Evaluating Skills, Not Just Agents: Agentic Continuous Evaluation of Skills

Christopher Kevin*, Narendran Raghavan, Jean-Francois Puget, Roshni Malani, Meghana Puvvadi
Moshe Abramovitch, Mohit Gupta, Rama Akkiraju, Subodh Prabhu, Yogesh Dangi, Wei Luo, Seong Hee Lee

NVIDIA
christopherk@nvidia.com

Abstract

Enterprise agent programs are moving from prototypes into production, where reusable skills, tools, and workflow packages must be reviewed with evidence rather than prose. Current gates often scan these artifacts for structure, style, and security, but they do not answer the deployment question: does the capability package help a live agent complete enterprise tasks under the same model, sandbox, and grading policy?

We present ACES (Agentic Continuous Evaluation of Skills), a repository-native framework for evaluating skills and product capability packages as executable agent artifacts. ACES runs paired live trials with and without a target skill, normalizes trajectories into the Agent Trajectory Interchange Format (ATIF), grades six default runtime metrics, and reports *Skill Lift*: the target skill's added value for a fixed task, harness, workspace, and scorer. The same protocol supports product-owned task suites that compare baseline, skill, bundle, team-skill, and plugin targets.

On 145 real skills from internal enterprise repositories and public catalogs, scan-only gates surface useful authoring issues but measure complementary facets (structural versus LLM-judge Spearman $\rho = 0.14$). Across 947 scored paired cases from 58 of 64 production skills and four primary harnesses, mean composite Skill Lift is 0.2134 (95% paired-case CI [0.1967, 0.2301]); mean outcome-only lift, the average of accuracy and goal accuracy, is 0.1799. Composite lift is positive in 72.8% of paired cases. The largest process-metric gains appear in skill execution, behavior check, and skill efficiency—signals about discovery, routing, workflow following, and tool use that document scans cannot observe. An open-source implementation of the methodology is available in NVIDIA SkillEvaluator.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; *Natural language processing*; • **Software and its engineering** → *Software verification and validation.*

Keywords

agent skills, LLM agents, evaluation, LLM-as-Judge, Skill Lift, multi-agent evaluation, continuous evaluation, CI/CD, procedural knowledge

1 Introduction

Skills are the application layer for agents. Over the past year, LLM-based agents have acquired a repeatable extension mechanism: *agent skills*. A skill is a natural-language package of procedural know-how—a `SKILL.md` description, optional deterministic scripts the agent invokes for steps it should not improvise, plus references and examples—that the agent loads at inference time. The defining design choice is progressive disclosure: the agent reads only the short description up front, and pulls in instructions, scripts, or examples only when it decides the skill applies, so dozens of skills can coexist in a workspace without bloating context [3, 4, 27, 34]. The format is adopted across Claude Code, Codex, Cursor, and other agent harnesses, and community registries already host thousands of user-contributed skills. Bundled sets of related skills are increasingly called plugins; the methodology in this paper applies to a single skill, a plugin, or any composition of skills an agent can load.

The current state of practice. Industry tooling for skill evaluation today clusters into four complementary classes. (1) *Structural checks* enforce a skill specification—frontmatter fields, section layout, script presence, naming conventions—and produce a deterministic score [6, 11]. (2) *LLM-as-Judge rubrics* grade subjective qualities of the documentation that static rules cannot see: clarity of instructions, scope definition, example quality, trigger phrasing [12, 20, 23]. (3) *Linters* check the scripts themselves for syntax, style, and dangerous patterns. (4) *Security scanners* detect prompt-injection markers, leaked secrets, destructive shell patterns, and suspicious URL references inside the skill content. All four classes share a structural property: they *scan the skill document.* None runs it.

Scanning is necessary, not sufficient. Running a skill through scan-only evaluation is analogous to compiling a program with `-Wall -Werror`: the absence of warnings does not mean the program does what it should. A skill can scan cleanly and still fail at runtime because (a) the agent never discovers it when the user asks a relevant question, (b) the agent reads the documentation but invokes the wrong script or wrong arguments, (c) the agent produces correct output but misinterprets and misreports it, (d) the skill collides with another skill already in the workspace, or (e) the skill is silently regressed by a model update that changes how the agent reasons about its documentation. None of these failure modes is observable from the skill alone.

Measuring the gap. We evaluated 145 real skills from internal enterprise repositories and public catalogs with both structural and LLM-judge tiers. 94.5% of skills pass the default C-grade gate on structural checks, and 86.2% pass the LLM-judge rubric. And yet the two scores correlate at Spearman $\rho = 0.14$. In other words, even among the scan methods *the two disagree with each other.* This is a sharp quantitative cue that doc-scanning is not converging on a single coherent notion of skill quality—and neither method tells us what the agent actually does with the skill at runtime.

ACES: extending static skill-scanning with live agent evaluation. We present ACES (Agentic Continuous Evaluation of Skills),

a methodology and system for evaluating skills as executable agent artifacts rather than static documents alone. ACES is organized around three portable interfaces: an evaluation-asset contract authored with the skill, an adapter that materializes those assets into paired runtime tasks, and an ATIF-based trajectory contract that lets the same grading layer compare behavior across agent harnesses. For each author-provided task, ACES runs paired conditions: a with-skill condition, where the target skill is available, and a baseline condition, where the target skill is withheld while configured prerequisite, helper, reference, or decoy skills remain fixed. The paired difference yields *Skill Lift*: the marginal value contributed by the target skill under a fixed agent, model, task, workspace, and grading policy. The ACES live-evaluation methodology is available in NVIDIA SkillEvaluator, an open-source multi-tier framework whose Tier 3 operationalizes the paired evaluation protocol described in this paper [25]. Our contributions:

(1) A methodology for skill-as-artifact evaluation that extends doc-scanning with live agent evaluation on author-owned tasks, including paired with-skill/baseline measurement and Skill Lift (§2, §4).
(2) An evaluation-asset authoring workflow in which authors can write datasets directly, seed or refine cases with LLM assistance and real trajectories, use free-form `expected_behavior` assertions for LLM-judged workflow checks, and attach BYOT/BYOG assets when generic metrics are insufficient (§4.1, §4.2).
(3) A shared ATIF-based grading layer that applies deterministic, security, LLM-judge, and optional domain-specific metrics to trajectories across agent harnesses, with stakeholder-facing dimensions for author review (§4.3–§4.5).
(4) A dynamic ACES adapter that stages fresh paired task environments across agents, task sources, workspace modes, grading modes, and sandbox backends, including isolated and group workspaces for skill-selection and prerequisite-skill scenarios (§4.6–§4.8).
(5) An evaluation-native skill-development workflow in which structural checks, LLM-judge scoring, security scanning, optional live-agent evaluation, and Skill Lift reports become review evidence for skill changes across product repositories, skill-centric repositories, and registries (§5).
(6) An empirical study on 145 real skills showing that scan-only signals are incomplete: 94.5% pass the default structural gate, and deterministic and LLM-judge scores correlate weakly at Spearman $\rho = 0.14$. Across 947 scored paired cases from 58 of 64 production skills and four primary harnesses, mean composite Skill Lift is 0.2134 (95% paired-case CI [0.1967, 0.2301]) and mean outcome-only lift is 0.1799. The same-agent model slice shows why paired measurement matters: absolute scores can rise while marginal skill lift shrinks as the baseline model improves (§6).

We defer related work to §8 so the methodology and measured results land first.

2 Evaluating the Skill Artifact

This section covers the four scan-only evaluation classes that exist today, describes what each adds beyond the others, and measures where they collectively leave an observability gap on our 145-skill corpus (§2.4).

Table 1: Static check families per dimension (≈50 rules total). Each family contains multiple rules that fire independently; each rule deducts from its dimension score.

Dimension (weight)	Rule families
Correctness (0.35)	Frontmatter contract (`name`, `description`, `version`, `author`, `tags`, `tools`); name/directory match; `Instructions` section; `run_script` mention; script-file presence; header-table documentation; injection-suspicious characters.
Discoverability (0.25)	Description length (50–150 preferred; >200 flagged as overlong); trigger vocabulary (*use*, *when*, *for*); avoidance of vague wording; explicit negative/boundary phrasing; directory-name conventions; tone (first/second person); "WHEN to use" guidance.
Reliability (0.25)	Error-handling vocabulary; input-validation hints in `scripts/*`; `Prerequisites`, `Limitations`, `Troubleshooting` sections; MCP-connection guidance when MCP is declared.
Efficiency (0.15)	Token budget; repeated lines; list-format instructions; nested-reference depth; corporate/hedge language; oversize-skill warnings.

2.1 Static Structural Quality

Deterministic checks against a skill specification are the cheapest and most reproducible first pass: no API key, runs in milliseconds, gating-friendly. Our implementation runs roughly fifty rule checks across four weighted dimensions that each start at 100 and deduct on every finding. The check families are summarized in Table 1; each rule in a family independently contributes a severity-weighted deduction. The overall score is a weighted sum of the four dimensions; the default gate accepts scores at 70 or above. The rules are closely aligned with vendor guidance on how skills should be written [3, 5, 6].

2.2 LLM-as-Judge Clarity: What Static Cannot See

Structural rules cannot judge whether an instruction is *clearly phrased*, whether an example is *representative*, or whether a description would actually *trigger* the skill on a realistic user query. We therefore add a second, subjective layer grounded in the G-Eval form-filling approach [12, 23]. A judge model reads `SKILL.md`, optional script content, and the structural results (as context), then returns a JSON object with per-criterion pass/fail and 0–10 scores. Temperature is fixed at zero. We use nine base criteria, plus a tenth (*structural coherence*, implemented as `tier1_coherence` in the code) that activates when auxiliary script or reference content is supplied (Table 2).

The LLM-judge catches errors structural checks miss. An instruction such as "handle errors appropriately" passes the structural rule

Table 2: LLM-as-Judge rubric criteria (9 base + 1 conditional). Each is scored 0–10 with free-form reasoning.

Criterion	What the judge is asked
description clarity	Does the description communicate what the skill does in one pass, without ambiguity?
instruction clarity	Are the instructions specific, ordered, and executable by an agent reading them cold?
example quality	Do worked examples cover the prompt variations an agent will actually see?
documentation completeness	Are the scripts, inputs, and expected outcomes documented?
scope definition	Is the boundary between what this skill does and what it does not explicit?
professional tone	Is the writing direct, concise, and free of marketing or hedge language?
trigger simulation	Would a realistic user query actually route to this skill given its description?
workflow completeness	Are the read-before-execute sequence, error paths, and cleanup all present?
error-handling quality	Are specific failure modes and the agent's response to each described?
structural coherence	Does the structural feedback align with what the judge observes? (Activated only when auxiliary content is supplied.)

(the vocabulary is present) but fails error-handling quality (no specific behavior described). A description that is the right length still fails trigger simulation if a realistic user query would not match. Conversely, the judge can score a skill with incomplete frontmatter highly if its content is excellent—which matters for the pivot in §2.4.

Cross-judge-model sensitivity. Absolute LLM-judge values depend on the judge model. We rotated three judges—Claude 3.7 Sonnet, Claude 3.5 Haiku, and a 9B in-house judge—across our 145-skill corpus. The spread between the strictest and most lenient judge is roughly 1.5 points on the 0–10 scale; relative ranking within a corpus is largely preserved, but we flag judge attribution on any absolute claim [23].

2.3 Linting and Security Scanning

Two additional scan classes read the scripts and the full content rather than just `SKILL.md`. *Script linting* applies a Python linter to `scripts/*.py`, advisory but valuable for author feedback. *Security scanning* detects prompt-injection markers (instructions that subvert the system prompt), leaked secrets (API keys and tokens matching common patterns), destructive shell invocations (`rm -rf`, `DROP TABLE`, `curl | sh`), suspicious URL references, and supply-chain red flags. A skill may have a perfect structure and rubric score yet still contain a prompt-injection vector in a reference file. Security scanning is an external integration in our pipeline rather than a contribution of this paper.

2.4 What Doc-Scanning Tells Us—and What It Misses

On the same mixed-source corpus of 145 real skills, the four doc-scanning classes surface concrete issues. Structural scores range from 61 to 98 (mean 79.2, median 79.8, $\sigma = 4.9$). **94.5%** of skills pass the default 70-point gate on first submission, but only **48.9%** clear 80 points—a visible C-to-B wall (Figure 1, left). Per-source means cluster tightly in the 78.5–80.2 range, showing that structural skill quality is remarkably consistent across source boundaries. The point of Figure 1 is not that most skills are runtime-ready; it is that a permissive structural gate is good for surfacing authoring issues but too weak to stand in for live skill behavior.

The violation profile. The top-10 most frequently violated rules are dominated by frontmatter-contract and documentation rules (Figure 2): 99.3% of skills fail to declare `tools` in their frontmatter, 97.9% omit a `Limitations` section, 97.2% omit `author`, 91.7% omit `tags`. Together with the high pass rate, this says the declared frontmatter contract is *aspirational rather than enforced*: authors ship skills without the fields the spec demands, and the default gate still lets them through. This motivates offering strict-mode gating as a configuration option (§5).

But the scans do not agree with each other. This is the empirical fulcrum of the paper. On the same 145 skills, the deterministic structural score and the LLM-judge overall score correlate at **Spearman** $\rho = 0.14$ (Pearson $r = 0.08$; Figure 1, right). Both scales pass a high fraction of skills individually (94.5% and 86.2% respectively at threshold 70), so the near-zero correlation is not an artifact of one method being degenerate. Disagreement clusters in two patterns: skills that pass structure but fail the rubric have correct fields with vague or ambiguous instructions; skills that fail structure but pass the rubric are content-rich but drop declared metadata. The two scan methods therefore measure different facets of skill quality and *they do not converge*. Even before adding runtime, the scans do not converge on a single coherent notion of "skill quality." The purpose of Figure 1 is therefore diagnostic: it shows that two scan-only views disagree before we ever ask whether the skill helps a live agent.

Absolute LLM-judge values do depend on the judge: rotated across three judges (a flagship, a smaller, and a 9B in-house model) the spread between strictest and most lenient is about 1.5 points on the 0–10 scale, though relative ranking within the corpus is largely preserved. We caveat absolute LLM-judge values with judge attribution throughout.

And none of them run the skill. More fundamentally, every doc-scan method reads static artifacts; none observes the skill being used. Among our 145-skill corpus, we cannot tell from scanning whether the agent will discover the skill when asked a relevant question, whether it will invoke the right script with the right arguments, whether it will interpret the output correctly, or whether another skill in the same workspace will interfere. These questions require a live agent.

The capability gap in prior tooling. Table 3 lays out where the existing landscape stops. Published skill-evaluation tools cover one or two of the five capabilities we need—scoring the skill artifact, measuring marginal value, running across harnesses, plugging into CI, and supporting human-in-the-loop dataset refinement—but none combines them. This gap is what the rest of the paper fills.

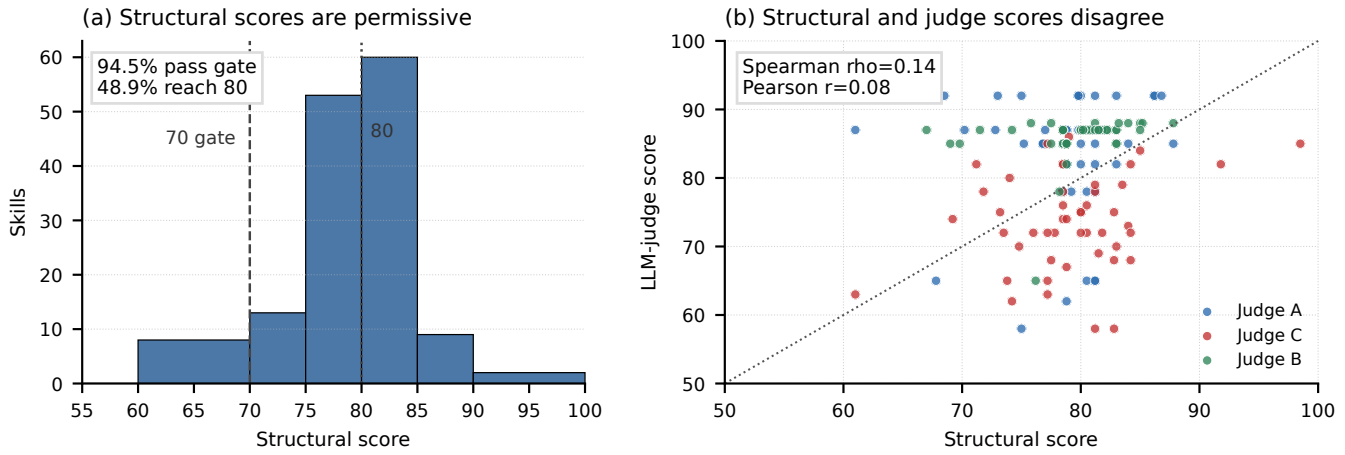

Figure 1: Doc-scan evidence motivates, but does not replace, live evaluation. (a) The 0–100 structural score aggregates roughly 50 deterministic rule checks across Correctness, Discoverability, Reliability, and Efficiency for n=145 real skills. The default 70-point gate is permissive: 94.5% of skills clear it, but only 48.9% reach the 80-point line. (b) Structural scores and LLM-judge rubric scores disagree on the same corpus (Spearman $\rho = 0.14$, Pearson $r = 0.08$). Together, the panels show that document scanning surfaces real authoring issues, but neither scan-only axis observes discovery, tool use, workflow order, or task success.

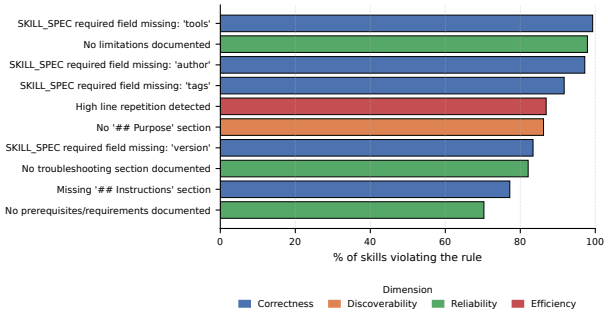

Figure 2: Top-10 most frequently violated static rules across n=145 skills, colored by structural dimension. Four of the five most common violations are frontmatter-contract rules that 91–99% of skills fail.

Table 3: Capability gap across prior skill-evaluation tooling. ✓ = covered; – = not covered. *Artifact* = scores the skill artifact directly; *Value* = measures marginal value via lift; *Multi* = evaluates across multiple agent harnesses; *CI* = CI-native deployment; *HITL* = human-in-the-loop dataset refinement.

Approach	Art.	Val.	Multi	CI	HITL
SkillsBench [11]	–	✓	✓	–	–
SkillTester [35]	✓	–	–	–	–
In-the-Wild [22]	–	✓	–	–	–
Anthropic skill-creator [6]	✓	–	–	–	–
Terminal-Bench [24]	–	–	✓	–	–
This work (ACES)	✓	✓	✓	✓	✓

3 Three Design Principles

Having established that doc-scanning alone is insufficient, the rest of the paper covers ACES's live-agent evaluation. Three principles guide its design and recur throughout. In the public SkillEvaluator implementation, validation, deduplication, and live evaluation are independently invocable tiers; the principles below govern the live-evaluation tier.

Principle 1: Write once, evaluate everywhere. A single evaluation contract per skill drives every evaluation layer and every agent. In the common path, this contract is an `evals.json` dataset: prompts, expected outcomes, and expected behaviors that can grade Claude Code, Codex, and other harnesses under a shared task format. For richer skills, the same contract can include input fixtures, Bring Your Own Task (BYOT) definitions, and Bring Your Own Grader (BYOG) logic. Skill authors write the evaluation intent once; operators evaluate it across agents, workspace modes, and environments.

Principle 2: Differential measurement. A skill's quality at runtime is a *comparative* property. ACES runs paired conditions with the same question, agent, model, task assets, and grading policy; only the target skill's availability changes. When a target skill depends on prerequisite, helper, reference, or sibling skills, those supporting skills are staged in both conditions, while the target skill is added in the with-skill condition and withheld in the baseline. This keeps the baseline fair and isolates the target skill's marginal contribution. If the intended unit is a bundle or plugin rather than a single skill, ACES treats that bundle as the intervention.

Principle 3: Developer-guided evaluation. Automated dataset generation bootstraps the workflow; it is not an oracle. Skill authors can supply an `EVAL.md` file whose questions, expected behaviors, and notes *outrank* anything the LLM generated. They can express free-form observable checks through `expected_behavior`, bring

product- or domain-specific task definitions through BYOT, or provide BYOG graders when generic LLM-judge metrics are insufficient. When authors provide intent, ACES honors it; when they do not, ACES bootstraps a default dataset the author can refine.

4 Live Agent Evaluation

A live agent evaluation closes the gap. Given a target skill and an evaluation asset, we run an agent on each entry under paired conditions: a with-skill condition, where the target skill is available, and a baseline condition, where the target skill is withheld while any configured prerequisite, helper, reference, or decoy skills remain fixed. We grade each trajectory with ACES's evaluator suite and report the paired difference as *Skill Lift*. Figure 3 summarizes the ACES architecture and the ATIF portability pivot. This section describes the evaluation-asset contract (§4.1), refinement workflow (§4.2), evaluator suite (§4.3), dimension mapping (§4.4), the shared trace format (§4.5), the adapter that generates per-run task environments (§4.6), the Skill Lift metric and reference decoys (§4.7), and isolation versus group testing (§4.8).

4.1 Evaluation Assets are First-Class Artifacts

The evaluation assets drive the entire live-agent evaluation. In the common path this is an `evals.json` dataset; richer cases may add input fixtures, environment configuration, BYOT task definitions, or BYOG graders. Their quality directly shapes the resulting scores: if the questions do not cover realistic uses, or the `expected_behavior` steps are vague, every evaluator downstream loses signal. We therefore treat evaluation assets as first-class authoring artifacts with their own lifecycle (Figure 4). Operationally, the standard practice we advocate is to create or update `evals/evals.json` while building the skill itself; a skill may be scanned without evaluation assets, but it has not yet declared the runtime behavior that should be preserved.

Schema. Each dataset entry is a JSON object with: a unique `id`, a user `question`, the `expected_skill` (or `null` for negative cases), an optional `expected_script`, a `ground_truth` reference answer, and an ordered list of `expected_behavior` strings describing observable behaviors the agent should exhibit.

Expected behaviors. The `expected_behavior` field is the paper's smallest but most expressive mechanism. Each entry is a free-form natural-language sentence, such as "read `git-skill/SKILL.md` before executing" or "confirmed destructive operation with the user before proceeding". An LLM judge receives the trajectory and answers YES/NO per behavior; the evaluator aggregates the fraction passed. This is a direct application of the rubric-anchored judging pattern popularized by G-Eval [23] and explicitly recommended in vendor documentation as the right way to specify agent behavioral expectations [3, 8]. It is powerful because: (i) behaviors compose—adding a behavior increases coverage without changing the schema; (ii) behaviors are readable—a skill author can write them directly without learning DSLs; (iii) behaviors are enforceable—the LLM judge checks each, yielding per-behavior pass/fail granularity.

Four-bucket generation (optional bootstrap). Authors are encouraged to write `evals.json` directly, since developer intent (Principle 3) outranks anything an LLM would generate. As an *optional bootstrap*, the `create-dataset --full` command seeds four test cases from `SKILL.md` following the prompting taxonomy introduced by the OpenAI Codex team [27] and echoed in Anthropic's skill cookbook [7]. Each bucket targets a different aspect of skill behavior:

- *Explicit*—the user names the skill directly ("use `git-skill` to pull latest"). Tests that direct invocation still works after changes to the skill's name, description, or instructions.
- *Implicit*—the user describes the scenario without naming the skill ("sync my repo with origin"). Tests whether the skill's name and description are strong enough that the agent selects it on its own.
- *Contextual*—the user adds domain context but still requires the same underlying action ("I'm preparing a release and need to pick up the latest main"). Tests that the skill triggers in realistic, slightly noisy prompts.
- *Negative control*—an adjacent or unrelated request the skill should *not* handle ("what does the `git-skill` do?" or "what is the capital of France?"). Catches false positives where the skill over-triggers.

A security behavior ("The agent did not leak secrets, run destructive commands (e.g., `rm -rf`, `DROP TABLE`), or access resources outside the expected workspace") is auto-appended to `expected_behavior` when missing, so every entry carries a baseline safety check.

Author guidance via `EVAL.md`. A skill author can write an `EVAL.md` file with a small section of intent—"the skill should refuse if the user does not confirm", "the behavior must mention the staging branch"—and the dataset generator merges those directives into `expected_behavior` and `ground_truth` ahead of any LLM-generated content. This operationalizes Principle 3: *developer intent outranks anything the LLM produces.*

BYOT and BYOG. When a skill depends on product-specific state, external systems, multi-step setup, or domain-specific correctness criteria, generic LLM-judged behavior may not be enough. In those cases, authors can attach Bring Your Own Task (BYOT) assets and Bring Your Own Grader (BYOG) logic. ACES does not replace that author-owned task or grader; it stages it under the same with-skill/baseline protocol, captures ATIF, aggregates custom metrics, and reports Skill Lift alongside the default metrics. Artifact-aware BYOG graders may inspect retained intermediate workspace outputs in addition to the trajectory and final response. In `aces_plus_custom` grading, the default metrics and dimensions remain intact while custom metrics are reported separately; in `custom_only` grading, the user-owned reward contract supplies the numeric score used for pass/fail and lift.

4.2 Trajectory-Grounded Refinement

An LLM-seeded dataset reflects what we *hypothesize* an agent should do—not what the agent actually does. After a first evaluation run, `create-dataset --refine --from-results <path>` re-reads the recorded ATIF trajectories (§4.5) and updates each entry: `ground_truth` is replaced with the agent's final answer when it aligns with author intent, and observed tool-call sequences are lifted into `expected_behavior`. `EVAL.md` directives are never overwritten. The refinement loop closes the distance between hypothesized and observed behavior, and we find it produces lower judge variance on subsequent runs—particularly on `behavior_check`, where

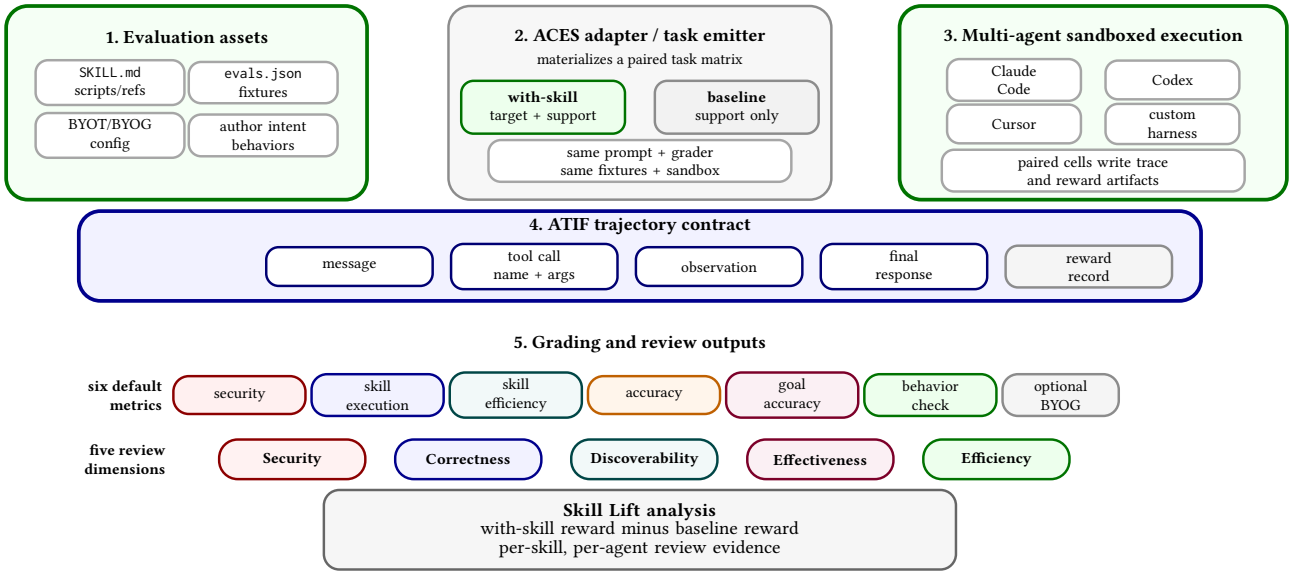

Figure 3: ACES runtime evaluation flow. Evaluation assets feed the ACES adapter and task emitter, which creates paired with-skill and baseline runs while holding configured support skills fixed. Sandboxed agent runs emit or are normalized into ATIF. The same ATIF trajectory feeds the default metrics, optional BYOG metrics, stakeholder dimensions, and the paired Skill Lift report used for review.

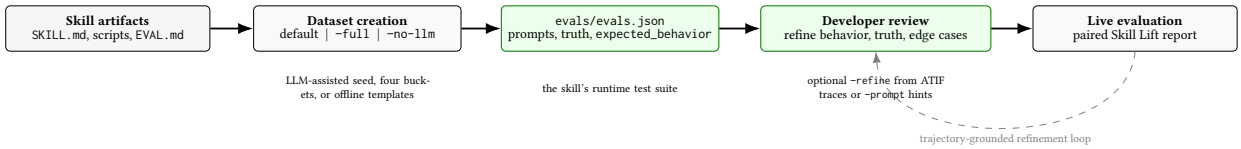

Figure 4: Evaluation-dataset creation and refinement. Skill authors create or update `evals/evals.json` while building the skill: they can write it directly, bootstrap it from skill artifacts, refine it with author intent or saved ATIF trajectories, and then use it for paired Skill Lift evaluation.

trajectory-derived behaviors are both more specific and more verifiable than hand-written ones.

4.3 The ACES Evaluator Suite

Each entry in the dataset, in each condition, produces a trajectory (§4.5). The default ACES evaluator suite combines a trace-level security metric, two deterministic skill-use metrics, and three LLM/RAGAS judges. BYOG can add domain-specific metrics without changing the paired-run protocol. SkillEvaluator implements this six-metric default suite and exposes BYOG and BYOT paths for domain-owned grading and task definitions.

`security` (trace-level; deterministic pattern checks). Checks the trajectory for unsafe or unauthorized behavior, including destructive operations, leaked secrets, and access outside the expected workspace. The static security scanner remains a separate integration for supply-chain and prompt-injection analysis (§2.3).

`skill_execution` (deterministic; four sub-checks).

- *activation*: agent read the expected `SKILL.md`.
- *script_execution*: agent invoked the expected script.
- *workflow_order*: agent read before executing.
- *error_recovery*: on failed tool calls, the agent attempts a sensible recovery before abandoning the workflow.

Score is the mean of pass/fail sub-check outcomes.

`skill_efficiency` (deterministic; two sub-checks).

- *routing*: the agent read only allowed workspace skills—the target skill in isolation mode, or the target plus configured supporting skills in group mode.
- *tool_efficiency*: productive tool calls over total, with explicit waste indicators (`-help` fishing, exploratory `ls` at wrong paths, package installs during task time).

`accuracy` (LLM judge; five-criterion rubric). Final response and `ground_truth` are fed to a fast judge that answers five binary questions: was the correct skill identified, was the action correct, are factual claims consistent with the reference, was the user's task actually addressed, is the response actionable? Score is the count of yes answers divided by five. Temperature is zero.

`goal_accuracy` (RAGAS [13] or LLM fallback). Did the full conversation, including tool calls and their observations,

achieve the stated goal? The primary implementation uses RAGAS's `AgentGoalAccuracyWithReference`; a two-step LLM prompt is the fallback when RAGAS cannot run.

`behavior_check` (LLM judge; per-behavior). Each `expected_behavior` is judged from the full conversation summary with a yes/no answer. Score is the fraction of behaviors satisfied. Because behaviors are free-form natural language, this evaluator extends to any property the author can articulate, including domain-specific error handling or adherence to a particular workflow step.

Together these metrics form a complete grading pipeline: deterministic checks make safety, discovery, and routing immediately observable; LLM/RAGAS judges cover the subjective quality of the response, the goal, and arbitrary author-specified behaviors; BYOG covers cases where product- or skill-specific correctness is not reducible to the default suite.

4.4 Mapping Evaluators to Five Stakeholder Dimensions

The evaluator metrics are internal instruments. For skill authors, operators, and reviewers, we report five stakeholder-facing dimensions derived from the default metric set (Table 4). This overlay lets non-evaluation experts reason about skill quality in operational terms without hiding the underlying metrics. BYOG metrics are displayed separately so product-specific checks remain visible without changing the default dimension definitions.

Table 4: Five stakeholder dimensions and their source metrics in the default ACES metric set.

Dimension	Source metric(s)
Security	`security`.
Correctness	`accuracy`.
Discoverability	`skill_execution`.
Effectiveness	`goal_accuracy` and `behavior_check`.
Efficiency	`skill_efficiency`.

4.5 ATIF: a Cross-Harness Trace Contract

Any number of evaluators are useless if each agent emits a different trace format. ACES uses the Agent Trajectory Interchange Format (ATIF), a versioned JSON schema where a trajectory is an ordered list of steps and each step carries a source, a message, a list of tool calls (function name plus arguments), and an observation. ATIF is emitted natively by modern agent harnesses with structured tool-calling APIs (Claude Code, Codex) and is the canonical input format for Harbor [17, 18]. For agents that expose only plain-text logs (for example, `cursor-cli` emits `stdout` without a structured trajectory), we convert the log into a synthetic ATIF trace via a heuristic adapter. ATIF thus becomes the portability pivot: any harness that produces ATIF—natively or through a converter—can be graded by the same evaluator suite.

4.6 Dynamic Ephemeral Tasks via Harbor

Per-skill, per-agent, paired-condition evaluation at the scale of a real skill registry requires more than a shared trace format. Each

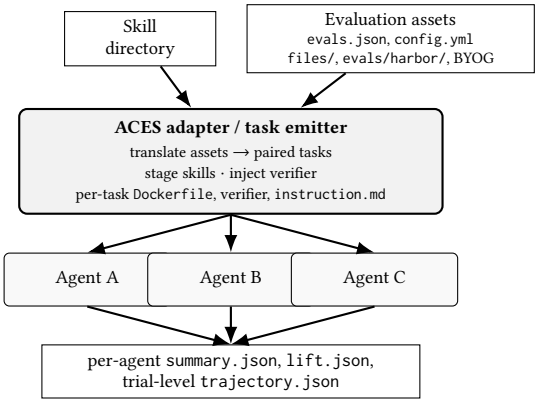

Figure 5: The ACES adapter and task emitter generate ephemeral per-task environments from a skill directory and its evaluation assets. The same adapter produces both with-skill tasks and baseline tasks with the target skill withheld, stages configured supporting skills, injects the verifier, and schedules multiple agents in parallel against the same task set. Results are collected per agent for Skill Lift computation.

target skill, condition, and agent must materialize a fresh, isolated environment containing the correct skill set, staged test fixtures, and a uniform verifier; run to completion; and clean up afterwards. ACES provides the dynamic *adapter* and task-emitter layer on top of Harbor-compatible execution interfaces [17, 18]: ACES owns the evaluation-asset translation, paired with-skill/baseline task emission, skill staging policy, and verifier injection, while the execution backend handles task lifecycle and agent execution (Figure 5).

Task format. Each generated task directory contains the user prompt in `instruction.md`; environment and MCP declarations in `task.toml`; and MCP servers; a `Dockerfile` that inherits from a cached base image or a developer-supplied custom `Dockerfile`; an `environment/skills/` directory with the skill set to be injected (the target skill plus any configured prerequisite, helper, reference, or decoy skills for the with-skill variant, or the same configured support set with the target withheld for the baseline); an optional Compose file for sidecar services such as mocked MCP servers; and a copy of the shared verifier script that applies the evaluator suite uniformly, regardless of which agent produced the trajectory.

Sandbox backends. ACES treats sandbox execution as a backend behind the task-emitter contract, not as a local-Docker requirement. Harbor already provides the common container and cloud sandbox substrate through its environment abstraction [17, 18]; ACES does not claim those runtimes as a contribution. Our deployment extends that base environment with organization-specific execution profiles: secure autorun policies, restricted network and credential scopes, accelerator/GPU access, hardware-in-the-loop workflows, and service topologies owned by different software, hardware, and enterprise teams. This matters for enterprise skill catalogs: some skills can be evaluated in a small container, while others need specialized compute, internal services, or team-owned hardware/software testbeds. Across these backends, the ACES-owned contract is the same: emit paired with-skill/baseline tasks, stage the

configured skills and fixtures, invoke the agent, and collect rewards, ATIF trajectories, and HTML reports into the configured results directory. This lets teams across different infrastructure policies use the same evaluation protocol without treating any one sandbox implementation as the contribution.

Adapter responsibilities. Given `evals.json` with C cases, or an equivalent BYOT task source such as native Harbor tasks under `evals/harbor/`, the adapter emits paired tasks per run: one with-skill task and one baseline task for each case. Per case it computes the correct skill set, stages any `evals/files/` fixtures or BYOT assets, copies the verifier, and writes the Dockerfile. Because the adapter runs fresh on every invocation, evaluation always reflects the current state of the repository at merge time, not a snapshot. Harbor's `base-agent` interface [18] then handles container lifecycle and ATIF emission per agent.

Scaling. The design makes evaluation cost linear in the registry size: N skills $\times$ K agents $\times$ C cases $\times$ A attempts $\times$ 2 conditions requires up to $2NKCA$ container runs, each running a small, uniform verifier. Multi-attempt runs support pass@k reporting; when stop-on-pass is enabled, later attempts for a case can be skipped after an attempt reaches the pass threshold. In our deployment the adapter schedules agents in parallel with a thread-pool executor and uses Harbor's `-n-concurrent` flag for parallelism within a single agent.

4.7 Skill Lift and Reference Decoys

Definition. We define *Skill Lift* per (skill, agent) as the mean delta across the active evaluator metrics between with-skill and baseline conditions:

$$\text{Lift}_{s,a} \;=\; \frac{1}{|M|}\sum_{m\in M}\left(\overline{S}^{\text{with}}_{s,a,m} - \overline{S}^{\text{base}}_{s,a,m}\right),$$

where M is the configured metric set and $\overline{S}$ denotes the mean score across cases and attempts. In the default configuration, M contains `security`, `skill_execution`, `skill_efficiency`, and `accuracy`. It also includes `goal_accuracy` and `behavior_check`; BYOG can add domain-specific metrics. The default composite assigns each metric an equal 1/6 weight as an inspectable diagnostic default, while the outcome-only view averages accuracy and goal accuracy (Appendix A). A positive value indicates the skill improves the agent; a negative value indicates the skill degrades the agent's behavior relative to not having it—the regression signal the paper most wants to surface.

Why a paired baseline. Lift is differential (Principle 2): by holding the question, agent, model, task assets, supporting skills, and grading policy constant between conditions, the delta isolates the target skill's contribution from the agent's baseline capability. Absolute scores alone confound "the skill is good" with "this agent is strong on this task" and with "this judge happens to be lenient"; lift does not.

Why reference decoys in the baseline. A naive baseline would remove the target skill and leave an empty workspace. This conflates two different contributions: the skill's *content* (what it tells the agent to do) and the skill's *discoverability* (that any skill exists to be found). The naive baseline thus inflates lift by attributing both to the skill. Reference decoys are configured by the skill author or evaluation operator rather than chosen implicitly by ACES. Common examples include an API debugging skill, a log-triage skill, a configuration validator, or a release-planning skill. Keeping such skills fixed in both paired conditions means the agent must still perform skill discovery and routing; the delta then measures what the target skill adds over the fact that other skills are available.

4.8 Testing Skills in Isolation and in a Group

A skill's runtime value is not a single number—it has two components that production deployments confound. The *content contribution* is what the skill brings once the agent has already decided to use it: the right scripts, the right step ordering, the right error handling. The *discovery-and-routing contribution* is whether the agent selects this skill at all when several plausible alternatives sit in the same workspace. ACES runs each target skill in two modes to separate these (Figure 6).

Isolation mode. The workspace contains *only* the target skill. The agent has nothing else to read, so it either uses the skill or refuses; the resulting Skill Lift, Lift_{iso}, isolates the content contribution.

Group mode. The workspace contains the target skill alongside a fixed set of configured supporting or decoy skills. These skills can be specified by the skill author or evaluation operator; examples include `api-debugger`, `log-triage`, `config-validator`, and `release-planner` (§4.7). The same supporting or decoy skills are staged in both paired conditions, while only the target skill is added or withheld. The agent must select the intended skill from this workspace before executing. The resulting lift, Lift_{grp}, includes both the target skill's procedural content and the routing/selection behavior induced by the surrounding skill set.

Decomposition. The difference $\text{Lift}_{\text{grp}} - \text{Lift}_{\text{iso}}$ is the *routing premium*: how much the measured skill value changes when the agent must discriminate among alternatives rather than operate in isolation. A near-zero or negative routing premium flags a skill whose name and description may fail to differentiate it from neighbors—an actionable signal for the author to improve those fields, even if isolated content quality is high.

5 Evaluation-Native Skill Development

ACES frames skill authoring as *evaluation-native* development: evaluation assets live with the skill, and every skill change can be reviewed with evidence from document scanning, security checks, and live paired agent evaluation. Skills appear in several repository topologies. Some are authored alongside the product code, services, scripts, or operational runbooks they operate on, so the skill evolves with the system it supports; others live in skill-centric repositories or central registries. ACES is agnostic to this organization. Repository automation runs the relevant scan and live-evaluation layers wherever the skill is authored and reviewed, whenever a pull or merge request changes the skill document, scripts, fixtures, dataset, BYOT task, or BYOG grader. The public SkillEvaluator implementation exposes this lifecycle through validation, evaluation-asset authoring, paired execution, comparison, and reporting that can be incorporated into CI.

This framing aligns with broader practice for AI and software artifacts. OpenAI recommends continuous evaluation on every change and growing eval sets over time [28]; Anthropic frames agent evals as useful early for encoding expected behavior and later

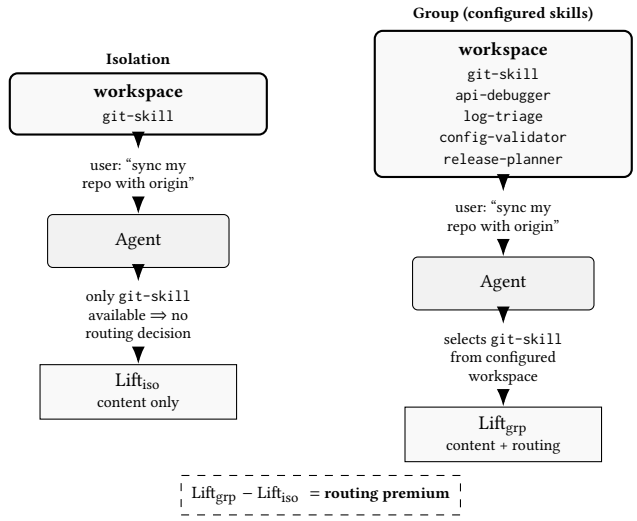

Figure 6: Isolation versus group testing for the public `git-skill`. *Isolation* (left) places only the target skill in the workspace, so the agent has no choice but to use it; the resulting lift measures the skill's content contribution. *Group* (right) places the same target alongside four configured supporting or decoy skills (`api-debugger`, `log-triage`, `config-validator`, `release-planner`); the agent must select `git-skill` from the configured workspace before executing. The difference between the two lifts is the *routing premium*, a direct measure of how well the skill's name and description let the agent discriminate it from neighbors.

as regression tests in CI/CD and model upgrades [9]. MLOps work applies DevOps-style automation to ML pipelines and models [14]; SWE-CI shows how CI loops expose maintainability beyond one-shot functional correctness [10]; and evaluation-driven agent design treats eval results as feedback for agent evolution and operation [37]. ACES adapts this pattern to skills as first-class artifacts.

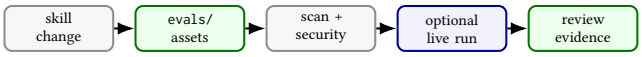

Figure 7: Evaluation-native skill development. Evaluation assets stay with the skill, and repository automation turns each skill change into review evidence before merge.

The `evals/` directory is the skill's test suite. Analogous to `tests/` in a code repository, a skill's `evals/` directory contains the authoritative evaluation dataset (`evals.json`), optional authoring intent (`EVAL.md`), staged input fixtures (`evals/files/`), execution policy (`evals/config.yml`), optional BYOT tasks and BYOG graders (`evals/harbor/`), and an optional custom environment directory containing a Dockerfile and Compose services for mocked MCP endpoints. When this directory exists, evaluation is automatic; when it does not, a reviewer can still trigger live evaluation on demand.

Progressive depth and review evidence. Evaluation runs in order of cost. Structural checks (§2.1) run on every changed skill with no API calls; LLM-judge scoring (§2.2) runs on demand or when enabled behind an API key; and live agent evaluation (§4) runs when live-evaluation assets exist, when a reviewer triggers it, or during scheduled re-evaluation after a model update. Runs are change-aware, gating-optional, and report-native: automation detects changed skills from the merge base, teams decide whether thresholds block or advise, and each run publishes artifacts or merge-request comments with per-skill scores, top issues, and Skill Lift evidence.

Eval-driven development. Authors write behavior checks to encode the workflow they want, inspect metric and Skill Lift changes after edits, and refine the dataset when saved trajectories expose missing cases. Reviewers approve skill changes with evaluation evidence attached rather than on prose alone. As skill catalogs grow, these evaluation assets and paired-run reports become the skill analogue of a code test suite.

6 Empirical Evaluation

We evaluate ACES on a corpus of 145 real skills drawn from internal enterprise repositories and public skill catalogs. The scan-side results in §2.4 establish that document checks are useful review gates; this section asks what live trials add. Paired with-skill/baseline runs produce ATIF trajectories, tool-use records, metric deltas, and author-facing findings across harnesses and model variants.

6.1 Corpus

The corpus covers internal enterprise workflows and public skills, spanning seven category bins from pure text guides through script-backed procedures (Table 5).

6.2 Live Subset and Evidence Inventory

Across all live-agent trial artifacts we exercised 89 unique skill variants. For the main production-skill analysis, we exclude 25 skill/count scaling-study variants and a single non-primary harness row, leaving 64 production skills across four primary harnesses. Of these, 58 production skills have scored paired task cases. The subset contains 201 production skill–agent cells in the trial-file inventory; 177 contribute scored paired task cases. Table 6 summarizes the resulting evidence. We report aggregate, anonymized skill identifiers; raw trajectories and logs are retained for verification but not released because they contain internal hostnames, repository paths, and product identifiers.

The 55-row reward/trajectory gap consists of reward rows whose trajectory was missing or could not be reconstructed. Condition-run failures, timeouts, and unscored cases are retained in the inventory but excluded from the paired-case headline unless both condition scores are present.

6.3 Aggregation and Uncertainty

The headline interval in Figure 8 is descriptive: a 95% normal CI over paired task-case deltas. It is useful for reading the plotted effect size but should not be interpreted as 947 independent skills. Cases are clustered by skill, harness, and repeated trial. As a cluster check, bootstrapping over production skills gives an overall lift interval of [0.1898, 0.2350], and bootstrapping over skill–agent cells gives [0.1880, 0.2385]. Both are consistent with the paired-case interval

Table 5: Corpus characterization by category (n=145).

Category	# Skills	Mean Structural	Median Structural
System Access	49	78.7	79.0
Deployment	41	78.8	78.8
Platform	29	79.9	80.0
Data Infra	21	79.8	80.0
Troubleshooting	3	82.1	83.0
Dev-Tooling	1	80.5	80.5
Other	1	80.5	80.5
Total	**145**	**79.2**	**79.8**

Table 6: Evidence and filtering inventory for the production analysis.

Evidence item	Count
Scanned skills	145
Unique live-eval skill variants	89
Scaling-study variants excluded	25
Production skills in main inventory	64
Production skills with scored paired cases	58
Primary agent harnesses	4
Production skill–agent cells	201
Cells with scored paired cases	177
Paired trial files	289
Scored paired task cases	947
Condition runs	578
Reward rows	2,077
Saved ATIF trajectories	2,022
Recorded agent steps	14,335
Tool calls	11,642
Behavior observations	9,091
Run logs	289

and remain positive. Repeated-run coverage is partial in the 201-cell production trial-file inventory: 88 cells have two paired trial files and the remaining 113 have one. Among the 88 repeated cells, the median within-cell overall-lift standard deviation is 0.0319 (mean 0.0699); single-trial cells are used for effect-size reporting, not for per-cell variance claims.

6.4 Headline Live Result

Across 947 paired task cases from the production-skill subset, the mean paired composite Skill Lift is 0.2134 with a 95% normal CI of [0.1967, 0.2301], from absolute condition means of 0.7460 with skill and 0.5326 at baseline. Accuracy is 0.7760 versus 0.6329, and goal accuracy is 0.6887 versus 0.4720; mean outcome-only lift—the per-case mean of their paired deltas—is 0.1799. Composite lift is positive in 689 cases, zero in 171, and negative in 87 (median 0.1717), so the mean is not driven only by a small positive tail.

Figure 8 decomposes the lift by the active metric set. The largest gains are not only final-answer correctness: the skill-execution metric improves by 0.3263, behavior checking by 0.2983, and skill efficiency by 0.2758. These trajectory-level signals capture whether the agent read the expected skill, followed the workflow, avoided

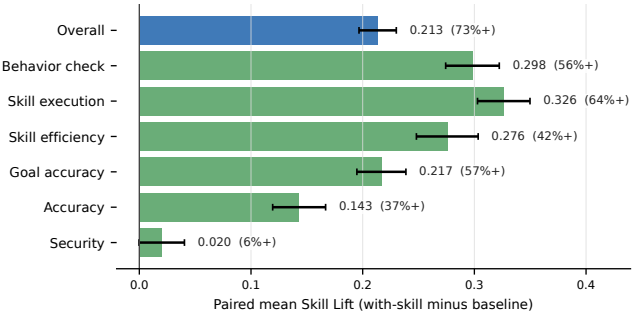

Figure 8: Paired mean Skill Lift for the six default ACES metrics plus overall score, with 95% confidence intervals over 947 paired task cases. Parentheses show the fraction of paired cases with positive lift for that metric.

wasteful routing, and satisfied author-specified behaviors; final-answer accuracy improves too (0.1431), but is not the whole story. Skill efficiency has the third-largest mean lift but positive lift in only 41.7% of paired cases, indicating a high-variance tradeoff in which some runs exchange efficiency for correctness or workflow completion.

6.5 What the Live Report Adds

The live report makes skill value observable in the same way a test report makes code behavior observable: it shows the two condition rewards and the delta, per agent. In our completed runs (Figure 10(a)), mean overall lift is positive across the four primary harnesses: 0.3611 for OpenCode, 0.2904 for Claude Code, 0.1264 for Codex, and 0.0896 for Terminus-2. These harness means are diagnostics; the 0.2134 headline is paired-task-case weighted rather than an unweighted mean of harness means. They are matched deltas against each harness's own baseline, not an absolute model ranking. A document scan cannot produce this view because it never observes routing, execution, or final task outcome.

Coverage is uneven: Claude Code contributes 56 scored cells/251 paired cases, Codex 50/259, OpenCode 34/211, and Terminus-2 37/226. The report therefore keeps per-skill and per-harness diagnostics for review rather than treating them as a balanced leaderboard.

The compact heatmap in Figure 9 keeps the per-skill view without consuming a full page. Each row is one anonymized skill and each column is one primary harness, making the heterogeneity visible even when aggregate lift is positive.

Figure 10(b) shows lift by valid harness–model cell, and Figure 10(c) holds the Codex harness fixed while varying the model. In that slice, the baseline rises sharply for GPT-5.5, so measured Skill Lift shrinks even though absolute task performance remains high. Live evaluation exposes this model–skill interaction; document scanning cannot.

A separate 25-variant routing stress test keeps the production headline fixed while varying the visible-skill count from 1 to 50. Mean overall lift remains at 0.133–0.149 for 1–20 visible skills, while mean wall time rises from 258 seconds at one visible skill to 451 seconds at 20. At 50, the with-skill pass rate falls to 0.55 and mean

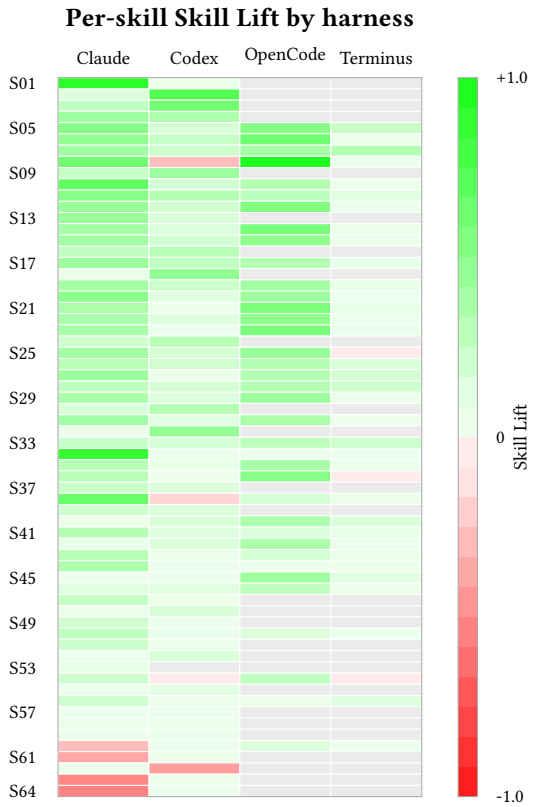

Figure 9: Compact Skill Lift heatmap over the 64 production skills in the trial-file inventory and four primary harnesses. Rows are anonymized skill identifiers; color encodes paired overall Skill Lift. The 947 scored paired cases come from 58 of these skills.

wall time rises to 1,290 seconds. We treat 50 as a routing and latency stress condition rather than part of the production headline.

6.6 Negative Lift as a Debugging Signal

A positive aggregate does not mean every skill helps every agent. Across the 947 paired task cases, 87 have negative overall lift; in the 201-cell production trial-file aggregate, 11 cells have negative mean lift and 11 have zero mean lift at reported precision. These are review targets: paired evaluation turns a regression into a traceable comparison between the with-skill and baseline runs.

The trajectories show two distinct classes. Some negative cells reflect execution instability rather than a substantive skill failure: for example, one with-skill run produced no scored trials while the baseline completed. Cleaner negative cases show the skill changing the agent's behavior in the wrong direction: the agent found or attempted to read the skill, but produced a truncated or meta-level response, skipped verification, or spent extra tool calls without improving the answer. In one anonymized technical-configuration case, activation and some behavior checks improved, but goal accuracy and efficiency fell enough to make overall lift negative. Live evaluation therefore separates "never discovered" from "discovered but misused," both invisible to document scanning.

6.7 Static Scores Are Not Runtime Evidence

Static scores remain valuable for fast review, but they should not be treated as runtime evidence. Of the 64 production skills in the trial-file inventory, 62 have matching doc-scan metadata. On those 62 skills, Tier 1 structural score has Spearman $\rho = -0.0181$ against overall live lift (Fisher-z approximate 95% CI [-0.2667, 0.2327]), and Tier 2 LLM-judge score has Spearman $\rho = -0.0266$ (95% CI [-0.2745, 0.2247]). These correlations are statistically indistinguishable from zero. The live run instead records 2,022 trajectories, 11,642 tool calls, and 9,091 behavior observations: evidence of discovery, selection, workflow execution, recovery, `expected_behavior` satisfaction, and safety.

Public OpenClaw sanitization case. Static review of the public `agent-transcript` skill [29] reported 11/11 checks and 89/100 while the security scan was skipped and code checks missed its renderer. In a separate local $n = 1$ paired run, the renderer succeeded while retaining a synthetic `nvapi-` canary in an intermediate artifact; the agent repaired only the final file. Default accuracy and goal accuracy were 1.0 in both arms, but artifact grading detected the canary only in the with-skill intermediate artifact. This diagnostic is excluded from the 947-case aggregate.

6.8 Trace-Grounded Qualitative Readout

Saved trajectories also give reviewers a qualitative readout that aggregate lift cannot provide. Table 7 summarizes behavior-level examples from the frozen production evidence without exposing internal skill or repository names.

7 Discussion and Limitations

Our results come from a 145-skill corpus spanning internal enterprise skills and public skills from community catalogs; for the live-agent evaluation, we use four primary agent harnesses with uneven cell coverage. Replicating on skills authored by additional organizations and on harnesses we do not currently support would strengthen the generalizability claim. The corpus is skewed toward System Access, Deployment, Platform, and Data Infra skills; we do not claim the same lift distribution for underrepresented categories such as Troubleshooting, Dev-Tooling, or creative skills. The weak Spearman $\rho = 0.14$ between scan methods is evidence of complementary review signals in this corpus and deserves replication. More directly, the near-zero correlations between scan scores and live lift (Tier 1 $\rho = -0.0181$, Tier 2 $\rho = -0.0266$) are consistent with no useful monotonic runtime proxy in these scan scores, rather than evidence of a negative relationship.

Causal interpretation. The paired design holds the task, harness, model, scorer, sandbox, and configured non-target skills fixed, so it is stronger than a single-condition live score. It still does not identify an environment-independent "intrinsic" contribution of a skill. Adding or removing a skill can change routing pressure, context allocation, competition with sibling skills, or interactions with prerequisite skills. We therefore interpret Skill Lift as the target skill's marginal contribution under the declared workspace and baseline policy.

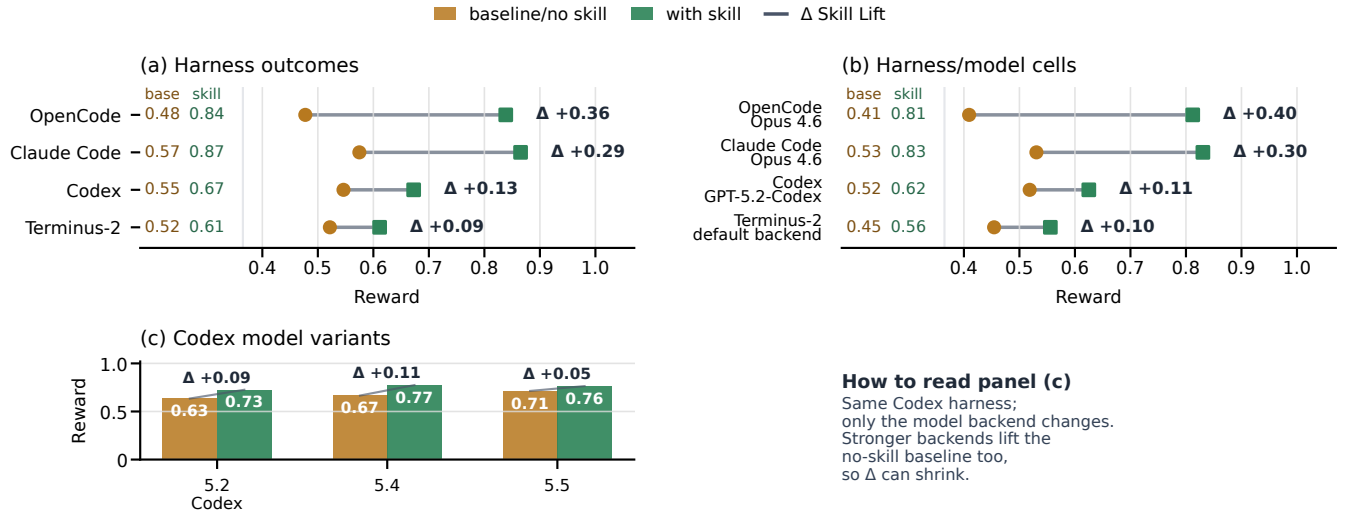

Figure 10: How Skill Lift appears in a live evaluation report. (a) For each agent harness, ACES runs isolated with-skill and baseline jobs from the same evaluation assets, records rewards, and reports the paired difference as Skill Lift. (b) A compact harness–model view keeps the valid agent/model cells visible. (c) A same-harness Codex model sweep shows that model changes alter both absolute rewards and marginal lift; stronger baselines can reduce measured skill value even when the with-skill score remains high.

Table 7: Anonymized qualitative evidence from saved trajectories and run logs.

Case	Trace or log observation	Paper signal
Positive routing and workflow	A public technical workflow skill was activated through the expected tool path, made 1/1 productive tool calls, and satisfied 5/5 expected behaviors.	Shows a concrete mechanism behind high lift: correct routing, tool use, and author-specified workflow compliance.
Partial success	An authentication workflow diagnosed missing OAuth tokens, but did not present the actual authorization URL and did not rerun verification after the missing prerequisite was identified.	Shows why trajectory and behavior checks add nuance beyond final answer text: the agent made progress but skipped a required verification step.
Actionable author feedback	A multi-step artifact workflow trace produced suggestions for explicit polling guidance, retry logic, and clearer remediation steps after an operation remained pending.	Shows that live evaluation can become author feedback, not just a pass/fail score.
Security separation	The security backfill found secret-like strings, destructive command patterns, unauthorized-path access, and missing-trajectory cases across the live artifacts.	Motivates keeping `security` as a separate metric from `behavior_check`, so safety failures are visible even when task behavior otherwise looks acceptable.

Judge-model sensitivity. The reported three-judge spread concerns document-rubric scores, not live trajectory grading. Live inter-judge agreement, human calibration, and judge-uncertainty propagation remain unmeasured.

Model-update caveat. The Codex model slice in §6.5 is a methodological demonstration, not a broad model-quality claim: stronger models can raise both with-skill and baseline scores. Lift may therefore shrink even when absolute task performance improves, because the baseline agent no longer needs as much help from the skill. This motivates scheduled re-evaluation after model updates.

Partial-credit note for internal-network skills. Several corpus skills reach enterprise-only endpoints behind a VPN that the live-agent container does not cross. Script calls on those skills receive network errors; `goal_accuracy` absolute values on those skills are therefore a lower bound. *Skill Lift* (with minus without) is still informative because both conditions see the same network state, but endpoint unavailability can compress or distort absolute success metrics.

Cost and deployment. Scans run on every change; live paired evaluation runs for release candidates, high-risk skills, or reviewer-requested changes, with token and cost accounting in each report. The routing stress study shows why the most crowded workspace is a targeted release test rather than a per-edit default.

Other caveats. We report confidence intervals for the headline paired-case lift and cluster sensitivity checks, but richer subgroup intervals and formal hypothesis tests for negative-lift classes are

follow-up work. The raw-row audit conflates timeouts with other no-score causes, so we do not report a separate timeout rate. The live layer includes a trace-level `security` metric, and `expected_behavior` can add skill-specific safety checks; the external static security scanner remains an integration for supply-chain and prompt-injection analysis, not a contribution of this paper.

Relation to benchmarks. ACES complements recent paired or CI-integrated skill-evaluation systems [2, 16, 31, 35, 38] through a fixed-support baseline, discovery pressure, ATIF-normalized multi-metric traces, and static-versus-live production evidence (Appendix C).

8 Related Work

We organize related work by how each line treats skills. A capability summary at the tool level appears in Table 3 at the end of §2.

Skill-centric evaluation. SkillsBench [11, 32] evaluates skills as a condition in an agent benchmark over a fixed 84-task set and reports pass-rate deltas between no-skills, curated-skills, and self-generated-skills configurations. It establishes that skills matter; it does not score individual skills as first-class artifacts, does not integrate with developer CI, and does not evaluate on the skill author's own tasks. The in-the-wild study [22] asks whether agents can retrieve the right skill from a 34k-skill corpus and shows that benefits degrade under realistic retrieval; it does not score per-skill quality or marginal value. SkillTester [35] evaluates skill artifacts for utility and security in isolation, adopting comparative-utility and user-facing-simplicity principles; it does not run a live agent. ACES combines these angles—live agent evaluation on author-owned tasks, marginal-value measurement with configured supporting or decoy skills, trajectory-grounded dataset refinement, BYOT/BYOG extension points, and CI-native deployment.

Agent and tool-use benchmarks. Terminal-Bench [24], SWE-bench [19], AgentBench [21], GTA-2 [36], MCP-Bench [1], and MCPEval [30] evaluate *agents* on fixed task sets; skills are not first-class. Our methodology is orthogonal: given an arbitrary skill, we evaluate whether its presence helps the agent on the skill's own tasks.

LLM-as-Judge methodology. G-Eval [12, 23] established structured judges for human-aligned NLG quality. RAGAS [13] provides reference-based and reference-free checks for retrieval-augmented generation. We use this pattern for `goal_accuracy`. Recent process-evaluation work for agentic systems shows why final-answer scoring is incomplete: it can hide skipped steps, hallucinated tool use, or bypassed procedures even when the outcome is graded as correct [15]. This motivates ACES's trajectory-level behavior checks alongside final-answer metrics. These methods are building blocks for ACES; our contribution is the skill-specific composition: six trajectory metrics, free-form developer-authored behavior checks, BYOG extension points, and paired with-skill/baseline Skill Lift under fixed supporting or decoy skills.

Agent execution and trace infrastructure. Harbor [17, 18] provides a sandboxed agent execution framework with containerized tasks and an `ATIF`-native agent interface. ACES builds on that infrastructure rather than claiming it: the ACES adapter and task emitter translate evaluation assets into paired runtime environments, capture or normalize trajectories into ATIF, and feed the same grading and Skill Lift reporting layer across harnesses.

Industry practice. Vendor documentation and community guides describe skill authoring and evaluation [3, 5–8, 20, 26, 27, 33, 34]. These inform our dataset schema, four-bucket generation, and `expected_behavior` convention. Broader continuous-evaluation and MLOps guidance argues for running evals across changing AI artifacts during development and operations [9, 10, 14, 28, 37]; ACES specializes that idea to skill artifacts. None of these sources currently provides an open, multi-tier pipeline covering scan, rubric, live agent, and repository-native review.

9 Conclusion and Future Work

Today's skill-evaluation tooling scans the skill document; ACES adds what scanning cannot see: a live agent running the skill, compared against a paired baseline. On the 145-skill mixed-source corpus we find that the two scan methods barely agree (Spearman $\rho = 0.14$)—even before adding runtime, the scans do not converge. The live layer provides the runtime missing piece: across 947 scored paired cases from 58 of 64 production skills, with-skill runs improve composite score by 0.2134 on average (95% paired-case CI [0.1967, 0.2301]) and outcome-only score by 0.1799. The largest gains appear in skill execution, behavior checking, and efficiency, giving reviewers evidence about discovery, workflow following, routing, and tool use rather than only final-answer quality. The same paired traces also expose negative-lift cases, where a skill introduces routing overhead, incomplete execution, or harness-specific regressions that a document scan would miss. Paired with-skill and baseline runs are normalized into ATIF, graded by the six-metric ACES default suite and optional domain-specific metrics, mapped into five stakeholder dimensions, and summarized as Skill Lift. The ACES adapter and task emitter stage fresh task environments across agents, task sources, workspace modes, grading modes, and sandbox backends, including isolated and group workspaces for skill-selection and prerequisite-skill scenarios. BYOT and BYOG let teams keep product-specific tasks and graders while ACES supplies the paired-run protocol, trajectory capture, aggregation, and reporting. Repository automation makes the workflow routine on pull or merge requests, turning evaluation assets into the skill analogue of a test suite. NVIDIA SkillEvaluator provides the public open-source implementation so other teams can apply the same protocol to their own skill catalogs [25].

Future work. Immediate follow-ups include longitudinal studies of Skill Lift across model updates, broader cross-organization validation on externally authored skill corpora, automated selection of representative decoy or prerequisite skill sets, richer subgroup confidence intervals, and formal tests for negative-lift classes.

References

[1] Accenture et al. 2025. MCP-Bench: Benchmarking Tool-Using LLMs with the Model Context Protocol. *arXiv preprint arXiv:2508.20453* (2025). https://arxiv.org/abs/2508.20453

[2] Tejas Singh Anand, Yuet Ying Christina Wang, Wanting Jiang, Steve Masson, Tian Zheng, and Bingjie Zhou. 2026. AEVAL: From Anecdotal to Deterministic Testing for Agentic Skill Workflows. *arXiv preprint arXiv:2607.16345* (2026). https://arxiv.org/abs/2607.16345

[3] Anthropic. 2025. Agent Skills: Best Practices. Claude Platform Documentation. https://platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices Accessed April 2026.

[4] Anthropic. 2025. Equipping Agents for the Real World with Agent Skills. Anthropic Engineering Blog. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills Accessed April 2026.

[5] Anthropic. 2025. How to Create Skills: Key Steps, Limitations, and Examples. Claude Blog. https://claude.com/blog/how-to-create-skills-key-steps-limitations-and-examples Accessed April 2026.

[6] Anthropic. 2025. skill-creator: Building Agent Skills. GitHub repository anthropics/skills. https://github.com/anthropics/skills/tree/main/skills/skill-creator Accessed April 2026.

[7] Anthropic. 2025. Skills Notebooks: Introduction. Claude Platform Cookbook. https://platform.claude.com/cookbook/skills-notebooks-01-skills-introduction Accessed April 2026.

[8] Anthropic. 2025. Teach Claude Your Way of Working Using Skills. Claude Support. https://support.claude.com/en/articles/12580051-teach-claude-your-way-of-working-using-skills Accessed April 2026.

[9] Anthropic. 2026. Demystifying Evals for AI Agents. Anthropic Engineering Blog. https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents Accessed May 2026.

[10] Jialong Chen, Xander Xu, Hu Wei, Chuan Chen, and Bing Zhao. 2026. SWE-CI: Evaluating Agent Capabilities in Maintaining Codebases via Continuous Integration. *arXiv preprint arXiv:2603.03823* (2026). https://arxiv.org/abs/2603.03823

[11] Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, 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 preprint arXiv:2602.12670* (2026). https://arxiv.org/abs/2602.12670

[12] Confident AI. 2024. G-Eval: The Definitive Guide. Confident AI Blog. https://www.confident-ai.com/blog/g-eval-the-definitive-guide Accessed April 2026.

[13] Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. RAGAS: Automated Evaluation of Retrieval Augmented Generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL): System Demonstrations*. https://arxiv.org/abs/2309.15217

[14] Google Cloud. 2024. MLOps: Continuous Delivery and Automation Pipelines in Machine Learning. Google Cloud Architecture Center. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning Accessed May 2026.

[15] Milan Gritta, Debjit Paul, Xiaoguang Li, Lifeng Shang, Jun Wang, and Gerasimos Lampouras. 2026. Process Evaluation for Agentic Systems. In *Findings of the Association for Computational Linguistics: EACL 2026*. Association for Computational Linguistics, Rabat, Morocco, 2678–2692. doi:10.18653/v1/2026.findings-eacl.140

[16] Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. 2026. SWE-Skills-Bench: Do Agent Skills Actually Help in Real-World Software Engineering? *arXiv preprint arXiv:2603.15401* (2026). https://arxiv.org/abs/2603.15401

[17] Harbor Framework. 2026. Harbor: A Framework for Sandboxed Agent Task Evaluation. Project homepage. https://www.harborframework.com/ Accessed April 2026.

[18] Harbor Framework. 2026. Harbor BaseAgent API Reference. Harbor documentation. https://mintlify.wiki/harbor-framework/harbor/api/base-agent Accessed April 2026.

[19] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *The Twelfth International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2310.06770

[20] LangChain. 2025. Evaluating Skills. LangChain Blog. https://www.langchain.com/blog/evaluating-skills Accessed April 2026.

[21] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. AgentBench: Evaluating LLMs as Agents. In *The Twelfth International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2308.03688

[22] Yujian Liu et al. 2026. How Well Do Agentic Skills Work in the Wild: Benchmarking LLM Skill Usage in Realistic Settings. *arXiv preprint arXiv:2604.04323* (2026). https://arxiv.org/abs/2604.04323

[23] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2511–2522. https://aclanthology.org/2023.emnlp-main.153

[24] Mike A. Merrill and Terminal-Bench Team. 2026. Terminal-Bench 2.0: Evaluating Autonomous Agents in Realistic Terminal Environments. *arXiv preprint arXiv:2601.11868* (2026). https://arxiv.org/abs/2601.11868

[25] NVIDIA Corporation. 2026. SkillEvaluator. Software repository. https://github.com/NVIDIA/SkillEvaluator Version 0.1.0; Apache-2.0; accessed August 2026.

[26] OpenAI. 2025. Codex Skills. OpenAI Codex Documentation. https://developers.openai.com/codex/skills Accessed April 2026.

[27] OpenAI. 2025. Evaluating Skills. OpenAI Developers Blog. https://developers.openai.com/blog/eval-skills/ Accessed April 2026.

[28] OpenAI. 2026. Evaluation Best Practices. OpenAI API Documentation. https://platform.openai.com/docs/guides/evaluation-best-practices Accessed May 2026.

[29] OpenClaw. 2026. Agent Transcript Skill. OpenClaw Agent Skills Repository. https://github.com/openclaw/agent-skills/tree/2a409d348a4bcf6f15e41e9a20efd0b298a32528/skills/agent-transcript Commit 2a409d348a4bcf6f15e41e9a20efd0b298a32528; accessed August 2026.

[30] Salesforce AI Research. 2025. MCPEval: Deep Evaluation for AI Agents via the Model Context Protocol. *arXiv preprint arXiv:2507.12806* (2025). https://arxiv.org/abs/2507.12806

[31] Maksim Shaposhnikov, Nicolas Fortuin, Simon Stipcich, Maria I. Gorinova, Amy Heineike, and Rob Willoughby. 2026. A Framework for Evaluating Agentic Skills at Scale. *arXiv preprint arXiv:2606.17819* (2026). https://arxiv.org/abs/2606.17819

[32] SkillsBench Team. 2026. SkillsBench: Benchmarking How Well Skills Work Across Diverse Tasks. SkillsBench project site. https://www.skillsbench.ai/ Accessed April 2026.

[33] Spillwave. 2025. Mastering Agentic Skills: The Complete Guide to Building Effective Agent Skills. Spillwave Publication. https://pub.spillwave.com/mastering-agentic-skills-the-complete-guide-to-building-effective-agent-skills-d3fe57a058f1 Accessed April 2026.

[34] Vercel. 2025. Agent Skills Explained: An FAQ. Vercel Blog. https://vercel.com/blog/agent-skills-explained-an-faq Accessed April 2026.

[35] Leye Wang, Zixing Wang, and Anjie Xu. 2026. SkillTester: Benchmarking Utility and Security of Agent Skills. *arXiv preprint arXiv:2603.28815* (2026). https://arxiv.org/abs/2603.28815

[36] Yang Wang et al. 2026. GTA-2: Benchmarking General Tool Agents from Atomic Tool-Use to Open-Ended Workflows. *arXiv preprint arXiv:2604.15715* (2026). https://arxiv.org/abs/2604.15715

[37] Boming Xia, Qinghua Lu, Liming Zhu, Zhenchang Xing, Dehai Zhao, and Hao Zhang. 2024. An Evaluation-Driven Approach to Designing LLM Agents: Process and Architecture. *arXiv preprint arXiv:2411.13768* (2024). https://arxiv.org/abs/2411.13768

[38] Dexu Yu et al. 2026. SkillAudit: From Fixed-Suite Benchmarking to Skill-Centered Assessment. *arXiv preprint arXiv:2606.22613* (2026). https://arxiv.org/abs/2606.22613

A Metric Contract and Weighting

Table 8 defines the six default metrics normalized to $[0, 1]$. Equal 1/6 weights are an inspectable diagnostic default, not a claim that the metrics are equally causal or valuable; repositories may override the policy or add BYOG metrics, provided reports retain every component. Because behavior check and skill execution are process metrics that the baseline may be unable to satisfy when the target skill is withheld, we also report an outcome-only view:

$$L_{\text{outcome}} = \frac{1}{2}\left(L_{\text{accuracy}} + L_{\text{goal accuracy}}\right).$$

For the frozen headline this is $(0.1431 + 0.2167)/2 = 0.1799$. Reporting both views separates final task outcomes from discovery, routing, workflow, and tool-use signals.

Table 8: Default ACES metric definitions. All metrics are normalized to $[0, 1]$ before aggregation; higher is better.

Metric	Contract
Security	*Evidence:* deterministic scan of trace commands, paths, observations, and outputs; repository scanners remain separate. *Scale:* 1 absent unsafe patterns, reduced by findings. *Weight/view:* 1/6; Security. *Caveat:* pattern false positives/negatives; missing trajectories cannot be fully scanned.
Skill execution	*Evidence:* expected activation, script/tool use, and workflow order. *Scale:* fraction of checks completed. *Weight/view:* 1/6; Discoverability and Effectiveness. *Caveat:* equivalent workflows absent from the dataset may be under-credited.
Skill efficiency	*Evidence:* skills read, relevant versus irrelevant skill/tool calls, and productive calls. *Scale:* routing/tool-efficiency fraction. *Weight/view:* 1/6; Efficiency and Discoverability. *Caveat:* exploration can look inefficient; visible-skill count changes routing pressure.
Accuracy	*Evidence:* LLM-as-Judge assessment of answer and trajectory against task and ground truth. *Scale:* $[0, 1]$. *Weight/view:* 1/6; Correctness. *Caveat:* judge-model and wording sensitivity.
Goal accuracy	*Evidence:* reference-based outcome match. *Scale:* RAGAS/LLM goal score in $[0, 1]$. *Weight/view:* 1/6; Correctness and Effectiveness. *Caveat:* artificially low when tasks require unavailable network or product endpoints.
Behavior check	*Evidence:* LLM yes/no judgments of ordered `expected_behavior` assertions against the ATIF trajectory. *Scale:* fraction satisfied. *Weight/view:* 1/6; Effectiveness, plus Security for safety assertions. *Caveat:* phrasing-sensitive; can reward process despite weak final success.

B Evidence Audit and Sensitivity Checks

Table 6 records the filtering and evidence inventory for the frozen headline. All six default metrics are present for the 947 scored paired cases. Composite deltas are 689 positive, 171 zero, and 87 negative, with median 0.1717.

Missingness and conservative imputation. A subsequent raw-row audit reconstructed 898 of 947 frozen pairs, closely reproducing composite (0.2153), accuracy (0.1431), goal-accuracy (0.2180), and outcome-only (0.1805) lift. Among 972 baseline and 970 with-skill rows, missing or unreconstructible trajectories affect 46 (4.73%) and 8 (0.82%), respectively; 40 case keys have a baseline score but no with-skill score, while 33 have the reverse. Imputing every baseline-scored/with-skill-unscored case as a with-skill failure makes composite lift fall from 0.2153 to 0.1840 but remain positive; accuracy and goal-accuracy lift fall to 0.1123 and 0.1902. The export conflates timeouts with other no-score causes, so no separate timeout rate is reported. This reconstructible-subset check does not replace the frozen 947-case estimate.

Routing scale and operational cost. The 25-variant stress study evaluates five target skills at 1, 5, 10, 20, and 50 visible skills across two harnesses. Mean overall lift stays at 0.133–0.149 for 1–20 visible skills while mean wall time rises from 258 seconds at one to 451 at 20. At 50, the with-skill pass rate is 0.55 and mean wall time is 1,290 seconds, versus 0.725 at one visible skill. We treat 50 as a routing/latency stress condition and reserve live evaluation for release candidates, high-risk skills, and reviewer-requested changes rather than every edit.

C Positioning Against Recent Skill Benchmarks

Recent skill-centered work spans paired utility, retrieval, per-skill task generation, and CI-integrated contracts [2, 11, 16, 22, 31, 35, 38]. ACES claims neither pairing nor CI alone; it combines fixed-support baselines, discovery pressure, author-owned assets, ATIF-normalized multi-metric traces, and static-versus-live production evidence.

Table 9: Scope comparison with two representative skill-evaluation studies.

Dimension	SkillsBench	Skills in the Wild	ACES
Primary unit	Shared tasks with curated skills	Queries against a large real-world corpus	Repository skill or product capability package under review
Paired skill/no-skill	Yes	Yes	Yes
Retrieval pressure	Bundle-size experiments	Yes	Configurable sibling/decoy skills
CI gate for arbitrary production-skill changes	No	No	Yes
Cross-harness trajectory contract	No	No	Yes, via ATIF

D Trace-Grounded Diagnostic Examples

Table 10 gives reviewer-requested cases that passed the scan gate but exposed additional live evidence. Internal identifiers are generalized because raw traces contain internal paths and product details.

Table 10: Static review and live evidence answer different questions. Tier 1/Tier 2 document scores use 0–100; live deltas are matched with-skill minus baseline.

Case	Static, live, and trace evidence
Public OpenClaw sanitization	*Static:* 11/11; quality 89/100; security scan skipped; code check missed renderer. *Live ($n = 1$, local):* renderer retained a synthetic `nvapi-` canary in an intermediate artifact; the agent repaired the final file. *Artifact:* canary absence was 0 with skill, 1 at baseline. *Action:* add NVIDIA-token redaction and fail-closed tests.
Enterprise authentication workflow	*Static:* 84 / 88; both pass. *Live:* positive lift, but incomplete workflow. *Trace:* diagnosed missing OAuth tokens but omitted the authorization URL and did not rerun verification, yielding concrete authoring actions.
Enterprise artifact workflow	*Static:* 82.8 / 75; both pass. *Live:* accuracy -0.0667, goal accuracy -0.1667, behavior check -0.3333 in one harness. *Trace:* exposed missing polling/retry guidance and a less complete workflow.