What Keeps Agent Skills from Being Reusable? Evidence from 138K SKILL.md Files

Chi Zhang
The Graduate Center, City University of New York
New York, NY, USA
CHI.ZHANG06@gc.cuny.edu

Yimin Liu
The Ohio State University
Columbus, OH, USA
yiminliu.career@gmail.com

Xinze Chen
The Graduate Center, City University of New York
New York, NY, USA
xinze.chen95@gc.cuny.edu

Ping Ji
Hunter College, City University of New York
New York, NY, USA
ping.Ji@hunter.cuny.edu

Abstract

Under the current standard, Agent Skills are `SKILL.md` files that combine instructions with supporting resources, enabling Large Language Model (LLM) agents to reuse procedures beyond a single conversation. Yet many public skills appear to originate from a single task, repository, or conversation, even when they are shared as reusable components. We analyze this gap across 138,133 public `SKILL.md` files from 20,556 repositories using a two-tier defect taxonomy grounded in the official specification and best-practice guidance. We find that 91.8% of skills contain at least one detected defect, with stable estimates across lenient and strict thresholds (88.8–94.6%). The dominant failures are ordinary packaging problems rather than exotic attacks: weak routing metadata, bloated or non-actionable bodies, and poor resource organization. A deterministic routing stress test over 20,000 skills shows the functional impact: skills with valid routing metadata are retrieved more reliably from startup descriptions than skills with routing defects. Defect rates vary by platform and provenance: specification-aware skills contain fewer defects, while AI-marked skills show more safety and portability problems. Lightweight enforcement and repair experiments support a quality-assured generation workflow combining spec-aware prompting, lightweight linting, automated repair, and safety gating.

Keywords: Agent Skills, LLM Agents, SKILL.md, Reusability Defects, Skill Routing, Quality-Assured Generation

1 Introduction

Recently, AI skills have become a practical way to extend Large Language Model (LLM) agents beyond the immediate prompt [6]. Under the Agent Skills standard, a skill is packaged as a `SKILL.md` file: frontmatter tells the agent when to load it, the body gives instructions, and optional files provide scripts, references, or assets [2]. In this form, skills turn task experience into reusable software artifacts.

That promise depends on a basic assumption: a published skill should still work outside the task, repository, platform, or conversation that produced it. Our crawl contains 138,133 public `SKILL.md` files from 20,556 repositories, and its growth has outpaced quality control. Many skills appear to be written for a narrow local purpose and then shared as reusable components. Some omit official routing or structure requirements; others inline too much code, include setup notes or changelogs, leak local paths or credentials, or depend on one model, platform, or operating system. These are not just formatting issues. For a skill to be reusable, an agent must be able to select it, load it, locate supporting resources, execute it safely, and port it beyond the author's original environment. Each stage corresponds to one or more categories in our taxonomy, distinguishing our defects from generic style or quality issues.

Existing work captures parts of this problem. SkillsBench [9] shows that curated skills improve task completion by 16.2 percentage points while self-generated skills provide no measurable benefit. SkillReducer [4] shows that non-actionable content wastes context, Wang et al. [16] find malicious skills in public registries, and Liu et al. [11] show that realistic skill retrieval remains brittle. These studies make clear that skill quality matters, but they leave a more basic question open: when a public `SKILL.md` file fails to transfer across tasks or platforms, what exactly has gone wrong? We answer this question by characterizing whether public skills have the artifact properties required for reuse.

We organize the study around four questions:

- **RQ1:** What reusability defects are prevalent in public skills?
- **RQ2:** How do these defects limit skill reuse?
- **RQ3:** Which platform and provenance signals are associated with reusable skill quality?
- **RQ4:** What traits characterize reusable, high-quality Agent Skills?

We make four contributions:

1. **A reusable-skill quality model:** a two-tier taxonomy of 7 categories and 31 checks grounded in official requirements and best-practice guidance.

2. **An ecosystem-scale diagnosis:** an analysis of 138,133 public skills showing that 89.3% violate the official specification and 91.8% contain at least one detected defect.
3. **Functional evidence that defects matter:** a routing stress test showing that R1-clean skills are retrieved more reliably than R1-defective skills from startup descriptions.
4. **A path from diagnosis to intervention:** platform, provenance, exemplar, enforcement, and repair analyses that motivate a quality-assured generation workflow and twelve evidence-based authoring guidelines.

The remainder of the paper is organized as follows. Section 2 reviews the Agent Skills format and related work. Section 3 describes our dataset, defect taxonomy, detector validation, and routing stress test. Section 4 answers the four research questions. Sections 5 and 6 discuss implications and limitations, and Section 7 concludes.

2 Background and Related Work

2.1 Agent Skills and the SKILL.md Format

An Agent Skill is a directory containing a `SKILL.md` file with two parts: YAML (Yet Another Markup Language) frontmatter for structured metadata and a Markdown body for instructions. The frontmatter must include a `name` (max 64 characters, lowercase alphanumeric and hyphens) and a `description` (max 1,024 characters) that specifies *what* the skill does and *when* to use it [2]. The description serves as the primary routing mechanism: agents load skill descriptions at startup to decide which skill to activate for a given task. The body contains instructions the agent follows when the skill is invoked.

The specification recommends keeping the body under 500 lines and organizing supporting content into `scripts/`, `references/`, and `assets/` subdirectories [2]. This *progressive disclosure* pattern loads metadata (~100 tokens) at startup, the body (<5,000 tokens) on activation, and resources only when needed, minimizing context window consumption.

2.2 Related Work

The recent literature has largely established that skills can help agents, but only under the right conditions. SkillsBench [9] makes this contrast especially clear: curated skills improve task completion by 16.2 percentage points, while self-generated skills provide *no measurable benefit*. The same study finds that focused skills with 2–3 modules outperform comprehensive documentation, suggesting that a useful skill is not simply a longer prompt or a larger knowledge dump. Liu et al. [11] reach a similar conclusion from a different angle: when agents must retrieve from 34K real-world skills, the benefit of skills becomes brittle. These findings move the question upstream. Before asking whether a skill improves a task, we need to ask whether the skill is written in a form that can be selected, loaded, and reused outside the context where it was created.

That upstream question turns skills into an artifact-quality problem. A `SKILL.md` file sits between prompt engineering, agent scaffolding, and software packaging: it must be concise enough for progressive disclosure, precise enough for routing, and concrete enough to guide execution. SkillReducer [4] studies one part of this artifact problem by showing that over 60% of body content in public skills is non-actionable and that 26.4% of skills lack routing descriptions. Those results explain why token efficiency and routing metadata matter, but they also point beyond compression. Public skills can fail through body bloat, weak descriptions, poor resource organization, local environment assumptions, unsafe commands, or conflicts with the agent's surrounding instructions. Our work therefore treats skill quality as a multi-stage reusability question rather than a single optimization target.

Agent harnesses make this question more consequential. ClawsBench [10] evaluates LLM productivity agents across realistic Gmail, Slack, Calendar, Docs, and Drive workflows, where domain skills expose API knowledge through progressive disclosure and a meta prompt coordinates behavior across services. In this setting, a skill is no longer a private note saved from one conversation. It becomes part of the interface between the agent, the harness, and external tools. For such an interface to work, a skill needs routable metadata, manageable instructions, organized supporting resources, and enough portability to survive changes in platform, model, or operating environment.

The same interface also raises safety concerns. Wang et al. [16] analyze 150,108 skills from 7 registries and find 620 malicious skills, while Skill-Inject [14] shows that injected skill files can reach up to 80% attack success. These works study deliberately malicious skills, but their threat model highlights a broader issue for public reuse: once a skill is shared, the agent may treat its contents as operational instructions. Even benign skills can therefore be risky if they preserve hardcoded credentials, unsafe shell commands, local paths, or instructions that redefine the agent's role. This connects skill quality to general LLM-agent safety concerns, including OWASP risks such as prompt injection and excessive agency [13], as well as instruction-hierarchy failures where lower-priority instructions can still override higher-priority intent [5, 7, 15].

Tooling has started to address parts of this problem. The `agnix` linter [1] implements 385 validation rules across 6 agent platforms, and the official `skills-ref` validator [2] checks frontmatter syntax. What remains missing is an ecosystem-level account of which defects are actually common, which ones affect functional reuse, and which checks should be enforced before repair or human review. Our study fills this gap by characterizing 138,133 public `SKILL.md` files with a two-tier taxonomy that spans routing, body design, resource organization, prohibited content, behavioral safety, portability, and persona/scope conflicts.

3 Methodology

3.1 Data Collection

To study reuse rather than isolated benchmark performance, we build the dataset from places where skills are actually published and copied. We collected `SKILL.md` files through GitHub code search, repository cloning, and the agentskills.in registry API. GitHub search used 40+ sharded queries to bypass the 1,000-result-per-query limit; repository cloning extracted all case-insensitive `SKILL.md` files from known and discovered skill repositories; registry collection fetched raw GitHub content for indexed skills.

The final SHA-256-deduplicated dataset contains 138,133 unique skills from 20,556 repositories; 98.8% have YAML frontmatter, 98.5% include a `name`, 98.1% include a `description`, and the median body length is 169 lines / 687 words.

Inclusion criteria and representativeness. We applied minimal filtering to preserve an ecosystem-wide view rather than a curated subset. The only filters were (i) the file must be named `SKILL.md` (case-insensitive), (ii) the host repository must be public at crawl time, and (iii) we keep one copy per SHA-256 content hash to remove verbatim duplicates. We did not filter by repository popularity (stars), freshness (last commit date), or author identity, because two questions in this paper—ecosystem-wide prevalence and platform/provenance variation—require including unpopular and stale repositories rather than excluding them. For context, repository-level skew is non-trivial: the most prolific repository contributes 17,284 skills (12.9%) and 47.0% of repositories contribute exactly one skill, so we report both aggregate and per-repository statistics throughout, and we use repository-stratified subsamples for the routing stress test (Section 3.4) and the exemplar analysis (RQ4) to avoid letting one repository dominate. A small enterprise or private-repo bias is unavoidable; we address its external-validity implications in Section 6.

We also collected GitHub issues from 12 major agent platform repositories using 28 skill-related queries; filtering 6,233 issues yielded 761 relevant issues classified by defect category.

3.2 Defect Taxonomy

We organize defects around the lifecycle of reuse. A skill must first be selected, then loaded, then supported by external resources, then executed safely in an environment that may differ from the one where it was authored. We developed the taxonomy through three iterative passes. First, we read the official Agent Skills specification [2] and the Claude Code skills documentation [3] clause by clause, extracting each verifiable structural requirement into a candidate Tier 1 detector. Second, we coded a stratified sample of 300 skills (drawn across platforms and size buckets) for failure patterns that did not map onto a spec clause but recurred in practice—install boilerplate, hardcoded credentials, persona redefinition, platform-specific paths—and clustered them against the peer-reviewed literature on prompt injection, instruction hierarchy, hardcoded secrets, and software portability [5, 8, 12, 15] to form Tier 2. Third, we cross-validated the candidate categories against the 761 GitHub issues we collected (Section 3.1): every category in the final taxonomy is grounded in either a spec clause or an observed real-world failure mode reported by skill consumers. The two tiers are kept separate by design so that compliance violations and best-practice issues can be reported and prioritized independently rather than blurred into a single "defect" bucket. We use the term *detected defect* for a measurable property of a skill file that either violates the official specification or contravenes established best practices, and for which we can point to documented evidence of real-world harm (GitHub issues, benchmark regressions, or security advisories). The taxonomy is designed for static characterization: it identifies skills that may be harder to route, load, maintain, execute safely, or reuse outside their original context. Each rule was operationalized as a regex-based or structural detector applied to the parsed skill.

Tier 1: Spec Conformance (14 checks). These rules are grounded directly in the official Agent Skills specification [2] and Claude Code documentation [3]. Violations affect the core reuse path: missing or non-functional descriptions cause routing failures, oversized bodies increase context cost, and redundant content wastes scarce context tokens.

- **R1. Routing Metadata** (6 checks): missing, too-short, too-long, or non-functional descriptions; routing information misplaced in body.
- **R2. Body Content** (5 checks): oversized bodies, non-actionable content, obvious explanations, name-as-heading duplication, description duplication.
- **R3. Resource Organization** (3 checks): excessive inline code, too many examples, monolithic skills (i.e., large skills that do not use the recommended `scripts/`/`references/` subdirectories).

Tier 2: Best Practice Compliance (17 checks). These rules are derived from peer-reviewed security and software engineering literature.

- **R4. Prohibited Content** (4 checks): install instructions, changelogs, license text, unfinished markers (e.g., `TODO`, `FIXME`) [4].
- **R5. Behavioral Safety** (6 checks): hardcoded credentials, safety bypasses (e.g., `--no-verify`), prompt injection patterns, unguarded destructive actions (e.g., `rm -rf`), error suppression, user path leakage [12, 13, 16].
- **R6. Portability** (4 checks): hardcoded model names (e.g., `gpt-4o`), platform-specific paths, platform-specific tool calls, OS-specific commands (e.g., `pbcopy`) [2, 8].

- **R7. Persona & Scope** (3 checks): persona redefinition (e.g., "you are a…"), instruction overrides, scope mismatches [5, 15].

Figure 1 presents the complete taxonomy as a hierarchical tree. Tier 1 categories (dashed box) are grounded in the official specification; Tier 2 categories are grounded in peer-reviewed literature and industry standards.

The resulting taxonomy contains 31 checks across the seven categories listed above.

3.3 Detector Validation

We validate detectors with mutation-style `SKILL.md` fixtures and threshold sensitivity checks. Each of the 31 detectors has a positive fixture that isolates the target defect and a paired negative fixture; all detectors fire only on the positive case. For numeric thresholds (description length, body length, inline-code ratio, code-block count, monolithic-body length), lenient, baseline, and strict settings on a deterministic 10,000-skill sample preserve the headline result: 88.8–94.6% of skills contain at least one detected defect, and the top three categories remain R1 Routing, R2 Body, and R3 Resources.

3.4 Routing Stress Test

We also run a deterministic retrieval stress test over 20,000 skills. We build a BM25 index over frontmatter descriptions only and ask whether metadata-derived or name/path-derived queries retrieve the source skill. This is not an end-to-end agent benchmark, and we do not claim BM25 mirrors production routing: shipped harnesses such as Claude Code and Cursor use LLM-as-selector over the full description set rather than lexical scoring. We choose lexical retrieval deliberately as a *lower-bound probe*: it is the simplest mechanism for which a routing-defective description (missing, too short, missing trigger language, or non-functional) can plausibly degrade discovery. A more semantic retriever (embedding reranker or LLM selector) may compress or shift the gap, particularly when descriptions are phrased differently than queries; we treat the BM25 result as evidence that routing-metadata defects have *at least* the magnitude observed here at the discovery stage, and we flag the semantic-retrieval condition as future work in Section 6.

4 Results

4.1 Ecosystem Overview

Before examining quality, we first ask what kind of ecosystem these public files represent. The answer matters because a reuse problem has different implications in a small curated registry than in a large, heterogeneous collection of repository-local artifacts.

We infer platforms from file paths. Claude Code is the largest identifiable platform (18.7%), while generic `skills/` directories dominate overall (58.3%). Skills primarily target development workflow automation: debugging (53.1%), documentation (51.0%), API development (48.0%), testing (42.2%), and code review (31.4%). Claude is mentioned in 25.4% of skill content, far exceeding GPT-4/OpenAI (3.4%), Gemini (3.2%), and Copilot (1.4%). At least 14.4% of skills carry explicit AI-generation markers, a conservative lower bound because many generated files are unmarked. The ecosystem is also concentrated: 47.0% of repositories contain one skill, while the largest contributor provides 17,284 skills (12.9%).

4.2 RQ1: What reusability defects are prevalent in public skills?

The first step is prevalence: if detected defects are rare, reusable-skill quality is mainly an edge-case problem; if they are common, quality control must become part of the skill generation and publication workflow. Of 138,133 skills analyzed, 89.3% trigger at least one Tier 1 detector derived from the official Agent Skills specification, and 91.8% have at least one detected defect of any tier. The average skill contains 2.5 detected defects (median: 2). Figure 2 shows prevalence by category. This prevalence result should be read as an ecosystem-level quality signal: some detected defects block routing or create safety risk, while others mostly waste context or indicate weak packaging.

Spec conformance defects dominate. The three most prevalent individual defects are all spec violations: R1.3 missing trigger guidance (52.3%), R2.4 name-as-heading duplication (44.3%), and R3.2 too many inline examples (32.1%). 67.0% of skills have at least one routing defect; combined with R1.4 (13.5%, non-functional descriptions), over half of all skills have descriptions that cannot effectively guide agent routing.

4.3 RQ2: How do these defects limit skill reuse?

Routing defects reduce discovery quality. Prevalence alone does not show whether a defect matters operationally. The most direct reuse failure is that an otherwise useful skill is never selected. Because R1 is both prevalent and tied to startup selection, we test this discovery-stage reuse risk in Table 1. When queries are derived from repository metadata, R1-clean skills achieve 88.5% hit@1 and 0.906 MRR, compared with 82.6% hit@1 and 0.855 MRR for R1-defective skills. The same direction holds under the stricter name/path-only query proxy (26.7% vs. 24.3% hit@1).

What the stress test does and does not show. The stress test isolates one stage of reuse—discovery from the startup description surface—and one defect category, R1. We chose R1 because it is the only category whose harms can be measured by a self-contained retrieval experiment over the artifact: a missing or non-functional description *must* reduce discovery probability, and that effect can be quantified without an external task harness. For R2–R3, the operational impact is context cost and progressive-disclosure friction;

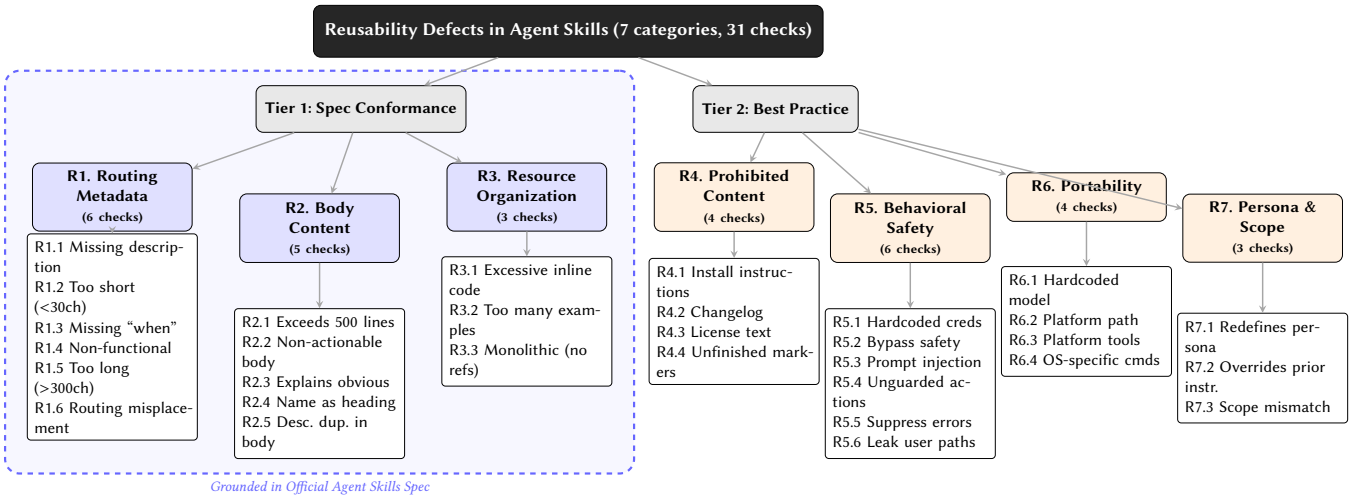

Figure 1. Two-tier defect taxonomy. Dashed box: Tier 1 (spec conformance). Tier 2: peer-reviewed literature and industry standards.

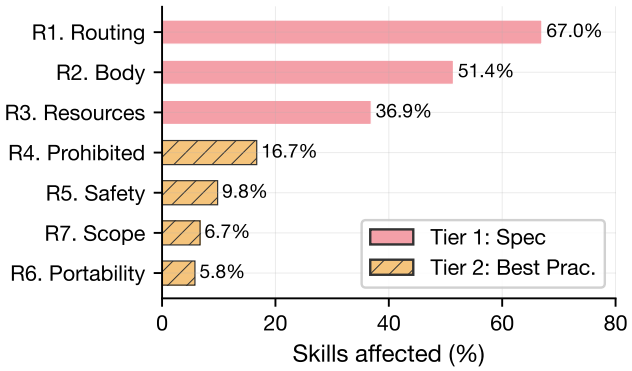

Figure 2. Defect prevalence by category. Tier 1 (red, solid): spec conformance violations. Tier 2 (orange, hatched): best practice violations.

for R4–R6, it is content pollution, execution safety, and portability; for R7, it is conflict with the host agent's instruction hierarchy. Each of these requires either an end-to-end agent benchmark or a deployment harness to measure faithfully, which is outside our scope here. We address the remaining categories through three indirect lines of evidence in the rest of the paper: GitHub-issue corroboration (Section 5), the automated repair experiment for R5 (Section 5), and prior task-success benchmarks that already report sensitivity to body bloat and resource organization [4, 9].

Table 1. Routing stress test over 20,000 sampled skills. The index contains frontmatter descriptions only.

Query	Group	n	Hit@1	MRR
Metadata	R1-clean	6,514	88.5%	0.906
	R1-defective	13,226	82.6%	0.855
Name/path	R1-clean	6,505	26.7%	0.347
	R1-defective	13,331	24.3%	0.322

4.4 RQ3: Which platform and provenance signals are associated with reusable skill quality?

Cross-platform quality. If defects reflect only individual author mistakes, platform-level patterns should be weak. Instead, the data suggest that platform conventions and provenance signals travel with different skill-quality profiles. Figure 3 reveals statistically significant quality variation across platforms (Kruskal–Wallis $H = 594.17$, $p < 10^{-125}$). Cursor skills are highest quality (avg 1.55 defects, 13.5% defect-free), while OpenClaw marketplace skills are lowest (avg 2.29, 3.1% defect-free). Cursor differs significantly from Generic, OpenClaw, and Claude Code (Bonferroni-corrected Mann–Whitney tests, all $p_{\text{adj}} < 10^{-34}$). Three factors correlate with platform-level quality: specification awareness (spec-aware skills average 1.83 defects vs. 3.00 for spec-unaware, Cliff's $\delta = -0.40$), AI-generation rate, and curation model.

AI-marked skills differ from unmarked skills. Figure 4 compares defect rates between the 19,857 skills with explicit AI-generation markers and the remaining 118,276 unmarked skills. We emphasize that this is an *association* between an observable marker and observed defect rates, not a causal claim about AI vs. human authorship. The AI-marked subset is not a random sample of AI-generated skills:

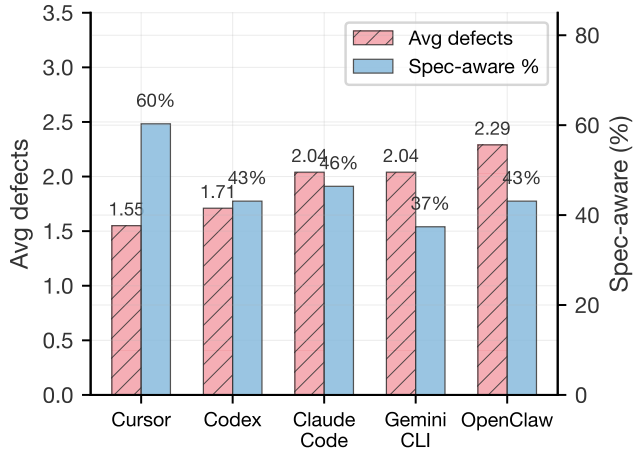

Figure 3. Cross-platform quality: average defects (red) vs. specification awareness (blue).

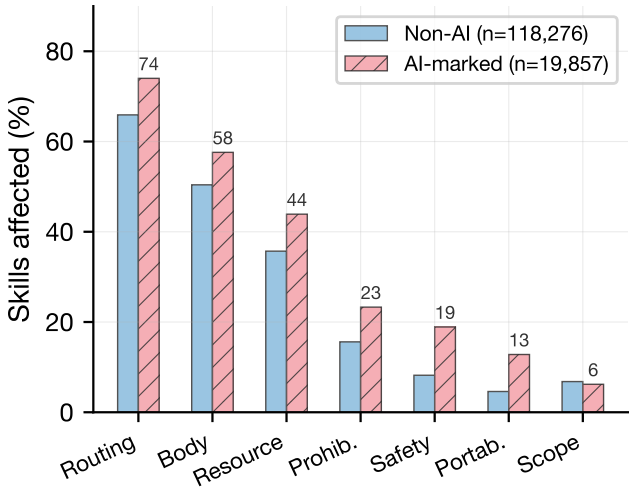

Figure 4. Defect rates by category: AI-marked skills (red) vs. unmarked skills (blue). AI-marked skills show higher detected defect rates across all categories, with the largest gaps in Safety (2.3×) and Portability (2.8×).

only some generators or authors self-disclose, sloppier authors may be more likely to ship the boilerplate marker, and some platforms auto-stamp markers, all of which can produce a confounded comparison. AI-marked skills average 3.23 detected defects versus 2.34 for unmarked skills, a 38.0% increase (Mann–Whitney U, $p < 10^{-300}$; Cliff's $\delta = +0.28$, small effect), and are defect-free at half the rate (5.0% vs. 8.8%). The gap is sharpest for safety-critical categories: 18.9% of AI-marked skills have behavioral safety defects (R5), compared to 8.2% of unmarked skills (2.3×; $\chi^2 = 2{,}209$, $p < 10^{-300}$, $\phi = 0.13$). Portability defects (R6) show a similar pattern (12.8% vs. 4.6%, 2.8×), consistent with generated or conversation-derived skills inheriting platform-specific paths and model references. We read this pattern as a signal that the explicit-marker subset warrants closer review at generation time, not as evidence that AI-authored skills are intrinsically worse; a propensity-matched analysis on length, platform, and repository popularity is left to future work.

Two additional correlates reinforce this pattern. Defect density increases with skill size (Spearman's $\rho = 0.508$, $p < 10^{-300}$): skills over 500 lines average 4.76 defects, compared with 1.48 for skills under 50 lines. Defects also cluster across categories: only 33.0% of defective skills are confined to one category, and safety defects co-occur with portability defects at 2.19× the rate expected by chance.

4.5 RQ4: What traits characterize reusable, high-quality Agent Skills?

The previous results identify failure modes. We next ask what public skills look like when those failure modes are absent. We identified 2,406 skills from repositories with ≥10,000 GitHub stars and analyzed the 419 with zero detected defects. We use these skills as exemplars for qualitative characterization, not as proof that zero detected defects are necessary or sufficient for task success.

These exemplary skills exhibit two distinct styles: *Minimal-and-precise*: React's `verify` skill (24 lines) contains only imperative commands and common mistakes. *Structured-workflow*: Discourse's service-authoring skill (220 lines) defines an eight-phase workflow with gate conditions and audit checklists; every line is actionable.

Five shared traits characterize these exemplary skills:

1. **Trigger-complete descriptions** (G1): every exemplary skill includes "Use when …"
2. **No name-as-heading** (G4): none duplicate their name as an H1 heading.
3. **Imperative throughout** (G5): body text consists of directives, not explanations.
4. **Project-specific knowledge**: skills encode *local* conventions, not generic programming concepts.
5. **No prohibited content** (G8): no install instructions, changelogs, or license text.

We note that traits 1, 2, 3, and 5 are partly definitional under our detectors: a skill with zero detected defects necessarily passes G1, G4, G5, and G8 because those guidelines correspond to the very detectors we use to select the subset. The genuinely non-tautological observation is trait 4 (project-specific knowledge), which we identified by manual reading of the 419 exemplars: across both the minimal and structured-workflow styles, exemplars consistently encode *local* procedural knowledge (specific to a repository, framework, or service) rather than reproducing tutorial content that an LLM could already generate on demand. We

treat this as a qualitative observation worth following up with an operationalized measure (e.g., embedding distance to LLM-generated baselines for the same description) rather than as a quantified finding. The common pattern is not merely absence of lint findings: these skills encode local, actionable procedures with clear activation conditions and little reusable-context waste. This characterization motivates the guideline table below, which combines official structural requirements with empirical and safety-derived constraints.

4.6 Evidence-Based Authoring Guidelines

Based on prevalence, platform variation, issue corroboration, and exemplar analysis, we derive twelve guidelines (G1–G12). Table 2 maps each guideline to target defects and the percentage of skills affected.

5 Discussion

The conversation-first provenance hypothesis. Why are detected defect rates so high despite a clear specification? A reusable skill is not merely a saved prompt; it is a routed, progressively disclosed, executable capability. We propose—and label explicitly as a *hypothesis* rather than a measured finding—that many public skills are instead extracted from problem-solving conversations: a developer solves a task interactively, then saves the conversation as `SKILL.md`. This would explain the recurring co-occurrence of routing metadata weakness, name-as-heading duplication, install instructions, hardcoded paths, model names, and safety bypasses, which together resemble what a chat transcript looks like when serialized to disk. The platform and AI-marker associations are consistent with this hypothesis but do not establish it: the exact authoring path of each public skill is not observable from the file alone, and we did not measure text overlap with chat transcripts or look for explicit "saved from chat" markers, both of which would be required to test the hypothesis directly. We surface it here because it offers a single mechanism that ties together the observed defect cluster and motivates the generation-time interventions in the workflow below; we leave its empirical evaluation to future work.

Specification compliance is necessary but not sufficient. Anthropic's official structure provides the minimal contract for reuse: frontmatter descriptions make skills routable, body-size guidance limits context cost, and resource directories enable progressive disclosure. Our data supports this guidance: spec-aware skills average fewer detected defects, and R1-clean skills perform better in the routing stress test. At the same time, a syntactically valid skill can still be generic, stale, unsafe, or tied to a local environment; conversely, a cosmetic defect such as repeating the name as an H1 heading is less serious than shipping a live API key. This distinction motivates our taxonomy: official-spec conformance (R1–R3) is separated from best-practice and operational risks (R4–R7), and our workflow combines spec-aware prompting with linting, repair, and safety gates rather than treating official formatting as sufficient.

Real-world impact: evidence from GitHub issues. Our issue analysis classified 303 of 761 relevant issues into our R1–R7 taxonomy. Routing defects (R1) are both the most prevalent (67.0%) and most reported (69 issues), followed by body/content issues (75), safety issues (62 plus 108 security-scan disputes), and scope/persona issues (12). Notable examples include skills that never activate despite matching descriptions, 28 plugin skills totaling 34K tokens causing 4+ minute cold starts, and published skills shipping live API keys. Of 108 security-scan false-positive appeals on ClawHub, 63 (58.3%) were triggered by credential-related patterns in legitimate security-auditing skills, confirming that keyword-based scanners cannot distinguish skills that *handle* credentials from those that *leak* them.

Prioritizing defects. Defect-free status is not the goal. Critical defects are routing failures (R1.1–R1.4) and safety violations (R5), because they prevent activation or create vulnerabilities. Important defects such as body bloat and monolithic structure waste context and hinder maintenance. Cosmetic defects such as name-as-heading duplication waste tokens but may not break execution. For practitioners, the priority is eliminating high-impact defects, especially hardcoded credentials, safety bypasses, unguarded destructive actions, and user path leakage.

Progressive enforcement: quantifying the return on investment. We simulated 12 cumulative linter configurations by adding guidelines in order of marginal defect coverage. Figure 5 shows a clear elbow at three guidelines: G1 catches 33.4% of observed defect instances, and adding G4 and G7 raises coverage to **71.9%** while flagging 85.9% of skills. The full suite covers 99.4% of observed defect instances, while adding G9 and G10 raises critical-defect coverage (routing + safety) from 74.7% to 98.6%.

A quality-assured generation workflow. Because self-generated skills provide no average benefit in SkillsBench [9], generation workflows need explicit quality gates. Our findings motivate four stages that correspond to the reuse lifecycle: write routable skills, catch cheap structural defects, repair fixable content, and block unsafe deployment.

Stage 1: Spec-aware prompting. Specification awareness is the strongest quality correlate we identified (Cliff's $\delta = -0.40$, medium effect): skills whose descriptions follow the "*[Verb] [what]. Use when [trigger]*" pattern average 1.83 detected defects vs. 3.00 for spec-unaware skills. Generation prompts should include routing requirements (G1–G3), structural constraints (G4–G7), and content prohibitions (G8).

Stage 2: Lightweight linting. Even spec-aware generation will not eliminate all defects. The three-rule linter (G1,

Table 2. Evidence-based skill authoring guidelines, ordered by impact.

Phase	Guideline	Target Defects	Affected
1. Description	**G1.** Write ≥30 characters: "*[Verb] [what].* Use when *[trigger].*"	R1.2, R1.3, R1.4	52.3%
	G2. Place routing info in `description`, not body.	R1.6	5.2%
	G3. Keep under 250 characters; front-load the use case.	R1.5	14.3%
2. Body	**G4.** Do not repeat `name` as H1 heading.	R2.4	44.3%
	G5. Write imperative directives, not explanatory prose.	R2.2, R2.3	3.2%
	G6. Keep body under 500 lines.	R2.1	10.4%
3. Resources	**G7.** Externalize code (>60%) and examples (>8 blocks).	R3.1–R3.3	37.0%
4. Content	**G8.** Remove install instructions, changelogs, licenses, TODOs.	R4.1–R4.4	16.8%
5. Safety	**G9.** Remove credentials, user paths, `-no-verify` flags.	R5.1, R5.2, R5.6	5.8%
	G10. Require confirmation for destructive actions.	R5.4, R5.5	4.7%
6. Portability	**G11.** No hardcoded models, platform paths, or OS commands.	R6.1–R6.4	5.8%
7. Identity	**G12.** Do not redefine agent persona or override prior instructions.	R7.1–R7.3	6.8%

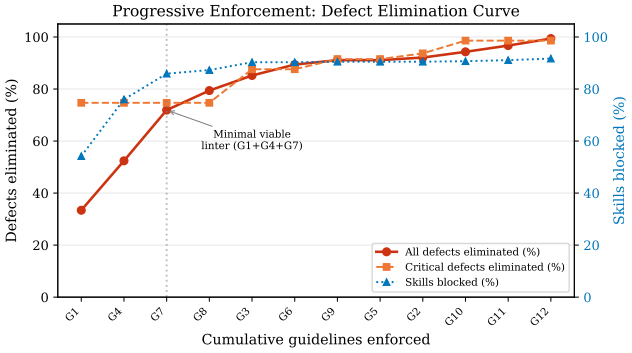

Figure 5. Progressive enforcement: cumulative coverage of observed defect instances as guidelines are added in order of marginal defect coverage (not in the phase-ordered sequence used in Table 2). The elbow at G1+G4+G7 marks the three-rule lightweight linter.

G4, G7) acts as a fast deterministic gate before repair or deployment.

Stage 3: Automated repair. Skills that fail linting enter an LLM-based repair loop. Our repair experiment on 200 defective skills shows that providing the LLM with the detected defect list and relevant guidelines fixes 58.4% of detected defects while introducing only 0.06 new detected defects per skill. Structural defects are repaired reliably, but content-reduction defects remain difficult; heavily defective skills should be regenerated rather than repaired.

Stage 4: Safety gating. In our repair experiment, some safety defects (R5: hardcoded credentials, safety bypasses, prompt injection, unguarded destructive commands) were not repaired reliably, with 0% fix rate for bypass and injection defects. For that reason, safety checks should be non-bypassable gates after repair and before deployment, with R5 subcategories guiding context-aware review beyond keyword matching.

6 Threats to Validity

Construct validity. Our defect detectors use regex-based heuristics, which may produce false positives or false negatives. We mitigate implementation risk through mutation-style fixture tests and threshold sensitivity analysis, but these checks do not capture every semantic context in which a skill may be used. The risk is highest for the two most context-dependent categories. R5.3 (prompt injection) catches lexical patterns that can appear legitimately inside security-auditing or red-team skills that deliberately discuss injection prompts. R7.1 (persona redefinition) catches second-person framing

such as "you are a…" that is appropriate in skills designed to teach an agent a specialized role. A stratified manual precision audit of these safety-critical detectors is the most important construct-validity check we did not run, and we flag it as the first item of follow-up work; the headline 91.8% number is robust to it because R5 and R7 contribute only a small share of the all-category count, but per-category R5/R7 prevalence should be read as upper bounds. The 91.8% figure also bundles defects of very different severity—cosmetic findings such as name-as-heading duplication (R2.4, 44.3%) are counted alongside shipped credentials—so we additionally report the Tier 1 89.3% number throughout and discuss severity prioritization in Section 5 so that the headline is not read as "92% of skills are broken." Ambiguous credential-handling or persona-based skills may still be misclassified, and downstream task success may depend on context beyond static file structure. Treating the official specification as the reference also means that legitimate deviations from the recommended format (narrative-style skills, longer-than-recommended bodies that remain actionable) can be labeled as defects when they may work well in practice; the distinction between Tier 1 (spec) and Tier 2 (best practice) is partly motivated by this asymmetry, and we recommend reading defect counts as static-analysis findings rather than as direct evidence of task-time failure.

Internal validity. Some detected "defects" may be intentional design choices (e.g., persona redefinition for specialized skills). We separate spec violations from best-practice recommendations and report prevalence rather than normative claims. The repair experiment uses GPT-4o-mini and a sample of 200 skills; stronger models may perform differently, and the per-category fix rates (notably the 0% for R5.2 bypass and R5.3 injection) are based on small per-category subsamples that we therefore use only to motivate the safety-gating stage, not to argue precise repair-rate point estimates. Platform and AI-marker analyses identify ecosystem patterns, but the underlying authoring process is not directly observed. The cross-platform comparison (Figure 3) further rests on the 41.7% of skills that file paths allow us to attribute; the remaining 58.3% land in generic `skills/` paths and are excluded from per-platform statistics, so platform-level claims should be read as conditional on the attributable subset.

External validity. Our dataset is limited to public GitHub repositories and the agentskills.in registry; enterprise skills may differ. The top 5 repositories contribute 23.5% of skills; we report both aggregate and per-repository statistics. Platform attribution via file paths is approximate because generic `skills/` paths (58.3%) cannot be attributed to a specific platform.

Functional generalization beyond BM25 and beyond R1. The routing stress test isolates the discovery stage of reuse with a BM25 retriever (Section 3.4); production harnesses use LLM-based selection over the full description set, so the absolute Hit@1 numbers should not be read as production routing performance, and the gap between routing-clean and routing-defective skills may compress under semantic selection. We also do not run end-to-end task benchmarks for R2–R7; their operational impact is supported only indirectly through GitHub-issue corroboration, our repair experiment, and prior work [4, 9, 11]. Extending the stress test with an embedding reranker, an LLM selector, and a harness-level benchmark over the remaining defect categories is a primary direction we intend to pursue in follow-up work.

Reliability. The specification is evolving; rules grounded in the current version may not apply to future revisions. We document the spec version and detection logic for reproducibility.

7 Conclusion

We studied what keeps public Agent Skills from becoming reusable agent capabilities. Across 138,133 public `SKILL.md` files, 89.3% violate at least one official-specification rule and 91.8% contain at least one detected reusability defect under our baseline detector, with the finding stable under lenient and strict thresholds. Routing is the clearest functional bottleneck we isolate: skills with clean routing metadata are retrieved more reliably than those with routing defects from the startup description surface under a BM25 lexical baseline; an LLM-based selector may compress this gap, and we treat lexical retrieval as a lower-bound probe of discovery-stage reuse.

These findings lead to a four-stage quality-assured generation workflow: spec-aware prompting, lightweight linting, automated repair, and safety gating, supported by static analysis, routing stress testing, enforcement simulation, and repair experiments on our dataset. The workflow addresses the practical reality that self-generated skills currently provide little average benefit by embedding routing, structure, resource organization, and safety checks into generation itself.

Data Availability. Our dataset (138,133 skills, CC-BY-4.0) and all analysis scripts are publicly available at https://huggingface.co/datasets/FayeZC/SkillMD-138K.

References

[1] agent-sh. 2026. agnix: The missing linter and LSP for AI coding assistants. Retrieved May 4, 2026 from https://github.com/agent-sh/agnix

[2] Agent Skills. [n. d.]. Specification - Agent Skills. Retrieved May 4, 2026 from https://agentskills.io/specification

[3] Anthropic. [n. d.]. Extend Claude with skills. Retrieved May 4, 2026 from https://code.claude.com/docs/en/skills

[4] Yudong Gao, Zongjie Li, Yuanyuanyuan, Zimo Ji, Pingchuan Ma, and Shuai Wang. 2026. *SkillReducer: Optimizing LLM Agent Skills for Token Efficiency.* arXiv:2603.29919 [cs.SE] doi:10.48550/arXiv.2603.29919

[5] Yilin Geng, Haonan Li, Honglin Mu, Xudong Han, Timothy Baldwin, Omri Abend, Eduard Hovy, and Lea Frermann. 2026. Control Illusion: The Failure of Instruction Hierarchies in Large Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 40. 30816–30824. doi:10.1609/aaai.v40i36.40339

[6] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. 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*. 79–90. doi:10.1145/3605764.3623985

[7] Chuan Guo, Juan Felipe Ceron Uribe, Sicheng Zhu, Christopher A. Choquette-Choo, Steph Lin, Nikhil Kandpal, Milad Nasr, Rai, Sam Toyer, Miles Wang, Yaodong Yu, Alex Beutel, and Kai Xiao. 2026. *IH-Challenge: A Training Dataset to Improve Instruction Hierarchy on Frontier LLMs.* arXiv:2603.10521 [cs.AI] doi:10.48550/arXiv.2603.10521

[8] ISO/IEC. 2023. ISO/IEC 25010:2023 — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Product Quality Model. https://www.iso.org/standard/78176.html

[9] 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. 2026. *SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks.* arXiv:2602.12670 [cs.AI] doi:10.48550/arXiv.2602.12670

[10] Xiangyi Li, Kyoung Whan Choe, Yimin Liu, Xiaokun Chen, Chujun Tao, Bingran You, Wenbo Chen, Zonglin Di, Jiankai Sun, Shenghan Zheng, Jiajun Bao, Yuanli Wang, Weixiang Yan, Yiyuan Li, and Han chung Lee. 2026. *ClawsBench: Evaluating Capability and Safety of LLM Productivity Agents in Simulated Workspaces.* arXiv:2604.05172 [cs.AI] doi:10.48550/arXiv.2604.05172

[11] Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. 2026. *How Well Do Agentic Skills Work in the Wild: Benchmarking LLM Skill Usage in Realistic Settings.* arXiv:2604.04323 [cs.CL] doi:10.48550/arXiv.2604.04323

[12] MITRE Corporation. 2026. CWE-798: Use of Hard-coded Credentials. Retrieved May 4, 2026 from https://cwe.mitre.org/data/definitions/798.html

[13] OWASP Gen AI Security Project. 2025. 2025 Top 10 Risk & Mitigations for LLMs and Gen AI Apps. Retrieved May 4, 2026 from https://genai.owasp.org/llm-top-10/

[14] David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. 2026. *Skill-Inject: Measuring Agent Vulnerability to Skill File Attacks.* arXiv:2602.20156 [cs.CR] doi:10.48550/arXiv.2602.20156

[15] Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. 2024. *The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions.* arXiv:2404.13208 [cs.CR] doi:10.48550/arXiv.2404.13208

[16] Shenao Wang, Junjie He, Yanjie Zhao, Yayi Wang, Kan Yu, and Haoyu Wang. 2026. *"Elementary, My Dear Watson." Detecting Malicious Skills via Neuro-Symbolic Reasoning across Heterogeneous Artifacts.* arXiv:2603.27204 [cs.CR] doi:10.48550/arXiv.2603.27204