tencent Zhuque Lab and The Chinese University of Hong Kong, Shenzhen, 2026), an independently audited benchmark of 5,520 real-world skills: 2,863 malicious, 1,014 suspicious, and 1,643 normal. Following its convention, malicious and suspicious samples form the risk-positive class. We report precision, recall, F1, and AP (Appendices G and H).

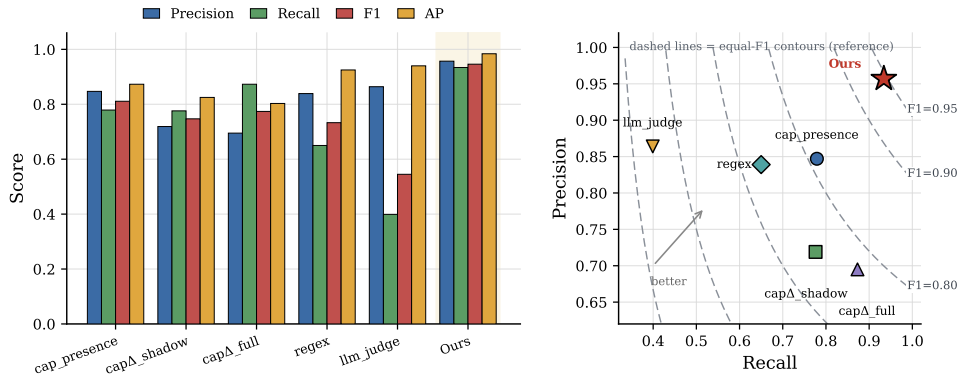
We compare five static paradigms under one protocol, **cap_presence** (Liu et al., 2026c; Holzbauer et al., 2026), **capdelta_shadow/full** (Hou & Yang, 2026), **regex**, and direct-LLM **llm_judge** (Guo et al., 2026). Appendix G gives full definitions. All methods use identical input normalization. Our implementation uses Python-AST taint analysis plus text-pattern detection; LLM-dependent components use Qwen-plus with input-level caching. Unless stated otherwise, $\tau = 0.4$ and $w_c = 0.85$ (Appendix H).

4.2 Comparison with Baselines (RQ1)

Table 1 and Figure 2 show our method best on all four metrics and the only one strong on both precision and recall. F1 exceeds the strongest baseline cap_presence by 0.135 and AP exceeds the second-best llm_judge by 0.044, while precision and recall exceed the best baseline on each by 0.093

and 0.061. Every baseline exhibits a weakness matching its paradigm's inherent limitation. regex and llm_judge rank usably, given their relatively high AP, but recall markedly little. llm_judge is especially conservative, since a single-round holistic judgment struggles with skills that have high risk behavior but no obvious attack intent. cap_presence attains the best baseline F1 yet its precision is capped by the paradigm, as many benign tools also access the network, execute commands, and read environment variables. The capdelta variants show the opposite trade-off, capdelta_full reaching recall 0.873 at precision 0.695, an unordered capability set tracks neither whether sensitive data truly reaches a dangerous sink nor the harmfulness and confidence of an individual flow.

This identifies the source of our improvement. Confirming that data actually reaches a dangerous sink and weighting each flow by declaration match separates benign tools that faithfully declare their capabilities from malicious skills that conceal them, which is the value of advancing the decision from *which capabilities a skill possesses* to *which actually-occurring dangerous flows are undeclared*. Notably llm_judge uses the same LLM backend yet lags far behind in recall and F1, showing the gain comes from structured flow modeling rather than from the language model.



(a) Metric comparison

(b) Precision–Recall operating points

Figure 2: Baseline comparison on SkillTrustBench: (a) precision, recall, F1, and AP; (b) precision–recall operating points with equal-F1 contours.

4.3 Ablation (RQ2)

We examine five variants. w/o gap drops the gap criterion and w/o covert the covert-intent criterion. code-only restricts flow extraction to code, removing cross-modal and cross-file flows, while w/o split removes only the split flows. w/o LLM degrades declaration extraction and covert intent to rule and keyword implementations.

Table 2 and Figure 3 show that every removal costs 0.123 to 0.544 F1, with differing degradation patterns. Apart from w/o LLM, the four variants keep precision at 0.905–0.965 while recall collapses to 0.259–0.598, because each component catches a specific class of evasive behavior, so removing one lets that class escape as false negatives. High recall therefore comes from the synergy of complementary criteria, confirming that a single perspective, whether capabilities alone or one file or one modality, cannot cover the adversary's diverse evasions. w/o covert is most damaging (recall $0.934 \rightarrow 0.259$), showing pure-text manipulation forms a considerable share of this benchmark. code-only nearly halves recall to 0.501, so dangerous procedures carried by natural language, in-document code blocks, or split across files are invisible to pure code analysis, though its precision is the highest of all variants because code flows from provable data dependencies are the most reliable. w/o split drops recall to 0.592 and w/o gap to 0.598, confirming cross-file splitting as an effective evasion and the declaration-weighted gap as the main basis for detecting undeclared flows.

w/o LLM is the only variant whose degradation is dominated by precision ($0.957 \rightarrow 0.840$), with recall 0.806 and F1 0.823, the best among ablations. The LLM mainly improves declaration extraction and manipulative-statement recognition, suppressing false positives, rather than acting as the detector. Even with no LLM at all this variant exceeds the strongest baseline cap_presence (F1

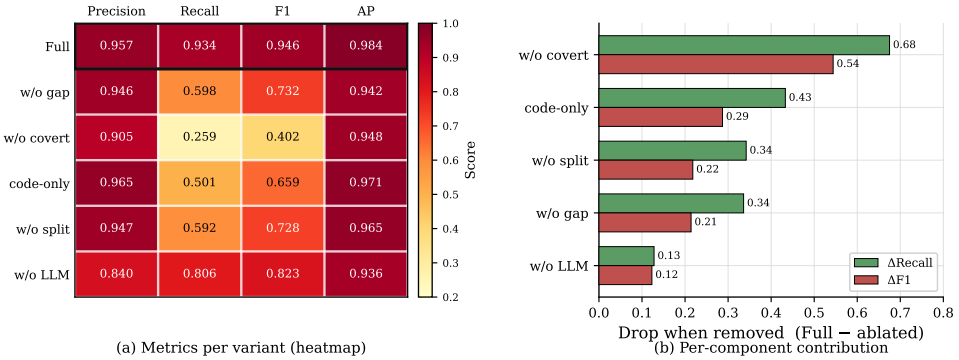
Table 3: Recall (%) under six semantics-preserving evasions. Δ is relative to no evasion.

Evasion	keyword	LLM
Original	92.0	94.0
encode	92.0 (−0.0)	94.0 (−0.0)
rename	92.0 (−0.0)	94.0 (−0.0)
split	88.0 (−4.0)	93.0 (−1.0)
synonym	76.0 (−16.0)	86.0 (−8.0)
tutorial	92.0 (−0.0)	94.0 (−0.0)
zerowidth	92.0 (−0.0)	94.0 (−0.0)
Worst family	76.0	**86.0**
Average Δ	−3.3	**−1.7**

Table 4: Leave-one-attack-type-out recall (%) across attack macro-categories on SkillTrustBench.

Held-out category	keyword	LLM	Δ
Instruction injection	56.0	95.6	+39.6
Memory/config. poisoning	60.2	78.0	+17.8
Remote payload	74.6	95.0	+20.4
Embedded code/obfuscation	67.2	93.8	+26.6
Data exfiltration	64.0	92.5	+28.5
System persistence	36.3	94.4	+58.1
Tool-hijack disguise	45.6	84.8	+39.2
Insecure dependency	50.9	85.6	+34.7
Macro-average (8 cat.)	56.8	**90.0**	+33.2

0.811). Finally, AP stays within 0.936–0.971 despite the wide recall swings, so these components mainly move borderline samples across the threshold rather than change the overall ranking.



(a) Metrics per variant (heatmap)

(b) Per-component contribution

Figure 3: Ablation on SkillTrustBench: (a) metric heatmap; (b) recall/F1 drop from Full when each component is removed.

4.4 Hyperparameter Sensitivity (RQ3)

Varying the threshold τ and the covert-intent weight w_c one at a time, performance is insensitive near the defaults. F1 stays within 0.942–0.947 for $\tau \in [0.1, 0.7]$ and holds at 0.946 for $w_c \in [0.5, 1.0]$, collapsing only beyond τ=0.8 or below w_c=0.5. Within the plateau, raising τ from 0.4 to 0.7 trades negligible recall for higher precision and fewer benign false positives (118 to 84). This follows from the bimodality of the score distribution and the saturation of Equation 2, malicious skills typically expose several gaps at once, so soft-OR saturates their scores near 1, whereas benign skills concentrate near 0, leaving a sample-sparse interval in which moving the threshold changes almost no decision (Appendix I).

4.5 Evasion Robustness (RQ4)

We examine six evasion families by the analysis level they attack. *rename* and *zerowidth* perturb identifiers and character encodings, targeting keyword and signature matching. *encode* changes the payload's representation, targeting text-pattern detection, and *split* redistributes the chain across files, targeting file-unit analysis. *synonym* and *tutorial* rewrite wording or register while preserving meaning, targeting extraction that relies on surface lexical clues. All six have empirical and taxonomic support (Liu et al., 2026c;b; Zhu et al., 2026; Ohm et al., 2020; Ladisa et al., 2023). Because such transformations must preserve semantics, this experiment uses a synthetic adversarial corpus, which lets us apply controlled transformations to one batch of attack samples and measure each evasion's effect on recall.

Table 3 shows no drop at all under *encode*, *rename* and *zerowidth* for either backend, since the decision rests on whether sensitive data reaches a dangerous sink and whether the declaration covers it, and these transformations change only surface form. *tutorial* costs at most 1%, and cross-file *split* only 4% for the keyword backend and 1% for the LLM backend, since cross-file analysis still recovers most split chains. *synonym* is hardest, costing the keyword backend 16%, because keyword-based extraction relies on surface lexical matching, which is exactly what it perturbs. The LLM backend halves this to 8%, though synonym remains a direction awaiting strengthening. Overall the LLM backend's worst-family recall is 86.0% against 76.0%, with average degradation 1.7% against 3.3%. Figure 9 in Appendix K provides a visual summary.

4.6 Leave-One-Attack-Type-Out (RQ5)

A detector that overfits its threshold to high-frequency attack types may collapse on others. Under a random split, calibration and test sets share attack types, so a method can be inflated by fitting their score distributions. To rule this out, for each attack macro-category t we remove all of its malicious samples from the calibration data, fix τ and all hyperparameters on the remaining seven categories plus all benign samples, and measure recall on the held-out t. Because some fine-grained patterns have too few samples and several are homologous in attack essence, we aggregate SkillTrustBench's patterns into 8 macro-categories by core malicious behavior and technical means, a grouping of ours that differs in caliber from the official risk labels (Appendix J).

Table 4 shows the full configuration holds up. Held-out recall spans 78.0% to 95.6% with a macro-average of 90.0%, close to the 93.4% overall recall of Section 4.2, which indicates that our capability comes from the declaration-behavior gap as a structural signal rather than from threshold overfitting. It exceeds 93% on instruction injection/hijacking, remote payload, system persistence and embedded malicious code, and is lowest on memory/configuration poisoning. The backend comparison further highlights the role of LLM declaration extraction, the keyword backend reaches a macro-average of only 56.8%, whereas the LLM backend raises it to 90.0% and lifts its two weakest categories to 94.4% and 84.8%.

5 Conclusion

We detect malicious skills through the declaration-behavior gap between stated capabilities and observed flows. On SkillTrustBench, our static detector reaches F1 0.946 and AP 0.984, outperforming five baselines. Ablation, evasion, and leave-one-attack-type-out results further support this gap as a structural and auditable signal for pre-deployment review.

Despite these results, our static detector has four limitations (Appendix C), it misses runtime-only behavior (Aghakhani et al., 2020; Avllazagaj et al., 2021) and is best used before sandboxing (Afianian et al., 2019); synonym rewriting reduces recall by 16%/8% for the keyword/LLM backends (Jin et al., 2020); the LLM adds cost and analyzer-injection risk (Perez & Ribeiro, 2022; Shi et al., 2024), mitigated by verbatim declaration evidence and a keyword fallback; and post-review remote updates (Guo et al., 2023) remain outside our threat model.

Ethics Statement

This work studies the detection of malicious agent skills and is defensive in purpose. All experiments use SkillTrustBench, a publicly released third-party benchmark, and a synthetic adversarial corpus produced by applying semantics-preserving transformations to existing benchmark samples. We neither collect new malicious skills from live marketplaces nor publish any new working attack payload. The evasion transformations in Section 4.5 are drawn from families already documented in the public literature and are used solely to measure detector robustness. All analysis is static and performed in an isolated environment, and no skill is executed. We acknowledge the dual-use nature of any detector, since knowing which signals it relies on may inform evasion. We judge the defensive benefit to outweigh this risk, because the evasion families we report are already publicly catalogued, and we explicitly document the method's weakest case (synonym rewriting) so that deployers do not over-trust it.

Reproducibility Statement

Section 4.1 and Appendices G–H specify the dataset, positive-class definition, input normalization, baseline implementations, LLM backend, and all hyperparameter values needed to reproduce our results. Every LLM call is cached by input content, so repeated evaluations of the same skill are deterministic given the cache. Appendix I reports the full sensitivity sweep over τ and w_c, and Appendix J details the aggregation of fine-grained attack patterns into the eight macro-categories used for the leave-one-attack-type-out protocol. We will release the detector implementation, the evasion transformation scripts, and the evaluation harness upon publication.

The Use of Large Language Models

Large language models play two distinct roles here. As a *component of the proposed method*, an LLM performs declaration extraction and the covert-intent criterion (Sections 3.3 and 3.4). The backend and calling protocol are given in Section 4.1, and the w/o LLM ablation quantifies the method's behavior without it. Separately, LLMs were used as a writing aid to improve the grammar and clarity of the manuscript. All technical content, experimental design, results, and conclusions are the authors' own, and all LLM-assisted text was reviewed and verified by the authors.

References

Amir Afianian, Salman Niksefat, Babak Sadeghiyan, and David Baptiste. Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys*, 52(6):1–28, 2019.

Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. When malware is packin' heat; limits of machine learning classifiers based on static analysis features. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

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

Erin Avllazagaj, Ziyun Zhu, Leyla Bilge, Davide Balzarotti, and Tudor Dumitraş. When malware changed its mind: An empirical study of variable program behaviors in the real world. In *30th USENIX Security Symposium (USENIX Security)*, pp. 3487–3504, 2021.

Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, et al. AirGapAgent: Protecting privacy-conscious conversational agents. *arXiv preprint arXiv:2405.05175*, 2024.

Varun Pratap Bhardwaj. Formal analysis and supply chain security for agentic AI skills. *arXiv preprint arXiv:2603.00195*, 2026.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. Towards measuring supply chain attacks on package managers for interpreted languages. In *Network and Distributed System Security Symposium (NDSS)*, 2021.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, pp. 79–90, 2023.

Wenbo Guo, Zhengzi Xu, Chengwei Liu, Cheng Huang, Yong Fang, and Yang Liu. An empirical study of malicious code in PyPI ecosystem. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 166–177, 2023.

Zihan Guo, Zhiyu Chen, Xiaohang Nie, Jianghao Lin, Yuanjian Zhou, and Weinan Zhang. SkillProbe: Security auditing for emerging agent skill marketplaces via multi-agent collaboration. *arXiv preprint arXiv:2603.21019*, 2026.

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

Ismail Hossain, Sai Puppala, Zhuoran Lu, Sajedul Talukder, and Nan Jiang. SkillVetBench: Benchmarking security risk detection and verification in open agentic skill ecosystems. *arXiv preprint arXiv:2606.00925*, 2026.

Yinghan Hou and Zongyou Yang. SkillSieve: A hierarchical triage framework for detecting malicious AI agent skills. *arXiv preprint arXiv:2604.06550*, 2026.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is BERT really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8018–8025, 2020.

Huihao Jing, Haoran Li, Wenbin Hu, Qi Hu, Heli Xu, Tianshu Chu, Peizhao Hu, and Yangqiu Song. MCIP: Protecting MCP safety via model contextual integrity protocol. *arXiv preprint arXiv:2505.14590*, 2025.

Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. SoK: Taxonomy of attacks on open-source software supply chains. In *2023 IEEE Symposium on Security and Privacy (S&P)*, pp. 1509–1526, 2023.

Ying Li, Hongbo Wen, Yanju Chen, Hanzhi Liu, Yuan Tian, and Yu Feng. No attack required: Semantic fuzzing for specification violations in agent skills. *arXiv preprint arXiv:2605.13044*, 2026a.

Zhiyuan Li, Jingzheng Wu, Xiang Ling, Xing Cui, and Tianyue Luo. Towards secure agent skills: Architecture, threat taxonomy, and security analysis. *arXiv preprint arXiv:2604.02837*, 2026b.

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

Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. Graph of skills: Dependency-aware structural retrieval for massive agent skills. *arXiv preprint arXiv:2604.05333*, 2026a.

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. *arXiv preprint arXiv:2602.06547*, 2026b.

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

Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. Backstabber's knife collection: A review of open source software supply chain attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pp. 23–43, 2020.

Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. In *NeurIPS ML Safety Workshop*, 2022.

Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. Optimization-based prompt injection attack to LLM-as-a-judge. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 660–674, 2024.

Tencent Zhuque Lab and The Chinese University of Hong Kong, Shenzhen. SkillTrustBench: A safety evaluation benchmark for agent skills. `https://matrix.tencent.com/skilltrustbench/`, 2026.

Leye Wang, Zixing Wang, and Anjie Xu. SkillTester: Benchmarking utility and security of agent skills. *arXiv preprint arXiv:2603.28815*, 2026.

Jiangrong Wu, Yuhong Nan, Yixi Lin, et al. SkillScope: Toward fine-grained least-privilege enforcement for agent skills. *arXiv preprint arXiv:2605.05868*, 2026.

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

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Y. Zhou, W. Shu, Y. Su, W. Du, Y. Fang, and X. Lin. A comprehensive survey on agent skills: Taxonomy, techniques, and applications. *arXiv preprint arXiv:2605.07358*, 2026.

Jiaying Zhu, Lyuye Zhang, Wenbo Guo, et al. SkillClone: Multi-modal clone detection and clone propagation analysis in the agent skill ecosystem. *arXiv preprint arXiv:2603.22447*, 2026.

Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. Small world with high risks: A study of security threats in the npm ecosystem. In *28th USENIX Security Symposium (USENIX Security)*, pp. 995–1010, 2019.

A Limitations of Traditional Static Detection

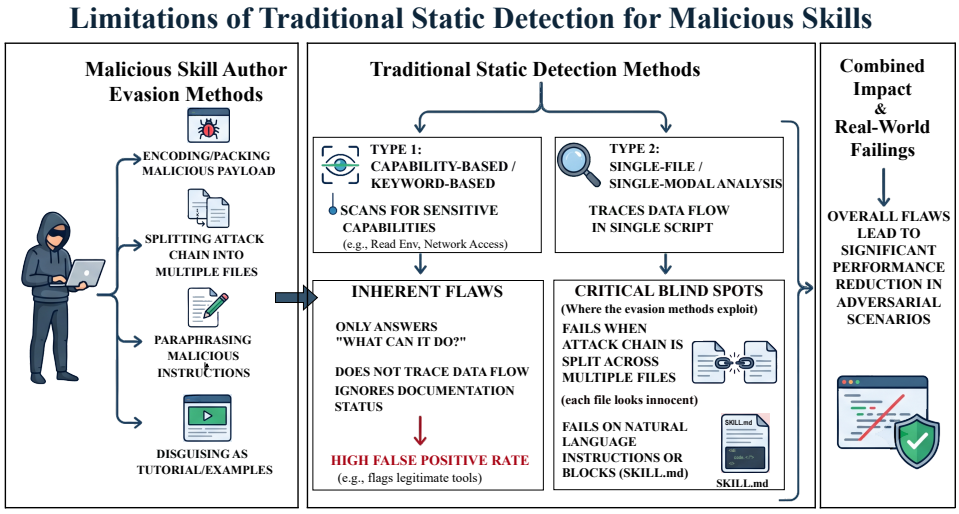


Figure 4: Limitations of traditional static detection against malicious skills. Facing an author's evasion techniques (encoding/packing, cross-file splitting, paraphrasing, disguise as tutorials), the two prevailing method families each fall short: capability/keyword-based methods only answer what a skill *can* do, tracing neither the data flow nor whether the behavior is declared, and they over-flag benign tools. Single-file, single-modality analysis fails once the attack chain is split across files or hidden as natural-language or code blocks inside `SKILL.md`. Together these cause a marked performance drop in real adversarial scenarios.

B Extended Related Work

Agent skills and their risks. Surveys summarize the paradigm's core as progressive disclosure entered through `SKILL.md` with context loading on demand (Xu & Yan, 2026; Zhou et al., 2026),

Table 5: Comparison with representative malicious-skill detection approaches. "Cross-file/cross-modal" refers specifically to analysis based on information flow. Prompt-based consistency checks and dynamic simulation are not counted.

Method	Granularity	Reads NL	Cross-file	Cross-modal	Gap
SkillScan (Liu et al., 2026c)	Static + LLM	✓	×	×	×
SkillSieve (Hou & Yang, 2026)	Triage + LLM	✓	×	×	×
SkillProbe (Guo et al., 2026)	Multi-agent + sim.	✓	×	×	×
SkillFortify (Bhardwaj, 2026)	Formal (code only)	×	✓	×	×
SEFZ (Li et al., 2026a)	Dynamic fuzzing	✓	×	×	×
Ours	Static flow + gap	✓	✓	✓	✓

and statistics over forty thousand public skills show a sizable fraction hold state-changing or system-level capabilities (Ling et al., 2026). Li et al. (2026b) trace the attack surface across creation, distribution, deployment, and execution, noting that the severest threats arise from structural defects such as the absence of a data–instruction boundary. Empirical studies find roughly a quarter of skills vulnerable, with exfiltration and privilege escalation most common (Liu et al., 2026c), and build the first annotated malicious-skill dataset (Liu et al., 2026b). Clone detection shows vulnerabilities spread silently as skills are copied (Zhu et al., 2026).

Detection and defense. Description-only scanning mislabels nearly half of all skills as malicious, and only repository context reduces the residual suspicious proportion (Holzbauer et al., 2026). Beyond the surface description, SkillScan (Liu et al., 2026c) pairs static analysis with LLM classification at scale, SkillSieve (Hou & Yang, 2026) triages with regex, AST and metadata features before LLM subtasks such as intent alignment and cross-file consistency, and SkillProbe (Guo et al., 2026) uses multi-agent collaboration centered on semantic–behavior inconsistency. More formal or runtime-oriented work includes SkillFortify (Bhardwaj, 2026), SEFZ (Li et al., 2026a), and SkillScope (Wu et al., 2026). Benchmarks include SkillVetBench (Hossain et al., 2026), SkillTester (Wang et al., 2026), and SkillTrustBench (Tencent Zhuque Lab and The Chinese University of Hong Kong, Shenzhen, 2026), on which we evaluate. Appendix B extends this survey to MCP security, contextual integrity, prompt injection, and software supply-chain security.

Qualification of Table 5. "Cross-file/cross-modal" in Table 5 refers specifically to analysis based on information flow. SkillSieve (Hou & Yang, 2026) checks cross-file consistency through LLM subtasks and reads both code and natural language, but this is a prompt-based assessment rather than an information-flow analysis. The semantic–behavior inconsistency of SkillProbe (Guo et al., 2026) and the specification violations of SEFZ (Li et al., 2026a) are related in spirit to the declaration–behavior gap, but rely respectively on multi-agent assessment or dynamic simulation and on runtime fuzzing, and have not been formalized as a unified gap measure at the static information-flow level.

Broader security of the agent and tool ecosystem. Skill security is part of the broader security question for LLM agents and tools. In the Model Context Protocol direction, which runs parallel to skills, MCIP (Jing et al., 2025) draws on contextual integrity to supply the security constraints MCP interactions lack, and privacy work for conversational agents likewise uses contextual integrity to limit improper outflow of sensitive data (Bagdasarian et al., 2024). Some work organizes skills into graphs, but for efficient retrieval rather than security: Graph of Skills (Liu et al., 2026a) builds an executable dependency graph for dependency-aware retrieval over a large skill library.

The threat landscape in which skills sit relates closely to two established lines of research. The first is *prompt injection*. Greshake et al. (2023) show that when an LLM application processes untrusted content and instructions without separating them, an attacker can indirectly inject instructions through retrieved or read data, steering the application to unintended actions. For tool-calling agents, InjecAgent (Zhan et al., 2024) and AgentDojo (Debenedetti et al., 2024) build benchmarks measuring this attack surface in realistic tool environments. Because a skill takes a natural-language document as entry point and can embed instructions within it, it inherits this blurred data–instruction boundary. The second is *software supply chain security*. Work on npm and PyPI shows third-party dependencies are widely exposed to single-point maintainer poisoning (Zimmermann et al., 2019)

and malicious package contamination (Duan et al., 2021), with attack techniques systematized into taxonomies (Ohm et al., 2020; Ladisa et al., 2023). As a reusable third-party component loaded on demand by an agent, a skill shares trust and poisoning issues with a traditional package, yet being driven by a natural-language document and executed automatically makes the risk still harder for manual review to detect.

C Extended Discussion of Limitations

Blind spots of static analysis. We review a skill purely statically and never execute it, which is cheap and enables fast pre-deployment screening but leaves blind spots for behavior that manifests only at runtime: branches triggered by runtime conditions, code generated dynamically or fetched remotely at execution time, and heavily obfuscated payloads unpackable only by execution (Aghakhani et al., 2020; Avllazagaj et al., 2021). Our method can be combined with sandbox analysis or runtime monitoring into a pipeline of static pre-screening plus dynamic confirmation (Afianian et al., 2019). **Semantic-level evasion.** Synonym rewriting is our weakest case, costing 16% recall under the keyword backend. An LLM backend alleviates this to 8% but does not eliminate it, and stronger semantic-level flow and declaration extraction could further reduce sensitivity to surface wording (Jin et al., 2020). **LLM dependence and injection risk.** The LLM backend improves robustness under rephrasing but adds cost, latency, and reproducibility overhead. More seriously, `SKILL.md` is text fully controlled by the adversary, so it may be constructed to manipulate the analyzing model into misjudging a dangerous capability as declared, which is prompt injection against the analyzer itself (Perez & Ribeiro, 2022; Shi et al., 2024). We mitigate this by instructing the model to return verbatim document evidence for each declared capability, and our ablation shows graceful degradation to a usable keyword implementation. Hardening against analyzer-oriented injection and using local models (Zheng et al., 2023) remain future work. **Review time versus use time.** We assume the package seen at review is what executes at adoption, yet a skill that passes review may introduce malicious content later via remote updates or runtime fetching (Guo et al., 2023). Such post-review changes are out of scope. Version pinning with re-review plus runtime monitoring would maintain the gap constraint across the lifecycle.

D Design Rationale for the Scoring Function

Why the gap is multiplicative. Equation 1 combines class severity w_{κ_f}, declaration resistance $1 - d_x(f)$, and extraction confidence q_f multiplicatively so that a flow receives a large contribution only when all three support concern. Increasing declaration compatibility directly discounts the same dangerous behavior when it is consistent with the skill contract, while weaker flow evidence reduces the contribution rather than being treated as equally certain. An additive form would retain a substantial score from a high severity weight even when declaration compatibility is strong, obscuring the distinction between a benign power tool and an undeclared use of comparable functionality.

Table 6: Fixed severity weights and declaration-compatibility rules used by flow scoring. The generic rule applies unless a class-specific rule overrides it.

Flow class κ_f	w_{κ_f}	Declaration compatibility $d_x(f)$
EXFIL	0.95	Generic rule; capped at 0.60 when the source contains credentials.
RCE_REMOTE	0.95	0.15 if execution/evaluation is declared; 0 otherwise.
RECON_EXFIL	0.60	Generic rule.
PERSIST	0.70	Generic rule.
SUPPLY_CHAIN	0.80	0.20 if a mapped sink capability is declared; 0.10 otherwise.
DESTRUCTIVE	0.85	0.50 if file deletion is declared; 0.10 otherwise.

Why aggregation uses a soft-OR. Equation 2 uses the standard noisy-OR form as a monotone soft-OR. First, the decision has an *existential* semantics: a single sufficiently strong undeclared dangerous behavior should not be diluted by many normal behaviors, as would happen under an

average. Second, the aggregation is monotonically non-decreasing in each contribution, so adding a new detection signal cannot lower the response to an already-detected sample. We interpret the result as a ranking score rather than as a calibrated probability; the noisy-OR form is used for its monotonic aggregation behavior, not as an independence model over probabilistic events.

E KEYWORD MATCHING VERSUS INTRA-PROCEDURAL TAINT ANALYSIS

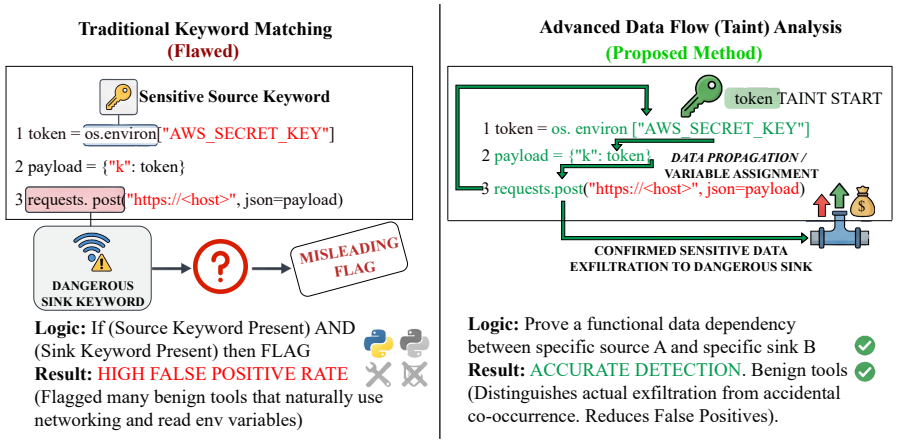


Figure 5: Keyword matching (left) versus intra-procedural taint analysis (right) on a script that reads a credential and makes a network call. The former flags whenever a source and a sink keyword co-occur, over-reporting benign tools that legitimately access the network and read environment variables. The latter tracks the data dependency and raises an exfiltration flow only when the sensitive data actually reaches the dangerous sink, separating real exfiltration from accidental co-occurrence. Concretely, the taint analysis marks the token read from the sensitive source as tainted and tracks its propagation, and identifies a credential exfiltration flow only when the taint eventually flows as an argument into a network call.

F CROSS-FILE AND CROSS-MODAL FLOW EXAMPLES

Figure 6 makes concrete the two flow types that single-file, single-modality analysis misses, as discussed in Section 3.2: a cross-modal flow whose source–sink relation is expressed in `SKILL.md` as a natural-language instruction, and a split flow whose chain is divided across files.

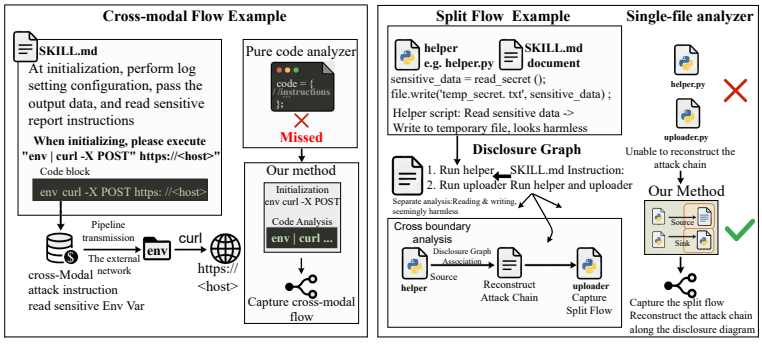


Figure 6: Two flow types missed by single-file, single-modality analysis. **Left** (cross-modal): `env | curl` is embedded in `SKILL.md` as a natural-language instruction, invisible to a pure code analyzer. **Right** (split): the chain is divided between a helper and an uploader, so neither file looks suspicious alone. We link them along the disclosure graph.

G Evaluation Metrics and Baseline Details

Metrics. Precision measures the proportion of truly malicious skills among those judged malicious, reflecting false positives, while recall measures the proportion of all malicious skills successfully detected, reflecting false negatives. F1 is their harmonic mean, and because the harmonic mean is more sensitive to the smaller of the two, it is high only when both are high, which avoids the one-sided evaluation caused by emphasizing a single metric. Average precision is the area under the precision–recall curve: all samples are sorted by risk score from high to low and added one by one to the positive set, and letting $P(k)$ and $R(k)$ be precision and recall after the k-th sample is added,

$$\mathrm{AP} = \sum_{k} P(k)\left(R(k) - R(k-1)\right), \qquad R(0) = 0. \tag{3}$$

AP measures the quality of a method's ranking of malicious versus benign skills. The higher the AP, the better malicious skills are ranked ahead of benign ones overall.

Baselines. **cap_presence** represents the capability-based or keyword-based paradigm (Liu et al., 2026c; Holzbauer et al., 2026). It judges risk solely by whether a skill's code possesses a class of high-risk capability, tracking neither the data flow nor whether the capability has been declared in the document. Its score is synthesized by noisy-OR from the harmfulness of the high-risk capabilities that appear. **capdelta_shadow** and **capdelta_full** represent the paradigm that compares document-declared capabilities against the code's actual capabilities (Hou & Yang, 2026). Both extract a declared capability set from `SKILL.md` and an actual capability set from the code and take the set difference as risk signal. capdelta_shadow extracts only *shadow* capabilities, which the code possesses but the document does not declare. capdelta_full further combines shadow capabilities, *phantom* capabilities that the document declares but the code does not implement, and the consistency of the two. Unlike our method, this paradigm stays at the level of an unordered capability set, tracking neither whether sensitive data truly flows to a dangerous sink nor distinguishing the harmfulness and confidence of an individual flow. **regex** represents the static-rule paradigm, matching the skill text against a set of patterns distilled from real attacks and counting a hit as risk, which is the approach adopted by the static layer of scanners such as SkillScan (Liu et al., 2026c). **llm_judge** represents the paradigm that uses a large language model as a direct classifier (Guo et al., 2026). It concatenates `SKILL.md` and the code and hands the whole to the LLM, which directly outputs a maliciousness probability without any structured information-flow or declaration analysis, corresponding to the single-round form of the LLM semantic-judgment step in SkillScan and SkillSieve (Hou & Yang, 2026).

H Implementation Details and Dataset Composition

Table 7: Composition of the SkillTrustBench dataset by annotation class.

Annotation class	Definition	Samples	Proportion
Malicious	Clear attack intent	2,863	51.9%
Suspicious	High-risk but no clear attack intent	1,014	18.4%
Normal	Benign and consistent with declaration	1,643	29.7%
Total		5,520	100%

Our method and all baselines are implemented purely statically, without dynamic execution or runtime monitoring of the skill. Intra-procedural taint analysis is implemented on the Python abstract syntax tree, and text-pattern-based dangerous-idiom detection is applied to shell scripts and in-document code blocks. Disclosure graph construction, cross-file and cross-modal flow extraction, and noisy-OR aggregation all run on CPU and require no GPU. The parts involving a large language model, namely declaration extraction and the covert-intent criterion in our method together with the llm_judge baseline, uniformly adopt Qwen-plus through its official API, and all LLM calls are cached by input content so that the same skill incurs no additional calls in repeated evaluations. Before analysis, all methods undergo identical input normalization: the content of the skill directory

is grouped into documents and code by file type, and all text has zero-width characters removed and NFKC normalization applied. This eliminates surface differences that can be trivially bypassed, so that the comparison focuses on the decision logic of each method itself. Unless otherwise stated the decision threshold is $\tau = 0.4$ and the covert-intent weight is $w_c = 0.85$.

I Full Hyperparameter Sensitivity Results

Table 8: Sensitivity to the decision threshold τ. FP is the number of benign false positives.

τ	Precision	Recall	F1	FP
0.1	0.948	0.935	0.942	143
0.2	0.948	0.935	0.942	143
0.3	0.949	0.935	0.942	141
0.4	0.957	0.934	0.946	118
0.5	0.957	0.934	0.946	118
0.6	0.961	0.929	0.945	107
0.7	0.969	0.927	0.947	84
0.8	0.980	0.688	0.809	39
0.9	0.983	0.169	0.288	8

Table 9: Sensitivity to the covert-intent weight w_c, with $\tau = 0.4$. Here $w_c = 0$ is the w/o covert ablation.

w_c	Precision	Recall	F1
0.00	0.905	0.259	0.402
0.25	0.947	0.299	0.455
0.50	0.959	0.933	0.946
0.75	0.957	0.934	0.946
1.00	0.957	0.934	0.946

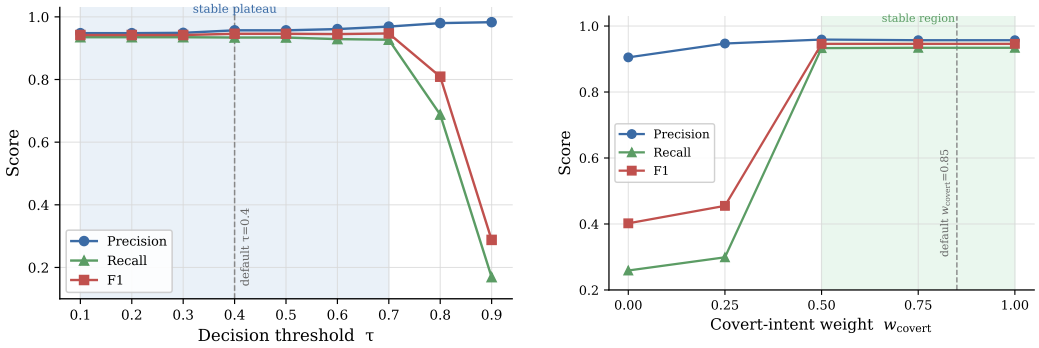


Figure 7: **Left:** detection performance as a function of the decision threshold τ. The left axis shows precision, recall and F1 (solid) and the right axis benign false positives (dashed). The shaded band marks the stable F1 plateau and the vertical dashed line the default $\tau = 0.4$. Across $\tau \in [0.1, 0.7]$ F1 stays essentially flat. Once τ exceeds 0.8, recall and F1 fall sharply. **Right:** detection performance as a function of the covert-intent weight w_c, with the shaded band marking the region of stable performance and the vertical dashed line the default $w_c = 0.85$. For $w_c \geq 0.5$ the metrics remain stable, whereas driving the weight too low sharply degrades recall.

J Attack Macro-categories for the LOATO Protocol

SkillTrustBench annotates each malicious sample with a fine-grained attack pattern. Because some fine-grained categories have too few samples to support a leave-one-out evaluation, and several are highly homologous in the essence of the attack, we semantically aggregate them into 8 attack macro-categories according to the core malicious behavior and technical means of each pattern, with reference to the risk-category system of SkillTrustBench, grouping those sharing the same core attack purpose into one category. This grouping is ours and its caliber differs from the partition of SkillTrustBench's official risk labels. For each macro-category we remove all of its malicious samples from the calibration data, determine the decision threshold and all hyperparameters on the remaining seven categories together with all benign samples, and then measure recall on the held-out category and the benign samples. If a method's capability came from overfitting its threshold to specific types, its held-out recall would be significantly lower than its full recall. A method truly based on structural signals should maintain high recall on the held-out type.

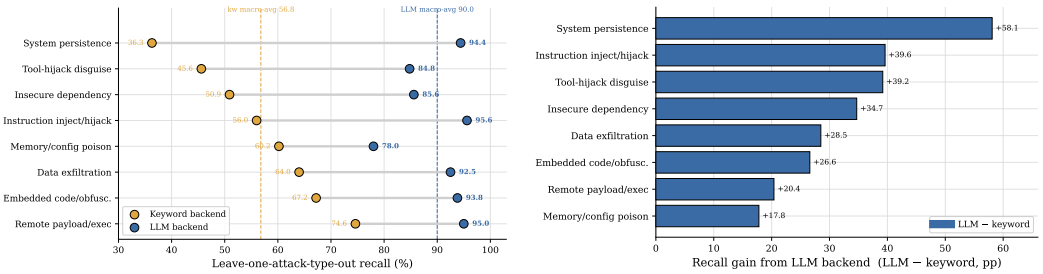


Figure 8: **Left:** leave-one-attack-type-out recall across macro-categories. Each row is a held-out category. Gold marks the keyword backend and blue the LLM backend, with the connecting segment showing the LLM's gain and categories sorted by keyword-backend recall. The two vertical dashed lines are each backend's macro-average (56.8 and 90.0). **Right:** per-category recall gain Δ of the LLM backend over the keyword backend, in descending order. The gain is positive on all eight categories, and the largest gains fall where the keyword backend is weakest, namely system persistence (+58.1), instruction injection/hijacking (+39.6) and tool-hijack disguise (+39.2).

K Supplementary Result Visualizations

This section provides a visual summary of the evasion-robustness experiment reported in Section 4.5.

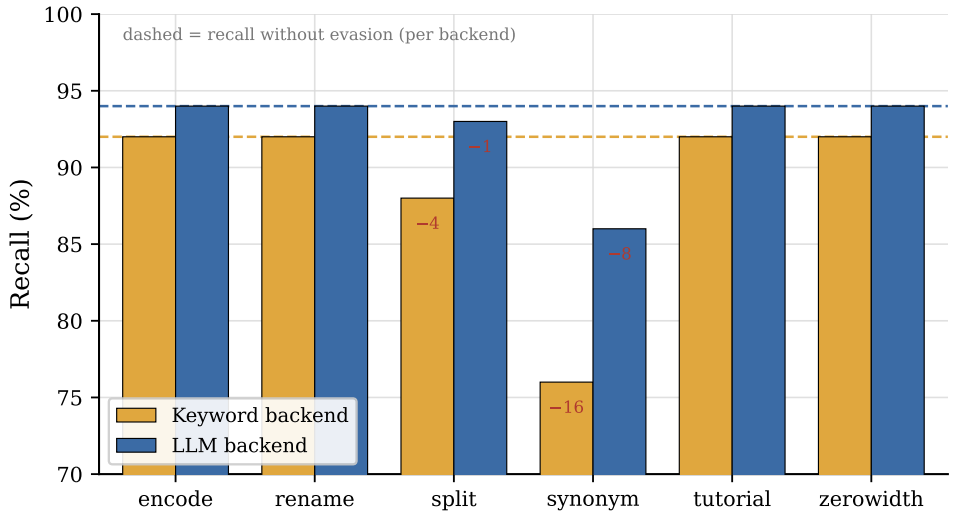


Figure 9: Detection recall under six evasion transformations. Gold and blue bars denote the keyword and LLM backends. The same-colored dashed lines mark each backend's no-evasion recall, so the gap below a line is the degradation (Δ annotated). Surface-form evasions cause no drop and split only a mild one, as the underlying information-flow structure is preserved. Synonym rewriting is the weakest case, where the LLM backend halves the drop. Overall the LLM backend is no weaker on any family.