An Empirical Study of Agent Skills' Downstream Utility

Yu Cheng
Zhejiang University
Hangzhou, China
yucheng1127@zju.edu.cn

Dehai Zhao
Zhejiang University
Hangzhou, China
dehai.zhao@zju.edu.cn

Zhongxin Liu
Zhejiang University
Hangzhou, China
liu_zx@zju.edu.cn

Qing Huang
Jiangxi Normal University
Nanchang, China
qh@jxnu.edu.cn

Zhenchang Xing
CSIRO's Data61
Australia
zhenchang.xing@data61.csiro.au

Xiaoxue Ren*
Zhejiang University
Hangzhou, China
xxren@zju.edu.cn

arXiv:2610.08875v2 [cs.AI] 8 Oct 2026

Abstract

Agent Skills package procedural guidance and resources for reuse, but a relevant Skill does not necessarily improve task performance. Existing studies characterize Skill content and evaluate downstream performance, yet provide limited explanations of how utility depends on content, execution configuration, and multi-Skill organization. We conduct an empirical study on 87 SkillsBench tasks, defining downstream utility as the pass-rate difference from No-Skill on the same tasks under the same model–harness configuration. We compare the same Skills across nine configurations, then examine alternative published Skills and organizations of fixed Skill sets under three selected configurations. We retrieve marketplace candidates from a curated corpus of 37,596 Skills. LLM-assisted analysis of content, execution traces, and final artifacts, followed by author review, relates provided support to actual use and task outcomes. The same Skills help some configurations and hurt others on 36.78% of tasks, with trajectories showing that recommended procedures can become an execution burden. Relevance rankings overlook more useful candidates. Within the evaluated candidate sets, reranking by support for required operations raises first-choice pass rates by 4.35–5.80 percentage points across the three configurations. We derive 17 authoring practices linking executable procedures to recovery, preservation of task requirements, and checks on final artifacts. Stage Plan and Dependency DAG outperform use order alone, with DAG's additional benefits concentrated in tasks supplied with five or six Skills. These findings guide developers to assess usable operation support, allow procedure adaptation while preserving task requirements, and make artifact dependencies explicit when organizing Skills.

CCS Concepts

• **Software and its engineering → Empirical software validation**.

Keywords

LLM agents, agent harnesses, skill utility, skill organization

1 Introduction

Agent Skills package procedural guidance, scripts, and reference materials as reusable resources that LLM-based agents load during task execution. A Skill typically describes procedures and usage requirements in a `SKILL.md` file and may bundle supporting resources [1]. These packages can capture domain workflows, project-specific rules, or review checklists, giving agents concrete guidance for carrying out tasks [2]. Developers can version, share, and adapt the instruction files and bundled resources for reuse across projects [2, 3]. Public marketplaces distribute tens of thousands of Skills, allowing developers to find relevant Skills and supply agents with third-party procedures instead of having agents rediscover them from scratch [4–6].

Supplying a relevant Skill, however, does not reliably help. Prior evaluations report that Skills can raise task pass rates, have no effect, or even lower them [7–10]. We define the downstream utility of a Skill condition as the difference in task pass rate between that condition and the No-Skill condition, evaluated on the same tasks under the same model–harness configuration. For developers reusing published Skills, task relevance alone is insufficient to judge their utility [11]. Skills with similar purposes can differ in the operations they cover, the resources they provide, and the conditions their procedures require [2]. Their use also depends on how the target model–harness configuration carries out those procedures and how intermediate results are passed between Skills [12–15]. Without understanding how these differences affect task completion, developers have limited guidance for selecting, adapting, or coordinating relevant Skills.

Existing studies examine Agent Skills primarily through content analysis and downstream performance evaluation. Content studies analyze semantic components, authoring practices, and downstream modifications [2, 3, 16]. They identify current and recommended content, but leave unclear whether these practices improve task completion. Execution benchmarks compare pass rates with and without Skills [8, 9]. Pass rates indicate performance changes, but do not show which instructions agents followed, which resources they used, or where they encountered difficulties. In short, *prior work characterizes Skill content and measures task performance, but does not sufficiently explain how downstream utility depends on Skill content, the model–harness configuration, and the organization of multiple Skills.*

Explaining Skill utility through comparisons of published Skills and analysis of their actual use involves two challenges. ***Challenge-1: Useful authoring practices must be grounded in the content and execution of real-world Skills.*** This requires comparing relevant alternatives on the same task and evaluating the same Skills across execution configurations and organization strategies. ***Challenge-2: Differences in Skill content may not be reflected in what agents actually do.*** Agents may skip instructions, adapt

*Corresponding author.

procedures, or supply missing operations, so execution traces and final artifacts are needed to identify the guidance used and execution problems.

To address these challenges, we conduct an empirical study combining real-world Skill comparisons with LLM-assisted analysis. We collect Skills from claude-plugins.dev [17] and ClawHub [18] and filter them for content completeness, adoption, language, and duplication, retaining 37,596 Skills [3]. From this corpus, we retrieve relevant alternatives for tasks supplied with a single Skill and compare their execution using each candidate's original instructions and resources. We then use an LLM to extract and compare task and Skill operations and identify patterns in Skill content, execution trajectories, and final artifacts [2, 13, 16]. Two authors review these patterns against the source evidence, examine alternative explanations, and derive authoring practices from the comparisons [19, 20]. Our study uses 87 SkillsBench tasks and their 232 supplied Skills, with three LLMs and three agent harnesses, and addresses the following research questions [8].

RQ1. How does the downstream utility of the same Skills vary across model–harness configurations? Aggregate gains do not show whether a Skill's benefit transfers across configurations. We cross three models with three harnesses and compare execution with and without the benchmark-supplied Skills in all nine configurations, at both the overall and the task level. We also use the results to select one configuration per model for RQ2 and RQ3.

RQ2. How does downstream utility differ among Skills targeting the same task capability? Relevance-based retrieval supplies candidates, but whether relevance predicts utility is untested. We compare the benchmark-supplied Skill with the top five relevance-ranked marketplace candidates on the 23 single-Skill tasks to avoid confounding from multi-Skill coordination. We decompose task requirements and Skill procedures into *atomic operations*—basic actions on concrete objects with identifiable outcomes—and rerank candidates by their support for these operations. Finally, comparisons of candidate content, trajectories, and deliverables within candidate groups yield authoring practices grounded in observed utility differences.

RQ3. How do Skill organization strategies affect downstream utility in multi-Skill tasks? To isolate the effect of organization, on the 41 tasks supplied with at least three Skills we keep the Skill set and resources fixed and compare execution without organization instructions (Flat) with three schemes that add progressively richer information: a use order (Sequence), stage objectives (Stage Plan), and artifact dependencies (Dependency DAG). Trajectories show how intermediate results are passed, checked, and revised across Skills.

Our analysis exposes actionable findings for adapting, selecting, authoring, and organizing Skills. **For utility across configurations,** Skills help when their procedures match what the target model–harness configuration can execute and adapt (Finding 1). The same Skills yield gains under some configurations and losses under others on 36.78% of tasks. Trajectories show that recommended procedures can become an execution burden when agents struggle to implement or adapt them. **For utility differences among Skills,** task relevance can mask gaps in the operations a Skill supports (Finding 2a). Lower-ranked candidates outperform the first choice in 23.19% of task–configuration groups. Assessing operation support improves selection among relevant candidates (Finding 2b). Reranking the same candidates raises first-choice pass rates by 4.35–5.80 percentage points across three configurations. Skill authoring for utility links executable procedures to recovery and checks on the final artifact (Finding 2c). From content and execution comparisons, we summarize 17 authoring practices, including preserving justified acceptance criteria when changing methods and specifying which changes require rechecking final artifacts. **For the effects of multi-Skill organization,** Stage Plan and Dependency DAG yield greater downstream utility than Sequence across all three configurations (Finding 3). DAG's additional gains over Stage Plan concentrate in tasks supplied with five or six Skills, reaching 3.03–12.12 percentage points. Trajectories show that intermediate artifacts support source checking and propagation of corrections.

With the findings above, we can connect practices in real-world Skills to downstream utility and distinguish the guidance Skills provide from what agents actually use. These results offer developers concrete criteria for selecting Skills with usable operation support, adapting procedures while preserving task requirements, and coordinating Skills through explicit artifact dependencies. There is a need for tools that identify unsupported operations, assess suitability for target model–harness configurations, and track which downstream operations and checks are affected by artifact changes. Finally, there are opportunities for researchers to investigate representations and evaluation methods that connect Skill content, execution behavior, and task outcomes. The identified authoring practices provide concrete starting points for testing which content revisions improve utility and whether their benefits transfer across tasks and configurations.

The main contributions of this study are as follows.

- We address the gap between Skill content analysis and downstream performance evaluation by jointly examining published Skills, execution traces, and final artifacts. This analysis distinguishes supplied support from agents' actual use and relates these differences to task outcomes.
- We systematically analyze how downstream utility varies across model–harness configurations, relevant Skill alternatives, and organizations of the same Skill set. Within-task comparisons reveal the limits of relevance-based selection and quantify the benefits of operation-support reranking and organization through stage objectives and artifact dependencies.
- We derive a five-stage framework of 17 authoring practices grounded in the content and execution of real-world Skills, addressing the need for utility-informed authoring guidance. We discuss their implications for Skill development, selection, coordination tools, and future empirical evaluation.

2 Background and Related Work

2.1 LLM Agents and Agent Skills

Agent Skills are read and used by LLM agents during task execution. Their utility depends on how models interpret the instructions and on the tools and control mechanisms provided by the execution framework. LLMs such as GPT, DeepSeek, and Qwen generate action decisions from task instructions and environmental feedback [21, 22], following the interleaving of reasoning and

action exemplified by ReAct [23]. An *agent harness* manages context, tool calls, and execution control. SWE-agent and OpenHands investigate agent interface design and general-purpose software execution platforms, respectively [24, 25]. For example, Codex and OpenCode support coding tasks through file operations and command execution [26, 27]. OpenClaw connects models to external tools and services for general-purpose tool use and automation [28]. Skill-Use shows that the same model can exhibit different Skill-use behavior across harnesses [12]. Skill utility thus depends on the model–harness pairing, which we call a *model–harness configuration*.

Agent Skills provide reusable procedural knowledge through instructions and supporting resources. Under the Agent Skills specification, each Skill contains a required `SKILL.md` file comprising YAML metadata and a Markdown body, optionally accompanied by scripts, references, and assets. Progressive disclosure first exposes the name and description for discovery, loads the body upon activation, and accesses supporting resources as needed [1]. This structure separates information for Skill selection, instructions for task execution, and resources available during use. Public skill marketplaces enable developers to distribute and reuse these materials [4, 5]. SkillNet further provides infrastructure for Skill creation, evaluation, and relational organization [6]. Procedural knowledge reuse also appears in experience-based learning [29, 30], including natural-language insights in ExpeL [31], executable programs in Voyager and ASI [32, 33], and task workflows in Agent Workflow Memory [34]. Our focus is on published Skills with procedural instructions and supporting resources, specifically their downstream utility when supplied to agents.

2.2 Skill Retrieval and Organization

Skill retrieval research examines how to select relevant support for a task from a candidate library. Related tool reuse research has established key challenges: general-purpose retrievers struggle with specialized tools [35], and structured documentation improves matching [36]. Skill-specific approaches push further. SkillRet introduces a large-scale retrieval benchmark and finds that Skill-specific training outperforms general-purpose retrievers [37]. SkillRouter shows that names and descriptions alone omit useful routing information and reports task-performance gains from its full-text retrieval and reranking pipeline [38]. Yet SRA finds that retrieving a relevant Skill does not ensure appropriate decisions about loading and use [11], highlighting a gap between retrieval quality and downstream performance. We compare observed utility within a fixed, relevance-filtered candidate set to assess whether operation support helps candidate selection.

Structured representations and compilation further characterize the operations and execution conditions required by Skills. SSL separates scheduling information, execution structure, and evidence of actions and resource use, applying this representation to Skill discovery and risk assessment [16]. SkVM decomposes Skill requirements into primitive capabilities, assesses support from model–harness configurations, and adapts execution through compilation and runtime mechanisms [13]. SkCC uses an intermediate representation to separate Skill semantics from platform-specific formats for deployment across frameworks [39]. Such representations provide a basis for extracting comparable operations and constraints from natural-language content. We adopt this operation decomposition to analyze support for task requirements, while using the original Skill content and resources for execution.

Multi-Skill organization research examines how selected capabilities are connected [40]. Graph of Skills incorporates dependencies into retrieval to return Skill sets within a context budget [41]. SkillDAG uses typed edges to represent dependencies, conflicts, and other relationships, allowing agents to update the graph using execution evidence [42]. GraSP converts Skill sets into graphs with precondition and effect relationships, combining node verification with local repair during execution [15]. AgentSkillOS organizes discovery through a capability tree and coordinates execution through a DAG, reporting higher artifact quality than flat invocation with the same Skill set [14]. These findings provide evidence for explicit organization. We compare the utility of four organization schemes while holding the original Skills and resources fixed. These schemes provide no additional organization instructions, a use sequence, stage objectives, or artifact dependencies. Execution trajectories let us trace how agents pass intermediate results between Skills, check them, and revise them under each organization scheme.

2.3 Empirical Studies of Agent Skills

Existing empirical studies examine Agent Skills primarily through content analysis and downstream performance evaluation. Content studies characterize what Skills contain and how developers revise them. Hong et al. analyze the semantic components of `SKILL.md`, synthesize authoring practices through a multivocal literature review, and define violations as Skill smells to study their prevalence and evolution [3]. Wu et al. examine downstream modifications, identifying adaptation patterns and relationships among changes to procedures, resources, and local environments [2]. These studies establish content requirements and recurring adaptation needs, but do not test whether alternative published Skills following different practices improve completion of the same tasks. Our authoring analysis grounds practices in comparisons of published Skill content, the procedures agents actually follow, and the outcomes they achieve on the same tasks.

Downstream evaluations measure whether Skills improve performance and whether agents use their guidance. SkillsBench compares execution without Skills, with curated Skills, and with model-generated Skills across domains using programmatic verification [8]. SWE-Skills-Bench evaluates Skills in software repositories, reporting limited gains and regressions involving conflicts with project context [9]. Skill-Use examines triggering, procedural compliance, and operational boundaries across harnesses [12]. These studies establish outcome differences and obstacles to Skill use. We compare relevant alternatives for the same task under the same configuration, tracing which procedures and resources agents use, skip, or supplement. This distinguishes support described in Skills from support used in task execution.

Some studies also connect content changes to downstream outcomes when improving Skill generation and management. Huang et al. compare experience generation, Skill extraction, and consumption, deriving content evaluation criteria and a meta-skill for

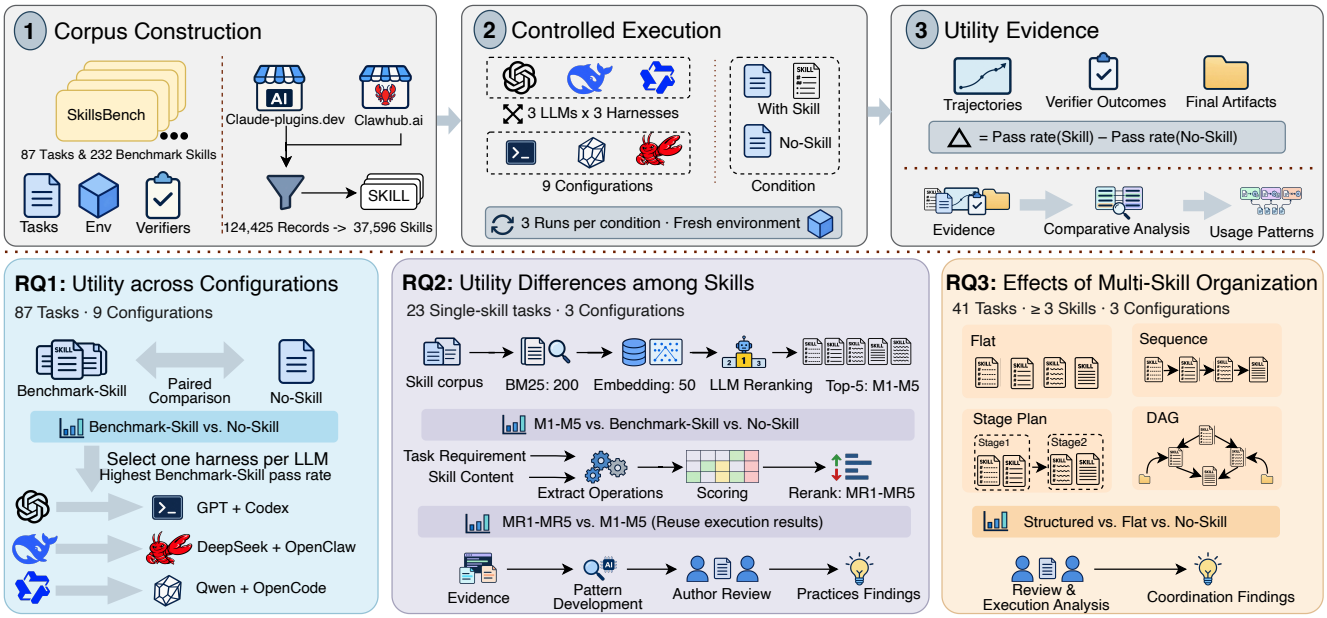

Figure 1: Overview of the empirical study on Agent Skill utility.

Table 1: Skill corpus construction.

Stage	Collection	Completeness	Popularity	Language	Deduplication
Records	124,425	113,972	65,048	40,412	37,596

Table 2: Distribution of tasks by supplied Skill count.

Skills per task	1	2	3	4	5	6	7
Tasks	23	23	20	9	6	5	1

subsequent extraction [10]. ASSAY estimates Skill contributions to restructure libraries and select Skills for individual tasks [43]. SkillReducer compresses and restructures content to reduce token costs [44], while SkillMOO optimizes Skill bundles for pass rate and inference cost and analyzes effective edits [45]. We compare relevant published Skills using their original instructions and resources, and examine configuration and organization effects while holding the supplied Skills fixed. Joint analysis of content, trajectories, and final artifacts relates these utility differences to observed Skill use, providing evidence for authoring, selection, and coordination decisions.

3 Methodology

We combine controlled execution with content and trajectory analysis to link task requirements, Skill content, observed use, and task outcomes, examining how Skill content and organization relate to downstream utility. As Figure 1 shows, RQ1 examines the utility of the same Skills across model–harness configurations and selects configurations for subsequent analyses; RQ2 compares alternative Skills for the same task to derive content design insights; and RQ3 compares organizations of the same Skill set to examine cross-Skill coordination.

3.1 Corpus Construction

Skill Curation. To study third-party Skill reuse, we collect candidate Skills from two public platforms serving different agent ecosystems: https://claude-plugins.dev, a community-maintained platform for Claude Code plugins and Agent Skills [17], and ClawHub (https://clawhub.ai), OpenClaw's public Skill registry [18]. The snapshots we collect on August 20, 2026 contain 47,645 and 76,780 records, respectively, for a total of 124,425 records. Following prior work [3], we filter the corpus for content completeness, adoption, language, and duplication, as summarized in Table 1. First, records must have nonempty names, descriptions, and bodies, since retrieval and subsequent content analysis depend on all three fields. Second, we keep records with at least 100 downloads as evidence of actual use. Third, names and descriptions must be English, and non-Latin letters must account for at most 3% of body letters, which limits language variation. Finally, we identify duplicates by hashing normalized Skill bodies with SHA-256. For normalization, we remove frontmatter and Markdown formatting, lowercase the body, and remove nonalphanumeric characters. We retain one representative per duplicate group, prioritizing download count, then star count, then source identifier. This leaves 37,596 Skills for candidate retrieval.

Task Benchmark. We select SkillsBench as the task benchmark for studying Skill utility. It provides tasks with associated Skills, executable environments, and programmatic verifiers, enabling utility assessment through task execution and outcome verification [8]. We use 87 tasks spanning eight domains, collected as of September 20, 2026 [46]. Table 2 summarizes their distribution by the number of supplied Skills. These tasks include 232 benchmark-supplied

Skills, which we refer to as *Benchmark Skills*. We select task subsets according to each RQ's comparison objective. RQ1 uses all 87 tasks to examine the utility of the same Skills across model–harness configurations. RQ2 uses the 23 tasks supplied with a single Skill to compare how alternative Skills support the same task. RQ3 uses the 41 tasks supplied with at least three Skills to compare how ordering, stage grouping, and dependencies for the same Skill set affect downstream utility.

3.2 Controlled Execution

LLMs and Agent Harnesses. We select a proprietary LLM and two open-weight LLMs with different architectures and parameter scales [10, 11]. GPT-5.5 is a proprietary model for coding and tool-based tasks [47]. DeepSeek-V4-Flash is an open-weight mixture-of-experts (MoE) model with 284B total and 13B activated parameters [48]. Qwen3.5-27B is an open-weight 27B dense model [49]. We select Codex and OpenCode for coding-oriented execution [26, 27], and OpenClaw for general-purpose tool use and automation [28]. Each harness retains its native interfaces and tools, as well as the mechanisms it uses to control task execution.

Task Execution. Skill conditions provide the corresponding Skills, and the task instruction explicitly asks agents to consult and use them. The *No-Skill* condition provides neither Skills nor the corresponding consultation and use request. We run each task under each condition three times per configuration, using a fresh environment for every run.

3.3 Utility Evidence

Utility Measurement. After each execution, the task's original verifier assesses the final state: runs that pass verification are recorded as successful, and all others as unsuccessful. We measure task performance by pass rate and downstream utility by the pass-rate change relative to No-Skill [10]. As in SkillsBench [8], repeated outcomes are first averaged within each task and then averaged equally across tasks. Let $p_{t,c,x}$ denote the fraction of three runs that pass for task t under configuration c and condition x. For task set T, the pass rate is

$$P_c(x;T) = \frac{1}{|T|} \sum_{t \in T} p_{t,c,x}. \tag{1}$$

On the same task set and model–harness configuration, we define the downstream utility of a Skill condition relative to No-Skill as

$$\Delta_c(x;T) = P_c(x;T) - P_c(\text{No-Skill};T). \tag{2}$$

The sign of $\Delta_c(x;T)$ indicates a pass-rate gain or loss relative to No-Skill.

Execution Evidence. Building on outcome evaluation, we examine execution trajectories and final artifacts to compare Skill use in actual operations and relate it to task outcomes across conditions. Subsequent RQ sections describe case selection and analysis.

3.4 RQ1: Skill Utility across Model–Harness Configurations

To examine how the utility of the same Skills varies across model–harness configurations, we cross three LLMs with three agent harnesses and evaluate all nine configurations on the 87 tasks. Each configuration includes Benchmark-Skill and No-Skill conditions; for a given task, the Benchmark-Skill condition provides the same benchmark-supplied Skills and resources across configurations. Execution follows Section 3.2, with evaluation defined in Section 3.3.

For each configuration, we compute overall pass rates under both conditions and the gain from Benchmark-Skill relative to No-Skill. We compare gains across harnesses for the same model and across models under the same harness, and compare model pass-rate gaps under the two conditions within each harness. At the task level, we distinguish four groups based on gains and losses across the nine configurations, comprising gains without losses, losses without gains, both gains and losses, and no changes. For tasks showing both gains and losses, we compare case trajectories across configurations to examine Skill resource use and the handling of task constraints.

Based on the overall RQ1 results, we select the harness with the highest Benchmark-Skill pass rate for each model, yielding three configurations for comparing the utility of Skill content and organization in RQ2 and RQ3. This retains all three models while limiting the experiment scale. We compare candidates and organization schemes within each configuration.

3.5 RQ2: Utility Differences among Skills Targeting the Same Capability

RQ2 uses 23 tasks, each supplied with one Benchmark Skill, so that alternative Skills for the same task can be compared and their differences examined through content and execution evidence. This focuses the comparison on individual Skill support and reduces variation from multi-Skill coordination. Under the three configurations selected in RQ1, we compare candidate utility, examine operation information for selection, and trace content use during execution.

3.5.1 Alternative Skill Selection. Comparing downstream utility across Skills for the same task first requires alternatives relevant to its requirements, which we select from the marketplace corpus in Section 3.1. We use a retrieve-then-rerank framework [11, 37], with the original task instruction as the query. We exclude records sharing the normalized body or source URL of the task's benchmark-supplied Skill, then retrieve 200 candidates with BM25 [50]. We encode the task instruction and each candidate's name and description using `text-embedding-3-large` and retain the 50 candidates with the highest cosine similarity.

Beyond names and descriptions, the `SKILL.md` body provides specific procedures and applicability conditions [38]. We use Claude-opus-4.6 to rank these 50 candidates using the task instruction and their names, descriptions, and bodies [51]. We call the top five Skills *marketplace candidates* and denote them M1–M5 in relevance order. Each rank refers to a candidate for that task, rather than a single Skill shared across all tasks in the benchmark.

Each task has seven conditions comprising Benchmark-Skill, five marketplace candidates, and No-Skill. Under the three configurations selected in RQ1, each candidate condition provides one Skill with its original content and supporting resources. Execution and evaluation follow Sections 3.2 and 3.3, respectively. We compute each condition's overall pass rate and each Skill condition's downstream utility relative to No-Skill. Marketplace candidates

are aggregated across tasks by relevance rank M1–M5 to examine how relevance rankings relate to observed utility. We also compare candidate pass rates within each task and configuration, providing contrasts for atomic-operation reranking and subsequent content and execution analysis.

3.5.2 Atomic-operation Reranking. The candidate utility comparison shows that Skills with higher relevance ranks do not necessarily yield higher downstream utility (Section 4.2). We observe that candidates for the same task differ in operation coverage, inputs and outputs, and applicability conditions. We therefore rerank candidates by examining how their procedures support required task operations and which steps the agent still needs to supply.

We decompose task requirements and Skill procedures into *atomic operations* for item-by-item comparison, defining them as basic operation units explicitly described in the source that act on concrete objects and have identifiable outcomes. We use an LLM to extract actions, objects, inputs or parameters, and constraints from task instructions [13]. For Skills, we extract procedures or interfaces from `SKILL.md` and available supporting files, recording objects, inputs and outputs, applicability conditions, and resources, along with source locations and excerpts [13, 16]. These representations support assessment, while execution uses the original Skill content. We use Claude-opus-4.6 to extract both task and Skill operations.

For each task operation, we assess whether the candidate's procedures meet the object, input, output, and constraint requirements, distinguishing support that requires only task-parameter binding from support that requires additional substantive logic. Multiple Skill operations may jointly meet a requirement, but their connections must have source evidence, and missing steps cannot be supplied to establish full support. We use Claude-opus-4.6 for LLM-based support assessment [52, 53], providing the task's atomic operations, the candidate Skill's operation descriptions, and the corresponding source excerpts. The model applies these criteria to each task operation and returns one of the following support labels in JSON format.

- **Full (1).** The required operation can be performed through task-parameter binding alone, with no identified conflict with the task's explicit constraints.
- **Partial (0.5).** Relevant support is provided without an identified constraint conflict, but additional substantive logic or decisions are required.
- **Absent (0).** No evidence of support for the operation is found.
- **Conflicting (−1).** The provided procedure violates an explicit task constraint, and the Skill provides no alternative compatible with that constraint.

To rank candidates, we assign heuristic scores of 1, 0.5, 0, and −1 to Full, Partial, Absent, and Conflicting judgments, respectively. These values distinguish full support, support requiring additional logic, missing support, and explicit conflicts. We average scores equally across task operations. Let O_t denote the atomic operation set for task t, and let N_{F}, N_{P}, and N_{C} count the Full, Partial, and Conflicting judgments for candidate Skill s. The score is

$$\mathrm{Match}(t, s) = \frac{N_{\mathrm{F}} + 0.5N_{\mathrm{P}} - N_{\mathrm{C}}}{|O_t|}. \tag{3}$$

We rerank the five marketplace candidates by descending score as MR1–MR5 and reuse execution results to aggregate pass rates by rank. Within each configuration, reranking gain is the mean task-level pass-rate difference between MR1 and M1 (Section 3.3).

3.5.3 Utility-Grounded Authoring Practices. Our unit of analysis is a candidate comparison group for the same task and configuration, used to identify authoring requirements associated with downstream utility. We combine pass-rate differences with Skill content, execution trajectories, and final deliverables to examine how content requirements are adopted and the corresponding task outcomes. Drawing on studies of prerequisites and adaptation needs [2], procedural guidance and verification [3], and program refinement through execution feedback and self-verification [32, 54], we organize observations and practices into five stages of Skill use, namely Preparation, Adaptation, Implementation, Verification, and Correction. Our analysis followed three steps comprising case comparison, LLM induction, and author review.

We selected the initial sample from execution outcomes. The 1,449 RQ2 runs form 69 comparison groups, each containing three runs under each of seven conditions. Five groups contrast marketplace candidates that pass all three runs with those that fail all three, and another five contain this contrast between the benchmark-supplied Skill and marketplace candidates. These ten groups form the initial sample, covering nine tasks and 210 trajectories, including each group's No-Skill runs. Using the operation representations in Section 3.5.2, we compared the procedures, resources, and constraints provided by each candidate, checking whether requirements were present, read, or followed, and comparing execution behavior and deliverables.

Following prior work [2], we used Claude-opus-4.6 to identify candidate patterns in these comparisons. The model received task requirements, original Skill content and resources, operation representations from both sides, execution trajectories, final deliverables, and verification results. It examined relationships between content requirements, observed behavior, and outcomes across the five stages. For each candidate pattern, the model returned its name, description, content and execution evidence, and potential counterexamples in JSON format. Source locations, trajectory excerpts, or deliverable evidence linked observations to interpretations [19].

Two authors checked the patterns and source evidence, distinguishing missing content requirements, requirements that are present but not followed, and support supplied by the agent during execution. They also examined alternative explanations for outcome differences. The authors merged similar patterns, clarified category boundaries, resolved disagreements through discussion, and revised or removed unsupported explanations [20]. We added cases from groups with smaller or no outcome differences to examine recurrence and scope and revisited previously analyzed cases. The two authors then derived concrete authoring requirements, specifying the operations or resources concerned, applicability conditions, and supporting evidence, and grouped them into the five-stage framework by their primary stage of use. For each practice, we record the corresponding tasks and configurations, content and execution evidence, and counterexamples. We distinguish insights based on recurring observations from those based on a single case.

3.6 RQ3: Effects of Multi-Skill Organization on Utility

We compare the downstream utility of the same Skill set under different use orders, stage groupings, and artifact dependencies. The experiments use 41 tasks supplied with at least three Benchmark Skills under the three configurations selected in RQ1. All Skill conditions retain the complete Skill set, original content, and resources, with instructions organizing their use.

Based on task planning [55, 56] and dependency-based coordination [14, 57, 58], we compare Sequence, Stage Plan, and Dependency DAG against Flat, which provides no task-specific organization instructions. Instructions may specify subsets or repeated uses of Skills.

- **Flat.** Original Skills are provided with a general instruction to consult and use them, without task-specific organization instructions.
- **Sequence.** The order in which Skills should be used is specified.
- **Stage Plan.** Execution is divided into ordered stages, each identifying the Skills to use and specifying the intermediate objective they should achieve.
- **Dependency DAG.** Each node represents one use of a Skill. Prerequisite dependencies and artifacts passed between nodes are explicitly specified, with subsequent nodes instructed to execute only when the required artifacts are available.

The authors manually constructed the experimental organization instructions using only SkillsBench task descriptions and the corresponding Skills. Another author checked whether use orders, stage objectives, and artifact dependencies were supported by the source materials [19]. The review distinguished explicitly specified relationships from author choices and resolved disagreements through evidence review and discussion.

Execution and evaluation follow Sections 3.2 and 3.3, respectively. We report each condition's overall pass rate and downstream utility relative to No-Skill, measuring organization gains by the mean task-level pass-rate difference from Flat. We also group tasks by the number of supplied Skills to compare pass rates and organization gains across groups. For execution analysis, we followed the case comparison, LLM induction, and author review procedure in Section 3.5.3. We compared organization conditions for the same task and configuration, checking whether use orders were followed, stage objectives were met, and intermediate artifacts were read, used, or revisited by subsequent operations. Cases with pass-rate gains and losses informed design insights for cross-Skill coordination and their scope of applicability.

4 Results

4.1 RQ1: Skill Utility across Model–Harness Configurations

As Table 3 shows, Benchmark Skills improve the overall pass rate in all nine configurations, with gains of 4.60–19.54 percentage points over No-Skill. The same Skills also widen the performance gap between GPT and Qwen. Under the same harness, GPT already has a higher No-Skill pass rate and obtains a Skill gain 3.45–8.43 percentage points larger than Qwen's. Under OpenCode, GPT and Qwen gain 19.54 and 11.11 percentage points, respectively, widening their pass-rate gap from 10.73 to 19.16 percentage points. In the GPT–Qwen comparison, the model with higher No-Skill performance also obtains larger Skill gains under all three harnesses.

Table 3: Pass rates and gains over No-Skill across model–harness configurations. Bold values indicate the highest Benchmark-Skill pass rate and gain within each model.

Configuration	No-Skill (%)	Benchmark-Skill (%)	Gain (pp)
GPT + Codex	42.15	**54.02**	11.88
GPT + OpenClaw	30.65	47.89	17.24
GPT + OpenCode	24.14	43.68	**19.54**
DeepSeek + Codex	34.10	44.44	10.34
DeepSeek + OpenClaw	35.63	**50.19**	**14.56**
DeepSeek + OpenCode	36.40	48.28	11.88
Qwen + Codex	2.30	6.90	4.60
Qwen + OpenClaw	3.45	17.24	**13.79**
Qwen + OpenCode	13.41	**24.52**	11.11

However, the same Skills can yield gains on a task in one configuration and losses in another. Figure 2 shows this pattern in 32 of the 87 tasks (36.78%), and the differences appear both across models under the same harness and across harnesses for the same model. What varies with the configuration is therefore not only the size of a Skill's gain but whether it helps at all.

Execution trajectories further show that Skill-recommended procedures can become an execution burden, with utility depending on whether the configuration can carry them out effectively. For example, `exam-block-sequencing` allows solvers, heuristics, or hybrid methods, while its supplied Skills prioritize integer programming. DeepSeek + Codex passes all three No-Skill runs but times out in all three Benchmark-Skill runs. One successful No-Skill run uses simulated annealing and local search. In contrast, one Benchmark-Skill run omits count coefficients from the integer program's objective function, then repeatedly debugs the search process after correcting the formulation, eventually timing out. GPT passes all three runs with the same Skills under the same harness. In one run, its solver process is also terminated, but it then switches to an allowed heuristic method, retains a feasible solution, and checks the final files before delivery. This contrast suggests that when a configuration struggles to carry out Skill-recommended procedures, the guidance can add execution burden and reduce task performance below No-Skill.

Following the selection rule in Section 3.4, RQ2 and RQ3 use the configuration with the highest Benchmark-Skill pass rate for each model, namely GPT + Codex (54.02%), DeepSeek + OpenClaw (50.19%), and Qwen + OpenCode (24.52%).

Finding 1: Skill utility depends on the fit between supplied procedures and the target model–harness configuration's ability to execute and adapt them. Recommended procedures can become an execution burden, even on tasks the configuration can solve without Skills.

Figure 2: Task-level gains over No-Skill from the same Benchmark Skills across nine configurations. Rows and columns represent configurations and tasks, respectively. Tasks are grouped by gains and losses across configurations, with group sizes in parentheses.

Table 4: Pass rates (%) by original relevance rank.

Configuration	No-Skill	Benchmark-Skill	M1	M2	M3	M4	M5
GPT + Codex	36.23	**49.28**	42.03	40.58	43.48	37.68	37.68
DeepSeek + OpenClaw	28.99	**44.93**	40.58	39.13	37.68	42.03	36.23
Qwen + OpenCode	17.39	18.84	14.49	**20.29**	14.49	15.94	17.39

4.2 RQ2: Utility Differences among Skills Targeting the Same Capability

We first compare the utility of alternative Skills for the same task, then examine whether operation support improves candidate selection, and finally relate Skill content and observed execution to authoring practices associated with utility differences.

4.2.1 Utility Differences among Alternative Skills. Relevant Skills for the same task do not provide equivalent downstream utility. Among the 69 comparison groups formed by 23 tasks and three configurations, 27 (39.13%) have different pass rates across the five marketplace candidates. Table 4 aggregates candidate pass rates by original relevance rank M1–M5 and shows that candidates selected for relevance can still yield losses. Under Qwen + OpenCode, M1, M3, and M4 have lower overall pass rates than No-Skill, only M2 exceeds it, and M5 ties it. Relevance to a task does not guarantee an improvement in task performance.

Higher relevance also need not translate into larger gains. In 16 of the 69 comparison groups (23.19%), the first-choice candidate M1 has a lower pass rate than at least one lower-ranked candidate. When aggregated by rank, the highest-performing marketplace groups are M3 (GPT + Codex), M4 (DeepSeek + OpenClaw), and M2 (Qwen + OpenCode). Overall relevance ranking misses utility differences of this kind, leaving more useful candidates below the first choice.

Table 5: Pass rates (%) and reranking gains over M1.

Configuration	No-Skill	Benchmark-Skill	M1	MR1	Gain (pp)
GPT + Codex	36.23	49.28	42.03	46.38	+4.35
DeepSeek + OpenClaw	28.99	44.93	40.58	44.93	+4.35
Qwen + OpenCode	17.39	18.84	14.49	20.29	+5.80

A candidate can match the task topic while leaving key operations for the agent to supply. The original first choice for `lab-unit-harmonization`, M1 (`medical-unit-converter`), provides medical unit conversion, but its interface requires known source and target units. The task also requires removing incomplete records, normalizing scientific notation and decimal formats, and using physiological ranges to identify values requiring conversion. Using M1 still leaves the agent to implement unit identification and additional cleaning steps, and operational gaps of this kind are not fully reflected in the overall relevance ranking. Candidate selection also needs to distinguish the support provided from the steps the agent must supply.

Finding 2a: Relevance ranking overlooks Skills with higher downstream utility. Highly ranked candidates can match the task topic while leaving operations for the agent to implement.

4.2.2 Utility Gains from Atomic-operation Reranking. Reranking by atomic operation support improves the overall utility of the first-choice Skills in all three configurations. MR1, the first choice after reranking, improves pass rates over the original first choice M1 by 4.35, 4.35, and 5.80 percentage points under GPT + Codex, DeepSeek + OpenClaw, and Qwen + OpenCode, respectively (Table 5). Under Qwen + OpenCode, the first-choice pass rate rises from 14.49% to 20.29%, turning a loss relative to No-Skill into a gain. The candidate set, Skill content, and execution records remain unchanged. This support information complements overall relevance judgments in identifying Skills with higher downstream utility.

Atomic operation matching distinguishes the specific support candidates provide and the prerequisites for using it. The score in Section 3.5.2 accounts for directly usable support, support requiring additional substantive logic, absent support, and constraint conflicts, relating candidate scores to individual task requirements. On `lab-unit-harmonization`, MR1 is the original M2 (`clinical-data-cleaner`). Whereas M1's conversion interface requires known source and target units, M2 provides unit-standardization guidance, a `drop` strategy for removing incomplete rows, operations for detecting outliers using clinical thresholds, and reference-range checks. These provisions address the required cleaning and anomaly-detection steps beyond conversion between known units. Although both candidates concern medical unit processing, they differ in the operations supported and the steps left for the agent to supply.

Execution trajectories further show how this support enters task processing. In one successful GPT + Codex run with M2, the agent reads the Skill's threshold reference file, removes incomplete rows, converts numeric formats, and processes units for each feature. It also checks output formats, missing values, and numeric ranges. This run links the candidate's reference resources to concrete processing steps, showing how operation support is used during execution. Together with the overall reranking gains, these observations suggest that Skill utility depends not only on relevance to the task but also on usable support for its required operations.

Finding 2b: Selecting Skills by their support for required operations improves downstream utility. Within the same candidate sets, atomic-operation reranking raises first-choice pass rates by 4.35–5.80 percentage points over relevance ranking.

4.2.3 Utility-Grounded Authoring Practices. Beyond task relevance and operation matching, downstream utility also relates to how Skill content specifies conditions of use, execution procedures, and required outcomes. We organize the authoring requirements identified through content and execution analysis into the 17 practices in Table 6. The framework groups practices into five stages by their primary use. Each applies to the relevant operations or resources.

Preparation and Adaptation address the conditions under which Skill support can be used for the current task. Preparation specifies applicability boundaries, reference use, and environment and access prerequisites (P1–P4), helping the agent assess applicability and prepare for use. Adaptation connects general guidance to the task by explaining method and parameter choices, input and output semantics, and the parts of examples that require replacement (A1–A3). These requirements explain how relevant capabilities can be used in the current task, addressing execution gaps left by unclear prerequisites or insufficient task adaptation.

Implementation concerns whether Skill content translates its stated capabilities into concrete operations. Beyond naming scripts or describing tools, content needs to provide accessible resources and explicit invocation instructions, including required parameters, input formats, and expected outputs (I1–I2). For procedures involving multiple steps, it should explain how intermediate data or states pass between operations and which outcomes and constraints alternative methods must preserve (I3–I4). These practices address the executability of operation support, guiding resource invocation, step coordination, and method substitution while reducing what the agent must supply to turn capability descriptions into concrete actions.

Verification and Correction address whether execution results meet task requirements and how to proceed when execution is blocked. Verification specifies acceptance criteria, the actual deliverable, and conditions for revalidation after changes (V1–V3), so that checks cover the final delivery state. Correction explains diagnosis and recovery, defines when to retry or switch methods, and preserves justified acceptance requirements during repair (C1–C3). Together, these practices connect operation completion, result checking, and failure handling to the delivery of results that meet task requirements, enabling Skill support to contribute to task gains.

Table 6: Five-stage Skill authoring framework with 17 practices.

PREPARATION 4 practices		
P1	Define Applicability Boundaries	State the supported tasks, objects, and conditions, including cases in which the Skill does not apply.
P2	Link References to Operations	At each operation that needs a reference, specify its location, when to consult it, and how to use it.
P3	Specify Environment Requirements	List required tools, versions, and environment conditions, with readiness checks and setup steps.
P4	Specify Access Requirements	State authentication methods and required permissions, and explain how to confirm resource access.
ADAPTATION 3 practices		
A1	Explain Selection Criteria	Explain which task conditions justify each method or parameter choice.
A2	Define Data Semantics	Define input and output meanings, including units, conventions, mappings, missing-value rules, and their basis.
A3	Mark Replaceable Example Parameters	Mark values to replace and structures to retain when adapting examples or templates.
IMPLEMENTATION 4 practices		
I1	Provide Referenced Resources	Include referenced scripts and resources or provide a valid way to obtain them; verify their locations.
I2	Document Resource Invocation	Provide executable invocation instructions, required arguments, input formats, and expected outputs.
I3	Specify Operation Dependencies	State necessary ordering and explain how intermediate files, data, or states pass to later operations.
I4	Preserve Outcomes Across Alternatives	Define the results and constraints that alternative methods must preserve, including any additional steps required.
VERIFICATION 3 practices		
V1	Define Acceptance Criteria	For each key requirement, identify what to check and the criteria for passing.
V2	Validate Final Deliverables	Identify the actual deliverable and require checks of its contents and final state.
V3	Specify Revalidation Triggers	Specify which changes invalidate earlier checks and which results must be checked again.
CORRECTION 3 practices		
C1	Explain Diagnosis and Recovery	Explain how to diagnose relevant failures and which corrective actions to take.
C2	Define Retry Boundaries	State what must change before retrying and when to stop or switch methods.
C3	Preserve Justified Acceptance Criteria	Preserve justified acceptance requirements during repair; revise them only with evidence that they are incorrect or inapplicable.

Finding 2c: Skill authoring for utility links executable procedures to recovery and checks on the final deliverable. The identified practices emphasize rechecking results after changes and preserving justified acceptance criteria during repair.

4.3 RQ3: Effects of Multi-Skill Organization on Utility

The downstream utility of the same Skills varies with their organization, with schemes specifying stage objectives or artifact dependencies outperforming order alone. As Table 7 shows, Sequence, Stage Plan, and Dependency DAG all achieve higher overall pass rates than Flat in all three configurations, while Stage Plan and DAG also outperform Sequence. DAG improves over Flat by 6.50, 11.38, and 6.50 percentage points under GPT + Codex, DeepSeek + OpenClaw, and Qwen + OpenCode, respectively, achieving the highest or tied-highest overall pass rate. All conditions provide the same Skills and supporting resources. Organizing their use thus yields additional gains, whose magnitude varies across organization schemes.

DAG's additional gains over Stage Plan occur mainly among tasks supplied with five or six Skills. The two schemes have identical overall pass rates in the three-Skill group for GPT and Qwen and differ by only 1.67 percentage points for DeepSeek (Figure 3). In the four-Skill group, their pass rates are identical across all three configurations. In contrast, across the 11 tasks supplied with five or six Skills, DAG improves over Stage Plan by 12.12, 3.03, and 9.09 percentage points in the three configurations, respectively. In this group, Stage Plan improves over Flat by 15.15–18.18 percentage points and DAG by 21.21–27.27 points, both exceeding their gains in the combined three- and four-Skill group under the corresponding configuration. Stage Plan benefits this task group, while the DAG scheme yields further gains.

Execution trajectories show that intermediate artifacts specified in organization instructions serve both to connect operations and to support checks and revisions. For example, `jpg-ocr-stat` requires extracting dates and total amounts from receipts and generating a workbook. Stage Plan assigns OCR and field extraction to one stage and workbook generation to another. DAG instead requires saving the OCR text produced using `image-ocr`, then extracting and saving structured fields for subsequent workbook generation using `xlsx`. GPT + Codex passes 0/3 runs under Flat, Sequence, and Stage Plan, but 3/3 under DAG. In one successful DAG run, the agent inspects the extracted fields, reads the OCR text for the corresponding receipts, then reruns field extraction, updates the field file, and rebuilds the workbook. The OCR text preserves the source of the fields, while the field file connects extraction to workbook generation. The agent can revisit these intermediate results and carry revisions through to the final deliverable. These artifacts support both cross-Skill result transfer and correction before delivery.

Table 7: Pass rates (%) under alternative organizations of the same Skill set.

Configuration	No-Skill	Flat	Sequence	Stage Plan	Dependency DAG
GPT + Codex	44.72	52.03	52.85	55.28	**58.54**
DeepSeek + OpenClaw	34.96	39.84	47.97	**51.22**	**51.22**
Qwen + OpenCode	11.38	21.14	21.95	25.20	**27.64**

Finding 3: Organizing the same Skills around stage objectives or artifact dependencies yields greater utility than specifying use order alone. Intermediate artifacts let agents check results and carry corrections through to the final deliverable.

5 Discussion

5.1 Addressing the Study Challenges

Our study addresses both challenges through comparisons of real-world Skills and analysis of their actual use. For Challenge 1, we compare relevant published alternatives on the same tasks under common configurations, retain their original instructions and resources, and examine the same Skills across configurations and organization strategies. These comparisons ground the 17 authoring practices in observed content and execution differences. For example, the unit-harmonization comparison reveals different support for identifying units and cleaning records despite both candidates matching the task topic. Comparing the support actually available helps identify concrete requirements that a topic-level assessment would miss. For Challenge 2, we jointly examine Skill content, execution traces, and final artifacts to distinguish guidance agents follow, skip, or adapt from operations they supply themselves. LLM-assisted analysis and author review connect these observations to task outcomes, providing evidence for interpreting utility differences without treating the presence of guidance as proof of its use. Reviewing alternative explanations and cases with smaller or no outcome differences also helps clarify the scope of the inferred practices. These comparisons support empirical interpretations, while the effects of individual content requirements need controlled validation.

5.2 Implications for Skill Development and Research

Skill Development and Reuse. RQ1 and RQ2 motivate making operation support, prerequisites, and evaluated configurations explicit when publishing Skills. This information can guide candidate selection, while the authoring framework supports adaptation that preserves task requirements and checks final artifacts. Building on prior work on Skill quality and adaptation [2, 3], the practices can inform review tools and revision suggestions. Representative tasks can also support regression checks when Skill content or execution configurations change. For authors, this means making clear which methods can change, which task requirements remain binding, and which checks must be repeated after a revision. For adopters, candidate descriptions could distinguish directly usable support from operations requiring additional logic. Recording the

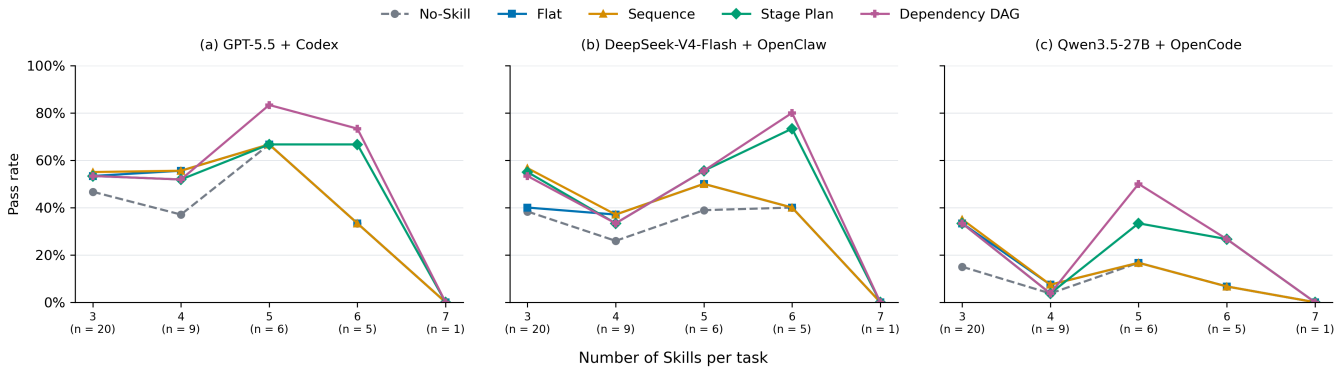

Figure 3: Pass rates by the number of supplied Skills.

tasks and configurations used for validation would further help users judge whether reported gains apply to their setting and decide what to retest before deployment.

Harness Design and Automation. Our findings motivate harness support that connects candidate assessment, execution, and multi-Skill coordination. Tools could identify unsupported operations, check prerequisites, and track which guidance agents use. RQ3 further suggests deriving stage objectives and artifact dependencies, retaining intermediate results, and flagging downstream operations or checks affected by changes. Automating these capabilities would connect planning with recovery and revision, while requiring evaluation of the correctness and cost of automatically generated organization instructions. The organization results suggest that such tools need to represent both the results passed between Skills and the conditions for checking them. When an upstream artifact changes, its consumers may need to rerun and earlier checks may become invalid. A harness could track these relationships and retain source artifacts for inspection. Our manually constructed organization schemes provide a starting point for studying automatic inference of these dependencies and their maintenance during execution.

Evaluation and Generalization. The authoring practices provide concrete variables for controlled experiments on individual content changes and their interactions. Combining pass rates with evidence of guidance use can help explain their effects, following the process perspective of Skill-Use [12]. Further evaluation can test transfer across tasks, configurations, and Skill combinations, including selection and interference in larger libraries [10]. Execution time, token use, and maintenance effort can help assess whether the gains justify adoption costs. Experiments can vary recovery guidance, acceptance criteria, and revalidation triggers separately, then test their combinations to identify interactions. Recording whether agents read and act on the revised instructions can help distinguish limited adoption from ineffective guidance. Independent task sets are also needed to evaluate the stability of operation-support reranking. Together, these evaluations can clarify when richer Skill content or more explicit organization is worth its additional execution and maintenance cost.

6 Threats to Validity

Construct Validity. We use the original task verifiers to compare pass rates under consistent acceptance criteria, and examine execution trajectories and final artifacts to assess how Skill content is used. This measure captures downstream utility in terms of task completion, leaving partial completion and execution cost outside its scope. Atomic operation-based reranking uses common operation representations, support labels, and scoring rules, with utility evaluated through candidates' actual execution outcomes. LLM-based operation extraction and support judgments use source evidence but may still introduce annotation errors that affect candidate rankings. The scores assigned to support labels are heuristic choices. Different label scores or operation granularities could change candidate rankings and the resulting selection gains.

Internal Validity. Each run starts from a fresh environment, and we compare conditions on the same task under the same configuration, which reduces interference from residual state and configuration differences. Skill conditions provide both content and an explicit request to consult and use it, whereas No-Skill provides neither. Gains relative to No-Skill therefore reflect this combined setting. RQ2 retains candidates' original content and resources and compares content with execution evidence to identify authoring features associated with utility. Because candidate attributes may vary together, these insights concern associations observed during reuse. RQ3 retains the same available Skill set, with organization instructions constructed solely from task descriptions and the corresponding Skills and reviewed by another author. Skill-use order, stage objectives, and artifact handoffs still involve author choices, and we therefore interpret gains at the level of complete organization schemes.

External Validity. The study covers 87 tasks across eight domains, three models, and three harnesses, and draws candidate Skills from snapshots of two public platforms to compare utility across tasks and configurations. Corpus filters focus on English-language Skills with complete content and some evidence of adoption. RQ2 and RQ3 use 23 and 41 tasks, respectively, under the selected configuration for each model. RQ2 candidates are restricted to the top five Skills retrieved for each task. Generalization to other languages, newly published or less widely adopted Skills, open-ended workflows, and other configurations requires further study.

Conclusion Validity. We repeat each condition three times, average outcomes within each task, and then average equally across tasks to reduce the influence of individual runs on aggregate comparisons. With only three repetitions per task and condition, task-level pass rates have limited resolution and remain sensitive to execution randomness [59]. RQ2 compares gains before and after reranking using the same candidate set and existing execution records, focusing the comparison on changes in selection order. Because the reranking method was motivated by observations on the current task set, the stability of its gains requires evaluation on independent tasks. RQ3 examines organization gains by supplied Skill count. The five- and six-Skill groups contain six and five tasks, respectively, so group differences may also reflect task composition.

7 Conclusion

We study the downstream utility of Agent Skills on 87 SkillsBench tasks, combining controlled execution with comparisons of Skill content and analysis of execution trajectories. The utility of the same Skills varies across model–harness configurations, and relevance rankings do not fully reflect candidate utility. Reranking by task-operation support improves first-choice pass rates across the three evaluated configurations. With the Skill set unchanged, schemes specifying stage objectives or artifact dependencies achieve higher overall pass rates than specifying order alone. Content and trajectory comparisons provide case evidence for authoring practices concerning deliverable verification, revalidation after changes, and dependency recovery, organized within a five-stage Skill authoring framework. These findings link Skill content, actual use, and task outcomes, informing decisions about Skill authoring, selection, and organization.

Data Availability

All materials supporting this study, including experimental data, execution records, analysis scripts, and paper sources, are available at https://anonymous.4open.science/r/skill_utility-D2C6.

References

[1] Anthropic and the Agent Skills Community. 2025. Agent Skills Specification. https://agentskills.io/specification. Accessed 2026-09-17.

[2] Xinjian Wu, Jingzhi Gong, Gunel Jahangirova, Zhenpeng Chen, and Jie M Zhang. 2026. An Empirical Study of Downstream Adaptation for Agent Skills. *arXiv preprint arXiv:2607.03238* (2026).

[3] David Boram Hong, Aaron Imani, and Iftekhar Ahmed. 2026. From Anatomy to Smells: An Empirical Study of SKILL. md in Agent Skills. *arXiv preprint arXiv:2607.01456* (2026).

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

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

[6] Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, et al. 2026. Skillnet: Create, evaluate, and connect ai skills. *arXiv preprint arXiv:2603.04448* (2026).

[7] Wei-Jung Huang. 2026. Do LLM-Generated Skills Make Better AI Data Scientists? A Component Ablation Across Data-Science Workflows. *arXiv preprint arXiv:2607.07504* (2026).

[8] Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. 2026. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670* (2026).

[9] Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, and Youcheng Sun. 2026. Do Agent Skills Actually Help in Real-World Software Engineering. *arXiv preprint arXiv:2603.15401* (2026).

[10] Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, et al. 2026. From raw experience to skill consumption: A systematic study of model-generated agent skills. *arXiv preprint arXiv:2605.23899* (2026).

[11] Weihang Su, Jianming Long, Qingyao Ai, Qiaozhi He, Yichen Tang, Changyue Wang, Yiteng Tu, Yingbo Wang, and Yiqun Liu. 2026. Skill retrieval augmentation for agentic ai. *arXiv preprint arXiv:2604.24594* (2026).

[12] Jinyi Han, Yuanjian Xu, Ying Liao, Xinyi Wang, Zishang Jiang, Zixiang Di, Fanyang Lu, Zhichao Hu, and Yanghua Xiao. 2026. Skill-Use: Can LLMs Actually Use Skills in Agentic Harnesses? *arXiv preprint arXiv:2608.04828* (2026).

[13] Le Chen, Erhu Feng, Yubin Xia, and Haibo Chen. 2026. Skvm: Revisiting language vm for skills across heterogenous llms and harnesses. *arXiv preprint arXiv:2604.03088* (2026).

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

[15] Tianle Xia, Lingxiang Hu, Yiding Sun, Ming Xu, Lan Xu, Siying Wang, Wei Xu, and Jie Jiang. 2026. Grasp: Graph-structured skill compositions for llm agents. *arXiv preprint arXiv:2604.17870* (2026).

[16] Qiliang Liang, Hansi Wang, Zhong Liang, and Yang Liu. 2026. From skill text to skill structure: The scheduling-structural-logical representation for agent skills. *arXiv preprint arXiv:2604.24026* (2026).

[17] claude-plugins.dev [n. d.]. Claude Code Plugins & Agent Skills—Community Registry. https://claude-plugins.dev/ Accessed 2026-09-26.

[18] OpenClaw. [n. d.]. ClawHub—Public Skill Registry. https://github.com/openclaw/clawhub Accessed 2026-09-26.

[19] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131–164.

[20] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, 275–284.

[21] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, Vol. 36. 68539–68551.

[22] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, Vol. 2024. 9695–9717.

[23] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X

[24] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 50528–50652. doi:10.52202/079017-1601

[25] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2025. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, Vol. 2025. 65882–65919.

[26] OpenAI. 2026. Codex CLI. https://github.com/openai/codex Accessed 2026-09-24.

[27] OpenCode. 2026. OpenCode Documentation. https://opencode.ai/docs/ Accessed 2026-09-24.

[28] OpenClaw. [n. d.]. OpenClaw Documentation. https://docs.openclaw.ai/. Accessed 2026-09-17.

[29] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, Vol. 36. 8634–8652.

[30] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234* (2026).

[31] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19632–19642.

[32] G Wang, Y Xie, Y Jiang, A Mandlekar, C Xiao, Y Zhu, L Fan, and AV Anandkumar. 2023. An open-ended embodied agent with large language models. *arXiv preprint arxiv:2305.16291* (2023).

[33] Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. 2025. Inducing programmatic skills for agentic tasks. *arXiv preprint arXiv:2504.06821*.

[34] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025. Agent Workflow Memory. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=NTAhi2JEEE

[35] Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*. 24497–24524.

[36] Xuan Lu, Haohang Huang, Rui Meng, Yaohui Jin, Wenjun Zeng, and Xiaoyu Shen. 2026. Tools are under-documented: Simple document expansion boosts tool retrieval. In *International Conference on Learning Representations*, Vol. 2026. 141631–141653.

[37] Ryangkyung Kang, Hongcheol Cho, and Youngeun Kim. 2026. SkillRet: A large-scale benchmark for skill retrieval in LLM agents. *arXiv preprint arXiv:2605.05726* (2026).

[38] YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, et al. 2026. Skillrouter: Skill routing for llm agents at scale. *arXiv preprint arXiv:2603.22455* (2026).

[39] Yipeng Ouyang, Yi Xiao, Yuhao Gu, and Xianwei Zhang. 2026. Skcc: Portable and secure skill compilation for cross-framework llm agents. *arXiv preprint arXiv:2605.03353* (2026).

[40] Kun Zeng, Yu Huo, Siyu Zhang, Zi Ye, Yuecheng Zhuo, Haoyue Liu, Yuquan Lu, Junhao Wen, and Xiaoying Tang. 2026. Group of skills: Group-structured skill retrieval for agent skill libraries. *arXiv preprint arXiv:2605.06978* (2026).

[41] Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. 2026. Graph-of-Skills: Dependency-Aware Structural Retrieval for Massive Agent Skills. *arXiv preprint arXiv:2604.05333* (2026).

[42] Tong Bai, Zhenglin Wan, Pengfei Zhou, Xingrui Yu, Yang You, and Ivor W Tsang. 2026. SkillDAG: Self-Evolving Typed Skill Graphs for LLM Skill Selection at Scale.

[43] Yixuan Wang, Yiyang Zhou, Yiming Liang, Congyu Zhang, Fuxiao Liu, Jiawei Zhou, and Huaxiu Yao. 2026. Not All Skills Help: Measuring and Repairing Agent Knowledge. *arXiv preprint arXiv:2606.15390* (2026).

[44] Yudong Gao, Zongjie Li, Yuanyuan Yuan, Zimo Ji, Pingchuan Ma, and Shuai Wang. 2026. Skillreducer: Optimizing llm agent skills for token efficiency. *arXiv preprint arXiv:2603.29919* (2026).

[45] Jingzhi Gong, Ruizhen Gu, Zhiwei Fei, Yazhuo Cao, Lukas Twist, Alina Geiger, Shuo Han, Dominik Sobania, Federica Sarro, and Jie M Zhang. 2026. Skillmoo: Multi-objective optimization of agent skills for software engineering. *arXiv preprint arXiv:2604.09297* (2026).

[46] SkillsBench Contributors. 2026. SkillsBench Task Directory. https://github.com/benchflow-ai/skillsbench/tree/main/tasks. Accessed 2026-09-20.

[47] OpenAI. 2026. GPT-5.5 Model. https://developers.openai.com/api/docs/models/gpt-5.5 Accessed 2026-09-24.

[48] DeepSeek. 2026. DeepSeek-V4-Flash Model Card. https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash Accessed 2026-09-24.

[49] Qwen Team. 2026. Qwen3.5-27B Model Card. https://huggingface.co/Qwen/Qwen3.5-27B Accessed 2026-09-24.

[50] Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and trends® in information retrieval* 4, 1-2 (2009), 1–174.

[51] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 conference on empirical methods in natural language processing*. 14918–14937.

[52] Toufique Ahmed, Premkumar Devanbu, Christoph Treude, and Michael Pradel. 2025. Can LLMs Replace Manual Annotation of Software Engineering Artifacts?. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. 526–538. doi:10.1109/MSR66628.2025.00086

[53] Ruiqi Wang, Jiyu Guo, Cuiyun Gao, Guodong Fan, Chun Yong Chong, and Xin Xia. 2025. Can llms replace human evaluators? an empirical study of llm-as-a-judge in software engineering. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 1955–1977.

[54] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, Vol. 36. 58202–58245.

[55] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*. 2609–2634.

[56] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 11523–11530. doi:10.1109/ICRA48891.2023.10161317

[57] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An LLM Compiler for Parallel Function Calling. In *Forty-first International Conference on Machine Learning*. https://openreview.net/forum?id=uQ2FUoFjnF

[58] Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, et al. 2024. Controlllm: Augment language models with tools by searching on graphs. In *European conference on computer vision*. Springer, 89–105.

[59] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint arXiv:2406.12045* (2024).