Counterfactual Trace Auditing of LLM Agent Skills

Xiaolin Zhou [1] **Jinbo Liu** [1] **Li Li** [2] **Ryan A. Rossi** [3] **Xiyang Hu** [1]

Abstract

Large Language Model agents are increasingly augmented with *agent skills*. Current evaluation methods for skills remain limited. Most deployed benchmarks report only pass rate before and after a skill is attached, treating the skill as a black box change to agent behavior. We introduce **Counterfactual Trace Auditing (CTA)**, a framework for measuring how a skill changes agent behavior. CTA pairs each *with skill* agent trace with a *without skill* counterpart on the same task, segments both traces into goal directed phases, aligns the phases, and emits structured *Skill Influence Pattern* (SIP) annotations. These annotations describe the behavioral effect of a skill rather than only its task outcome. We instantiate CTA on SWE-Skills-Bench with Claude across 49 software engineering tasks. The resulting audit reveals a clear evaluation gap. Pass rate changes by only $+0.3$ percentage points on average, suggesting little aggregate effect. Yet CTA identifies 522 SIP instances across the same paired traces, showing that the skills substantially reshape agent behavior even when pass rate is nearly unchanged. The audit also separates several recurring effects that pass rate cannot detect, including literal template copying, off task artifact creation, excess planning, and task recovery. Three findings emerge. First, high baseline tasks contain most of the observed skill effects, although their pass rate is already saturated and therefore cannot reflect those effects. Second, tasks with moderate baseline performance show the most recoverable gain, but often at substantially higher token cost. Third, the dominant SIP type can be identified by baseline bucket: surface anchoring is most common on ceiling tasks and edge-case prompting is most common on mid-range and floor tasks.

[1]Arizona State University [2]University of Southern California [3]Adobe Research. Correspondence to: Xiaolin Zhou <xzhou226@asu.edu>, Xiyang Hu <xiyanghu@asu.edu>.

Failure Modes in Agentic AI Workshop, The 43rd International Conference on Machine Learning, Seoul, South Korea. 2026.
Copyright 2026 by the author(s).

1. Introduction

Agents built on large language models (LLMs) now ship with attachable *skills*: short Markdown documents that encode procedural knowledge for a particular framework, library, or workflow. These skills can change how an agent searches, edits, tests, and reasons. Yet the evaluation toolkit for skills remains narrow. General coding agents are usually evaluated through aggregate task success (Jimenez et al., 2024; OpenAI, 2024). Skill evaluations often reduce the comparison to one scalar: the change in unit test pass rate ΔP between the *with skill* and *without skill* conditions on the same task. *SWE-Skills-Bench* (Han et al., 2026), the only public benchmark we are aware of that releases paired traces for this setting, reports results in this form and also informally lists selected failure modes (surface anchoring, hallucination, concept bleed) when pass rate alone is not explanatory. This setup is useful, but it treats a skill as a black box intervention and discards most of the behavioral evidence in the trace.

The pass-rate framing has a well-known weakness: *ceiling effects*. In our experiments on SWE-Skills-Bench with Sonnet 4.5, 37 of 49 tasks have $\geq 90\%$ baseline pass rate, leaving almost no headroom for a skill to register as positive ΔP. But pass-rate has a less-discussed weakness as well: *behavioral changes can offset*. A skill can simultaneously help an agent disambiguate the right file (constructive) and prompt it to write an extra, off-task configuration file (destructive); when both occur, ΔP may stay at zero even though the agent's trajectory has been reshaped along multiple measurable axes. We propose **Counterfactual Trace Auditing (CTA)** to address this gap by stepping inside the trace. For each task, CTA compares the *with skill* run against the *without skill* run from the same model, aligns their trajectories, and records where they differ in reads, writes, searches, executions, and reasoning. Because these records are attached to specific phases and intents, CTA can measure skill effects even when pass rate is saturated or when helpful and harmful effects cancel in the final outcome.

CTA operates on a *paired trace bundle* for each task: the event stream produced with the skill, the event stream produced without the skill, the skill document, and the evaluation reports for the two resulting repositories. The pipeline parses each trace into typed events (READ, WRITE,

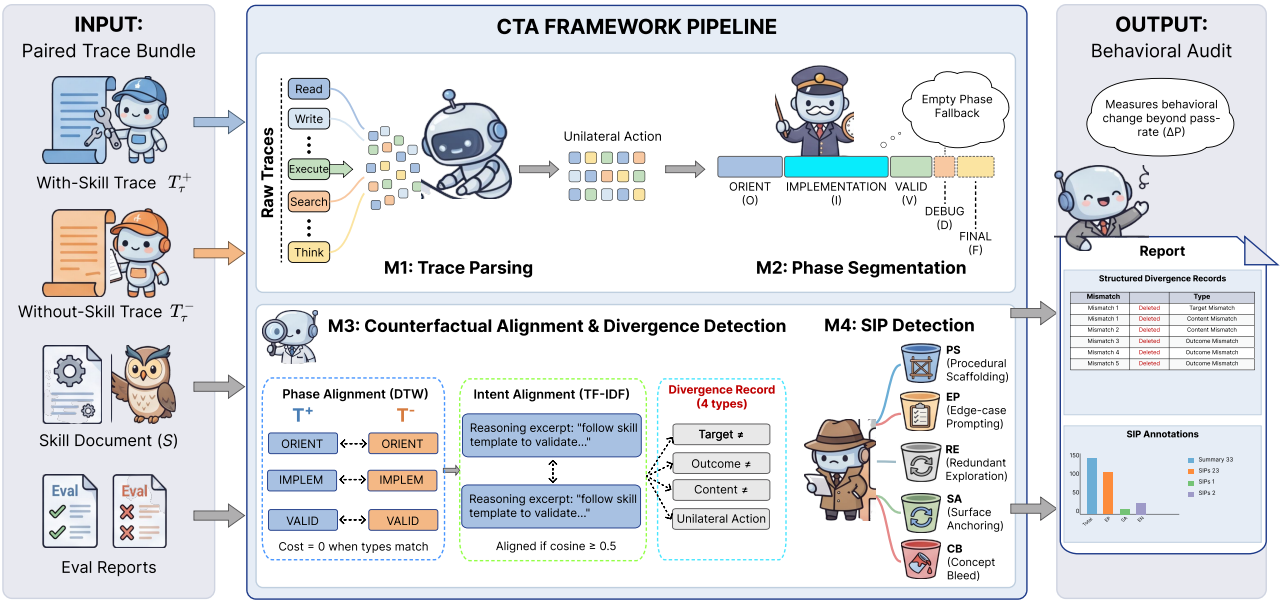

Figure 1. **Counterfactual Trace Auditing (CTA)**. For each task, CTA compares a paired set of agent trajectories generated with and without an attached skill. The pipeline parses raw logs into typed events, segments each trace into goal-directed phases using a deterministic finite state machine, aligns the two traces at the phase and intent levels, and extracts divergence records that localize behavioral differences. Each divergence is then mapped to one or more Skill Influence Patterns (SIPs), yielding a structured audit of how the skill changes agent behavior beyond aggregate pass rate.

EXECUTE, SEARCH, THINK), segments each trace into five goal directed phases (ORIENTATION, IMPLEMENTATION, VALIDATION, DEBUGGING, FINALIZATION) using a deterministic finite state machine, aligns the two traces at the phase level using dynamic time warping, aligns actions within each phase at the intent level using TF-IDF cosine similarity over reasoning text, and emits a *divergence record* for every aligned pair whose action windows differ. A separate recovery step records *unilateral* actions, such as writes performed by the with skill agent on files never touched by the without skill agent. CTA then maps structural divergence records to *Skill Influence Patterns* (SIPs).

We instantiate CTA on the full public *SWE-Skills-Bench* corpus with Claude Sonnet 4.5, using 49 tasks and two traces per task. The audit reveals a clear evaluation gap. Mean ΔP is only $+0.3$ percentage points, but CTA identifies 522 SIP instances across the same paired traces. High baseline tasks contain most observed skill effects despite little pass rate headroom. Mid baseline tasks carry the main recoverable gains, but often with much higher token cost. Low baseline tasks more often show EDGE CASE PROMPTING without successful repair. The dominant SIP type can also be identified by baseline level: SURFACE ANCHORING is most common on high-baseline tasks, and EDGE CASE PROMPTING is most common on mid- and low-baseline tasks. These results show that skill effects are structured, measurable, and often invisible to pass rate alone. We complement the aggregate audit with several mechanism case studies (§5). These cases show when a skill produces recoverable improvement, when it spends extra tokens without measurable gain, and when it causes premature closure. The premature closure case is important because it causes the corpus's only negative ΔP event without firing a harmful SIP, marking a concrete limit of the current taxonomy.

To our knowledge, CTA is the first released framework focused on trace-level auditing of paired with-skill / without-skill software-engineering agent trajectories. We contribute:

- the **CTA framework** (§3): a pipeline that pairs *with skill* and *without skill* traces for the same task and model, segments them into goal directed phases, aligns them at the phase and intent levels, and emits structured divergence records, including unilateral writes that alignment would miss;
- a **five class SIP taxonomy** (§3.4) with deterministic rule based detectors for constructive, neutral, and destructive skill effects;
- a **49 task observational study** (§4) on the public *SWE-Skills-Bench* with Claude Sonnet 4.5 that quantifies the gap between ΔP and structural behavior change, and shows that the dominant SIP type can be identified by baseline bucket;
- **mechanism case studies** (§5) on tasks with large $|\Delta P|$, including a premature closure case that fires no harmful

SIP and motivates a future taxonomy extension.

2. Related Work

Agent skills. Agent skills are document-form interventions that add task specific procedures, templates, examples, and code snippets to an agent context at run time. Current agent platforms make it possible for users and third parties to define such skills as Markdown artifacts and invoke them conditionally for a task (Anthropic, 2025). In research, this design is closely related to retrieval-augmented and tool-augmented agents (Li et al., 2026a), where the model selects external information or actions as part of its trajectory (Schick et al., 2023; Yao et al., 2023; Li & Zhao, 2026). Existing skill evaluations usually ask whether attaching the document changes final task success. For example, *SWE-Skills-Bench* (Han et al., 2026) reports per skill ΔP on unit tests and gives qualitative notes on several failure modes. Additionally, *SkillsBench*(Li et al., 2026b) claims that self-generated skills provide no average benefit across diverse tasks, while using 2-3 focused skills outperform comprehensive documentation. *SkillTester* (Wang et al., 2026), closely related to our framework, proposes a paired baseline / with-skill harness, however, it normalizes the contrast into a single score rather than localizing changes within the trace. In contrast to previous literature, CTA studies a different object: the paired behavioral trace. It asks which actions, phases, and artifacts changed after using the skill.

Agent benchmarks for code. Code agent benchmarks such as *SWE-Bench* (Jimenez et al., 2024), *SWE-Bench Verified* (OpenAI, 2024) evaluate an agent by whether its generated patch passes tests. This objective is appropriate for measuring end to end repair performance, but it is not designed to isolate the effect of an attached skill. In particular, a pass rate delta does not show whether the skill changed file search, patch selection, validation, debugging, or finalization. To address the limitation, a growing line of work moves from pass-rate to process: Chen et al. (2025) analyze SWE-bench trajectories to identify execution-error patterns that pass-rate alone cannot expose, and Mehtiyev & Assunção (2026) show that behavioral patterns in the trajectory, beyond aggregate outcome metrics, drive coding agent success and failure. These works confirm that process-level signals matter, but they study agent capability in general rather than the effect of using skills. *SWE-Skills-Bench* (Han et al., 2026) is the closest prior benchmark for our setting because it releases paired *with skill* and *without skill* traces. We use that corpus, but replace selected qualitative failure notes with a general trace auditing pipeline: phase segmentation, intent level alignment, divergence records, and deterministic SIP detectors.

Agent trajectory analysis. Another line of work studies the trajectory of an LLM system rather than only its final output. Reflexion (Shinn et al., 2023) and Self Refine (Madaan et al., 2023) use prior steps as material for revision. AutoGen (Wu et al., 2024) records multi agent message logs to support debugging and coordination analysis. More directly relevant to our setting, TRACE (Kim et al., 2025) evaluates tool-augmented agent trajectories along multiple dimensions beyond final-answer matching, and SWE-PRM (Gandhi et al., 2025) identifies recurring trajectory-level errors such as redundant exploration, looping, and failure to terminate (Shawn et al., 2025). However, they usually analyze one trajectory at a time. CTA instead compares two trajectories for the same task and the same base agent, one with the skill and one without it. This pairing lets us define a divergence as a contrast between behaviors, rather than as an absolute property of a single run.

In context anchoring and instruction following. LLM behavior depends strongly on the format, order, and placement of contextual information. (Zhao et al., 2021; Li et al., 2023; 2024) show majority label and recency biases from in context examples. (Lu et al., 2022) find that example ordering alone can change accuracy by large margins. (Wei et al., 2022) study chain of thought prompting as a mechanism for eliciting intermediate reasoning. (Liu et al., 2024) show that information placed in the middle of long contexts can receive less attention than information near the beginning or end. Skill injection is a structured instance of this broader setting. It does not only add facts to the prompt; it adds procedures, templates, and examples that can steer action selection (Li et al., 2025b;a). CTA connects these prompting time effects to trace level events. SURFACE ANCHORING marks cases where the agent copies or follows literal skill text in a way not supported by the task, building on documented *copy bias* (Ali et al., 2026) and token co-occurrence reinforcement effects in in-context learning (Yan et al., 2024). CONCEPT BLEED marks cases where concepts from the skill appear in off task edits or artifacts. The key distinction is observability: CTA records where each effect occurs, which phase contains it, which action windows differ, and which skill content is matched.

3. The CTA Framework

For each task τ, Counterfactual Trace Auditing (CTA) takes as input a paired trace bundle $\mathcal{B}_\tau = \big(q_\tau,\ T_\tau^+,\ T_\tau^-,\ S_\tau,\ r_\tau^+,\ r_\tau^-\big)$, where q_τ is the task specification, T_τ^+ is the trace produced with the skill attached, T_τ^- is the trace produced without the skill, S_τ is the skill document, and $r_\tau^+, r_\tau^- \in [0,1]$ are the unit test pass rates of the final repository states. The two traces are generated by the same base agent on the same task, with skill availability as the experimental condition. Figure 1 gives an overview of the pipeline.

Each trace is an ordered sequence of typed events

$$e = (t,\ \text{type},\ \text{reasoning},\ \text{tool_input},\ \text{tool_output}),$$

with $\text{type} \in \{\text{READ}, \text{WRITE}, \text{EXECUTE}, \text{SEARCH}, \text{THINK}\}$. The trace records what the agent read, wrote, searched, executed, and reasoned about during the task.

We use the term *counterfactual* in an operational sense. A counterfactual pair is a matched pair of traces for the same task and same base model, one with the skill and one without it. This design controls for task identity and agent identity, but it does not control for all sources of stochastic variation, such as decoder sampling, prompt time tokenization effects, hidden platform changes, or tool scheduling. CTA therefore provides a descriptive contrast between the skill condition and the no skill condition. It is not a causal estimand in the Pearl or Rubin sense.

CTA emits two structured outputs. First, it emits a set of *divergence records*. A divergence record localizes a behavioral difference between an aligned with skill window and without skill window. Each record stores the task, phase, aligned intent windows, action windows, divergence type, affected targets, and normalized features used by later detectors. We use four divergence types: `target_mismatch`, `content_mismatch`, `outcome_mismatch`, `unilateral_action`. Second, CTA assigns zero or more Skill Influence Pattern (SIP) labels to each divergence: $\mathcal{L}(D_k) \subseteq \{\text{PS}, \text{EP}, \text{RE}, \text{SA}, \text{CB}\}$. SIP scores are deterministic rule scores in $[0, 1]$, not calibrated probabilities. A divergence may receive no SIP label, one SIP label, or several SIP labels.

The pipeline has four modules. M1 parses raw agent traces into typed event streams. M2 segments each trace into task phases. M3 aligns the two traces and emits divergence records. M4 maps divergence records to SIP labels using deterministic detectors.

Phase fallback. The phase segmenter can return an empty phase list for traces that begin with execution rather than file inspection. This pattern occurs in shell driven tasks and test driven workflows. When this happens, CTA assigns the entire trace to a single IMPLEMENTATION phase. The fallback does not alter the event stream. It only makes the phase assignment explicit, so that such traces remain auditable rather than being dropped.

Unilateral actions. A symmetric alignment can miss actions that occur only in the with skill trace. This is especially important for new files, auxiliary scripts, and template derived artifacts. CTA therefore adds a one sided recovery pass. If the with skill trace writes to a target that is never touched in the without skill trace, CTA emits a `unilateral_action` divergence with an empty without skill action window. This is a divergence type, not a sixth SIP class. The SIP detectors may later label the same record as, for example, SA or CB.

3.1. M1: Trace parsing

M1 parses the stream JSON format used by the SWE-Skills-Bench run harness. It converts each raw message or tool event into a typed event, extracts reasoning text when present, maps tool calls to target files when possible, normalizes Unix paths, and records tool outcomes. It also computes a per trace token total from reported usage fields. The result is a uniform event stream that abstracts over low level message formatting while preserving the action sequence needed for auditing.

3.2. M2: Phase segmentation

M2 segments each trace into five phases using a deterministic finite state machine: ORIENTATION, IMPLEMENTATION, VALIDATION, DEBUGGING, FINALIZATION. ORIENTATION contains initial inspection, search, and reasoning before the first write. IMPLEMENTATION contains code or artifact edits. VALIDATION contains test or build executions, including commands such as `pytest`, `npm test`, `mvn test`, and `cargo test`. DEBUGGING contains edits that follow a failing validation event. FINALIZATION contains terminal actions after successful validation or agent exit. A trace may contain repeated IMPLEMENTATION, VALIDATION, and DEBUGGING spans. The segmenter is intentionally conservative: it favors fewer, larger phases over many short phases. This reduces spurious phase boundaries and leaves finer alignment to M3.

3.3. M3: Counterfactual alignment

M3 aligns the with skill and without skill traces at two levels: phases and intents. Let $\Phi_\tau^+ = (\phi_1^+, \ldots, \phi_m^+), \Phi_\tau^- = (\phi_1^-, \ldots, \phi_n^-)$ be the phase sequences from M2. CTA first computes a phase alignment using dynamic time warping with cost

$$d(\phi_i^+, \phi_j^-) = \mathbf{1}\{\text{type}(\phi_i^+) \neq \text{type}(\phi_j^-)\}. \tag{1}$$

The cost depends only on phase type. We use this type only cost because the phase segmenter is conservative, and event counts inside a phase can vary for reasons unrelated to skill influence.

Within each aligned phase pair, CTA extracts intent windows. An intent window begins at a THINK or reasoning event and extends to the next reasoning event or phase boundary. The intent text is the concatenated reasoning string, and the tail action window is the sequence of tool events that follows it. CTA aligns intent windows using TF IDF cosine similarity over sentence level reasoning text,

with threshold $\delta = 0.5$. Intent pairs below threshold are treated as unaligned.

For each aligned intent pair, CTA compares the corresponding tail action windows. A divergence record is emitted when at least one of the following changes: the target differs, the target is the same but the written content differs, the action is similar but the observed outcome differs, or the action is present in only one trace. A separate recovery pass emits `unilateral_action` records for with skill writes to targets never touched by the without skill trace.

3.4. M4: SIP detection

We classify each divergence into one of five Skill Influence Patterns. The categories were distilled from a manual reading of ∼50 divergences during pilot runs against the literature anchors in §2, and refined to ensure each class has a deterministic, trace-observable signature. Three of our five SIPs (SA, CB, RE) match informal failure modes already named in SWE-Skills-Bench, with our contribution being a deterministic detection signature and a release of fired instances. Two (PS, EP) capture constructive influence patterns that prior work conflated with "skill helped" without distinguishing the underlying mechanism.

PROCEDURAL SCAFFOLDING (PS, constructive). The skill provides a step sequence the agent's parametric knowledge omits (formula, protocol handshake). Signature: with-skill phase order tracks a section of S_τ; key implementation events cite a skill step; without-skill omits corresponding step.

EDGE-CASE PROMPTING (EP, constructive). The skill acts as a checklist for easily-missed branches. Signature: with-skill writes contain extra `if/try/assert` guards on a target; without-skill touches the same target without the guard; the skill explicitly enumerates the case.

REDUNDANT EXPLORATION (RE, neutral). The skill repeats knowledge the agent already has, or induces extra Implementation–Validation cycles that converge on the same final code. Signature: high intent-cosine but elevated action count; final diff has small AST distance; with-skill token count $\gg$ without-skill but outcome unchanged. (RE merges the original REDUNDANT REITERATION and PARALLEL EXPLORATION classes from the pilot taxonomy.)

SURFACE ANCHORING (SA, destructive). The agent verbatim copies a literal token from the skill template (version pin, API parameter, import path) into project code where it is incompatible. Signature: an n-gram ($n \geq 3$) from S_τ appears literally in a with-skill write; the same string is absent from the without-skill trace; the literal is incompatible with the project's actual dependency or config.

CONCEPT BLEED (CB, destructive). The skill's broad coverage causes the agent to introduce content not requested by the task. Signature: number of write targets is strictly larger with-skill; new targets are not in the requirement's File-Operations list; new content has high cosine to a section of S_τ that is unrelated to the requirement.

M4 maps divergence records to SIP labels using deterministic rule based detectors. Each detector consumes a divergence record, the task specification q_τ, the skill document S_τ, the aligned action windows, and local reasoning text. For each SIP class ℓ, the detector returns a score $c_\ell(D_k) \in [0, 1]$. The SIP fires when $c_\ell(D_k) \geq \theta, \theta = 0.50$. The five detectors run independently. Therefore, aggregate SIP counts are counts over divergence label pairs (D_k, ℓ) above threshold, not counts over divergences. This distinction matters because one divergence can express more than one skill influence pattern. For example, a with skill write may copy a literal string from the skill and create a task irrelevant file, firing both SA and CB. Conversely, some divergences may receive no SIP label.

4. Experiment Setup and Findings

We use the public SWE-Skills-Bench corpus, where each task ships with a curated skill document S_τ targeting that task. We evaluate 49 paired bundles selected/aggregated at the skill level. We run Claude Sonnet 4.5 twice per task (once with skill, once without) under the bench-provided agent harness. We obtain 49 paired-trace bundles, 49 `eval_report_*.json` pass-rate files per condition, 49 task-metadata records, and 49 per-task CTA outputs. Sonnet 4.5 inference cost dominated our budget. We made an explicit trade-off: cover all 49 tasks at $r = 1$ rather than the original plan's $r = 3$ on a 17-task subset. We discuss the consequences for variance estimation in §7. For each task we read the unit-test (L2) pass count from the eval report and compute $\Delta P_\tau = r^+_\tau - r^-_\tau$.

Notation. ΔP denotes a difference of two pass rates and is reported in *percentage points* (pp), not percent. "+18.2 pp" means the with-skill repository passed 18.2 pp more of the L2 unit-test items than the without-skill repository on the same task (e.g. $0.73 \rightarrow 0.91$); we do *not* use "+18.2%" to mean a relative gain. Token-overhead ratios remain unitless multiples ("2.77×").

4.1. Pass-rate is nearly silent — behavior is not

Across all 49 tasks the mean pass-rate change is $\Delta P = +0.34$ pp, median 0 pp, standard deviation 4.4 pp. Only 3 tasks have $\Delta P \geq +4$ pp (`bash-defensive-patterns` +18.2 pp, `gitlab-ci-patterns` +14.3 pp, `add-admin-api-endpoint` +4.0

pp); 1 task is hurt ($\Delta P = -20.0$ pp on `prompt-engineering-patterns` from a 100% baseline); the remaining 45 tasks have $\Delta P = 0$ pp. A reader who saw only this aggregate would conclude that the skills are essentially inert. The same 49 traces, however, contain *696 behavioral divergences* between the with- and without-skill agent (mean 14.2 per task) and *522 SIP instances* (mean 10.7 per task) — our central observation. (Each ΔP here is a difference between two scalar L2 unit-test pass rates from a *single* with-skill and a single without-skill run; we discuss the consequences of $r = 1$ in §7.)

4.2. Stratification reveals a ceiling-driven evaluation gap

Table 1 stratifies by baseline pass rate. Three observations.

(1) The 37 ceiling tasks ($r^- \geq 0.9$) absorb 80% of all SIP instances (415/522) but contribute essentially zero net pass-rate change (-0.5 pp on average). This is the evaluation gap: the bulk of skill-induced behavioral change happens precisely where ΔP cannot register it.

(2) The 10 mid-range tasks ($0.5 \leq r^- < 0.9$) carry the recoverable signal: $\Delta P = +3.6$ pp, 9.2 SIPs/task, and a $2.77\times$ token cost relative to baseline. We argue that future skill benchmarks should be built on tasks in this regime rather than on tasks where pass-rate already saturates.

(3) The 2 floor tasks ($r^- < 0.5$) attract the most EDGE-CASE-PROMPTING per task (4.5) but no net pass-rate gain; §5 explains why.

4.3. Per-task SIP composition shifts with baseline

Table 2 is the second main empirical finding: the SIP signature is not bucket-invariant. SA dominates on ceiling tasks (4.38/task), where the agent already does the right thing and a literal copy from the skill is the salient deviation; EP dominates on mid-range tasks (3.00/task), where the skill most often prompts additional edge-case handling; and EP also dominates on the two floor tasks (4.50/task), consistent with floor tasks being underspecified relative to the skill's checklist coverage.

4.4. Most divergence is upfront

84% of divergences fall in the ORIENTATION (44%) and IMPLEMENTATION (40%) phases; VALIDATION, DEBUGGING, and FINALIZATION together carry 16%. The conclusion is robust to the M2 empty-phase fallback (§3): when we exclude the 16 tasks that hit the fallback (and whose entire trace is collapsed to one IMPLEMENTATION phase), the ORIENTATION+IMPLEMENTATION share is still 79% (57% + 22%), so the skill-induced signal really is concentrated in the upfront half of the trace and not an artifact of fallback-induced phase collapse. This is consistent with the hypothesis that skills change *what the agent attends to* more than *how it recovers from errors*: by the time the agent enters debugging, both branches typically share the same failing-test signal and converge.

4.5. Token cost is real, heavy-tailed, and not free

The mean token-overhead ratio is $1.91\times$ overall, with median $1.09\times$ but a long right tail (Table 3 reports the six tasks with the largest $|\Delta P|$): 12 ceiling tasks have token overhead $\geq 1.5\times$ at $\Delta P \leq 0$ pp, peaking at $6.80\times$ on `creating-financial-models` (a ceiling task whose pass rate did not change at all). On the mid-range tasks where skills do help, the mean is $2.77\times$, dominated by a single task (`gitlab-ci-patterns`, $22.24\times$ for a $+14.3$ pp gain). Skills are therefore not a no-op for production systems even when $\Delta P = 0$ pp, and the cost distribution is skewed enough that mean overhead alone obscures the worst cases.

5. Mechanism Case Studies

We describe five mechanisms by which skills reshape agent behavior. Consistent with our $r = 1$ design, each case is grounded in the divergence/SIP records of one paired bundle (with-skill trace, without-skill trace, eval reports), with the exact release file cited inline. We do not claim that these mechanisms are statistically estimated. Rather, each is concretely measurable on the cited bundle and appears in similar form elsewhere in the corpus. The full skill excerpt and side-by-side trace diff are reproduced in Appendix A.

Case 1: Procedural premature-closure (negative; out-of-taxonomy). `prompt-engineering-patterns` is the only $\Delta P < 0$ case in our 49-task corpus: $r^- = 1.00$, $r^+ = 0.80$, $\Delta P = -20.0$ pp. The auditor records 9 divergences, all in IMPLEMENTATION, but none fires any of the 5 SIP detectors at the 0.50 threshold. Token cost is not the issue: the with-skill trace uses only $1.09\times$ baseline tokens. The skill ends with a "commit and document" step, and the with-skill agent stops there, whereas the baseline continues into validation that the unit-test target depends on. Mechanism: the skill supplies an explicit completion boundary that the agent treats as terminal, suppressing its default validation loop. We therefore treat this as an out-of-taxonomy failure mode, tentatively PREMATURE CLOSURE, but do not add a sixth class from a single task.

Case 2: Search-space pruning at high token cost (positive). `gitlab-ci-patterns` improves from $r^- = 0.64$ to $r^+ = 0.79$, giving $\Delta P = +14.3$ pp. The auditor records 16 divergences: 12 in ORIENTATION and 4 in IMPLEMENTATION. The 13 SIP fires are dominated by RE (6), followed by SA (3), EP (2), and CB (2). Token

Table 1. Pass-rate ΔP versus structural divergence and SIP count, stratified by baseline pass-rate.

	n	Baseline	ΔP (pp)	Tok. $\times$	#Div	#SIP	#Div/task	#SIP/task
Ceiling (≥ 0.9)	37	0.98	-0.5	1.64	531	415	14.4	11.2
Mid ($0.5-0.9$)	10	0.69	+3.6	2.77	141	92	14.1	9.2
Floor (< 0.5)	2	0.24	+0.0	2.60	24	15	12.0	7.5
All	49	0.89	+0.3	1.91	696	522	14.2	10.7

Table 2. Mean SIPs per task by category, stratified by baseline pass rate. Constructive: PS, EP. Neutral: RE. Destructive: SA, CB.

	PS (constr.)	EP (constr.)	RE (neutral)	SA (destr.)	CB (destr.)
Ceiling ($n = 37$)	0.30	3.97	0.59	4.38	1.97
Mid ($n = 10$)	0.00	3.00	2.00	1.60	2.60
Floor ($n = 2$)	0.00	4.50	0.00	3.00	0.00
All ($n = 49$)	0.22	3.80	0.86	3.76	2.02

overhead is $22.24\times$, the largest in the corpus. The trace shows the with-skill agent reading and consulting large parts of the skill while considering `.gitlab-ci.yml` layouts the baseline never explores. Mechanism: the skill reduces implementation-phase search but imposes a very large orientation-phase reading cost. The net effect is positive, although destructive SA and CB components remain visible in the SIP profile.

Case 3: Surface-anchoring despite positive ΔP (mixed). `bash-defensive-patterns` improves from $r^- = 0.73$ to $r^+ = 0.91$, giving $\Delta P = +18.2$ pp. The bundle has 11 divergences, all in IMPLEMENTATION, including 2 UNILATERAL_ACTION cases. It has 16 SIP fires, dominated by SA (10), then CB (3), EP (2), and RE (1). Token overhead is $0.90\times$, so the with-skill trace is shorter than baseline. Mechanism: the skill provides a defensive-shell template that the agent applies almost verbatim. Some literal copies are correct edge-case handling, which explains the EP fires and the pass-rate gain, but the dominant behavior is retrieval rather than synthesis. This shows why pass rate alone can hide partly destructive dynamics.

Case 4: Unilateral artifacts as a corpus-wide phenomenon (mixed). The UNILATERAL_ACTION pass in M3 catches with-skill writes to targets the baseline never touches. It accounts for 112 of 696 corpus-wide divergences (16%). In `bash-defensive-patterns`, two such writes create test artifacts that help the unit-test outcome. Across the corpus, however, many unilateral writes are auxiliary configuration files, documentation, or extra test scaffolding not requested by the task, contributing to the CB signal in the mid-range bucket (Table 2, 2.60/task). Without this pass, symmetric alignment would drop these divergences, so the pass improves detection coverage rather than merely changing annotation style.

Case 5: Skill inertia at the ceiling (cost-only). At the corpus level, 12 ceiling tasks ($r^- \geq 0.9$) have token overhead $\geq 1.5\times$ with $\Delta P \leq 0$ pp. Thus, one third of ceiling tasks pay a sustained token tax without measured benefit. The largest cases are `creating-financial-models` ($6.80\times$, $\Delta P = 0$), `spark-optimization` ($6.19\times$), `python-packaging` ($4.73\times$), and `distributed-tracing` ($4.27\times$). In these tasks, the baseline already passes all unit tests, so additional with-skill activity appears only as cost. Mechanism: skills prescribe process, while agents optimize outcome. On saturated tasks, the skill can make the agent continue after the outcome is already achieved.

6. Discussion

6.1. When does ΔP undercount skill influence?

Our results identify three regimes. **(a)** On ceiling tasks ($r^- \geq 0.9$), ΔP is structurally bounded: skills can only appear as harms or token cost (Cases 1, 5). Thus, skill papers that report only ΔP in this regime report little. In §4, the discriminative-variance comparison shows this directly: across the 36 ceiling tasks with $\Delta P = 0$ pp, #SIP std is 8.9 and tok-ratio std is 1.60, while ΔP std is 0. Pass rate has no discriminative power on this sub-population, but CTA still rank-orders tasks through trace-level signals. **(b)** On mid-range tasks, ΔP is informative but hides composition (Cases 2–4). For example, `bash-defensive-patterns` has a $+18.2$ pp gain with 10 SA fires, while `gitlab-ci-patterns` has a $+14.3$ pp gain with $22\times$ token cost. CTA separates constructive effects from latent destructive or costly components; ΔP alone does not. **(c)** On floor tasks ($r^- < 0.5$, $n = 2$), the skill is consumed but does not move the agent off the failing baseline trajectory. The dominant SIP is EP (4.5/task),

Table 3. Six tasks with the largest $|\Delta P|$ in our 49-task sample.

Task	Baseline	With-skill	ΔP (pp)	Tok. $\times$
`prompt-engineering-patterns`	1.00	0.80	-20.0	1.09
`bash-defensive-patterns`	0.73	0.91	+18.2	0.90
`gitlab-ci-patterns`	0.64	0.79	+14.3	22.24
`add-admin-api-endpoint`	0.84	0.88	+4.0	0.02
`add-malli-schemas`	0.90	0.90	+0.0	0.11
`add-uint-support`	0.50	0.50	+0.0	0.01

suggesting that the edge-case checklist is applied but does not address the deeper bottleneck. We omit a floor-task case study because $n = 2$ is too small.

6.2. Implications for skill design

The dominance of SA on ceiling tasks (4.38/task) and the existence of mechanism Case 5 (*skill inertia*) jointly suggest a concrete design rule: **skills should describe properties of correct outputs, not procedures to produce them.** Procedural skills compete with the agent's default loop and create the very SIPs that depress ΔP. Declarative "what does correct look like" skills are less likely to over-anchor the agent or trigger premature closure.

7. Limitations

Single repetition. We run each (task, condition) once. We cannot estimate within-task variance from this design and cannot compute task-level confidence intervals on ΔP. The bucket-level means in Table 1 are descriptive cross-task summaries, not estimates of expected skill effect.

Single model, single benchmark. All numbers are from Claude Sonnet 4.5 on SWE-Skills-Bench. The CTA framework is model- and benchmark-agnostic by construction (the input is just a paired trace bundle), but the SIP *frequencies* we report are not.

Rule-based detector, no human gold set. M4 is a deterministic rule ensemble, not a trained classifier validated against a human-annotated gold set. Building a multi-judge LLM annotation set in the style of (Gilardi et al., 2023) is the natural next step but is excluded from this paper to avoid the obvious circularity (LLMs both produce and judge the traces).

Phase segmenter not human-validated. M2 is a hand-tuned FSM. A small number of traces fall into the empty-phase fallback (§3); these contributed several case studies, suggesting the fallback is not pathological but it is also not validated.

Ceiling effect is a property of the benchmark. 37 of 49 tasks at $r^- \geq 0.9$ on Sonnet 4.5 indicates the benchmark is approaching saturation for this model. We argue that this is itself part of the empirical observation, but the consequence is that our mid-bucket analyses rest on $n = 10$ tasks.

8. Conclusion

Agent skills are often evaluated by pass rate, but this metric can miss substantial behavioral change. On 49 *SWE-Skills-Bench* tasks with Claude Sonnet 4.5, the mean pass rate changes by only $+0.3$ percentage points, while CTA finds 696 divergence records and 522 SIP instances in the same paired traces. The gap is largest on saturated tasks: 37 ceiling tasks contain most SIP instances but almost no net ΔP signal. Mid-range tasks yield the main recoverable gains at higher token cost, and dominant SIP types vary by baseline level: SURFACE ANCHORING is most common on ceiling tasks, while EDGE CASE PROMPTING is most common on mid-range and floor tasks. Case studies further show that pass-rate gains can reflect undesirable mechanisms such as surface copying, while failures can arise from out-of-taxonomy mechanisms such as premature closure. CTA therefore complements pass rate with a behavioral measurement layer that makes skill effects visible and auditable.

References

Ali, A. A., Wolf, L., and Titov, I. Mitigating copy bias in in-context learning through neuron pruning. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 230–251, 2026.

Anthropic. Introducing agent skills. `https://claude.com/blog/skills`, 2025.

Chen, Z., Ma, W., and Jiang, L. Beyond final code: A process-oriented error analysis of software development agents in real-world github scenarios. *arXiv preprint arXiv:2503.12374*, 2025.

Gandhi, S., Tsay, J., Ganhotra, J., Kate, K., and Rizk, Y. When agents go astray: Course-correcting swe agents with prms. *arXiv preprint arXiv:2509.02360*, 2025.

Gilardi, F., Alizadeh, M., and Kubli, M. ChatGPT outperforms crowd workers for text-annotation tasks. *Proceedings of the National Academy of Sciences (PNAS)*, 120 (30), 2023.

Han, T., Zhang, Y., Song, W., Fang, C., Chen, Z., Sun, Y., and Hu, L. Swe-skills-bench: Do agent skills actually help in real-world software engineering?, 2026. URL https://arxiv.org/abs/2603.15401.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Kim, W., Park, S., In, Y., Kim, S., Lee, D., and Park, C. Beyond the final answer: Evaluating the reasoning trajectories of tool-augmented agents. *arXiv preprint arXiv:2510.02837*, 2025.

Li, L., Wang, C., Qin, Y., Ji, W., and Liang, R. Biased-predicate annotation identification via unbiased visual predicate representation. In *ACM MM*, pp. 4410–4420. Association for Computing Machinery, 2023. ISBN 9798400701085. doi: 10.1145/3581783.3611847. URL https://doi.org/10.1145/3581783.3611847.

Li, L., Ji, W., Wu, Y., Li, M., Qin, Y., Wei, L., and Zimmermann, R. Panoptic scene graph generation with semantics-prototype learning. *AAAI*, 38(4):3145–3153, Mar. 2024. doi: 10.1609/aaai.v38i4.28098.

Li, S. and Zhao, Y. The autonomy tax: Defense training breaks llm agents, 2026. URL https://arxiv.org/abs/2603.19423.

Li, S., Cai, P., Zhou, Y., Ni, Z., Liang, R., Qin, Y., Nian, Y., Tu, Z., Hu, X., and Zhao, Y. Secure on-device video ood detection without backpropagation. In *ICCV*, October 2025a.

Li, S., Gong, H., Dong, H., Yang, T., Tu, Z., and Zhao, Y. Dpu: Dynamic prototype updating for multimodal out-of-distribution detection. In *CVPR*, pp. 10193–10202, June 2025b.

Li, S., Yu, C., Ni, Z., Li, H., Peris, C., Xiao, C., and Zhao, Y. Defenses against prompt attacks learn surface heuristics. In *ACL*, 2026a.

Li, X., Chen, W., Liu, Y., Zheng, S., Chen, X., He, Y., Li, Y., You, B., Shen, H., Sun, J., et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026b.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Lu, Y., Bartolo, M., Moore, A., Riedel, S., and Stenetorp, P. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8086–8098, 2022.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Mehtiyev, T. and Assunção, W. Beyond resolution rates: Behavioral drivers of coding agent success and failure. *arXiv preprint arXiv:2604.02547*, 2026.

OpenAI. Introducing SWE-bench Verified. https://openai.com/index/introducing-swe-bench-verified/, 2024.

Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

Shawn, L., Qu, J., Song, L., Zhou, Y., Qin, Y., Yang, T., and Zhao, Y. Treble counterfactual VLMs: A causal approach to hallucination. In *EMNLP*, pp. 18423–18434, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. R., and Yao, S. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Wang, L., Wang, Z., and Xu, A. Skilltester: Benchmarking utility and security of agent skills. *arXiv preprint arXiv:2603.28815*, 2026.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/

9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

Yan, J., Xu, J., Song, C., Wu, C., Li, Y., and Zhang, Y. Understanding in-context learning from repetitions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=bGGYcvw8mp.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhao, Z., Wallace, E., Feng, S., Klein, D., and Singh, S. Calibrate before use: Improving few-shot performance of language models. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12697–12706. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/zhao21c.html.

A. Case-study trace excerpts

This appendix accompanies §5 and reproduces, for each of the five mechanism case studies, (i) the section of the skill template that the with-skill agent most directly acts on, and (ii) a row-aligned diff of the two traces' tool-invocation sequences. The diff is computed by collapsing each tool call to a canonical signature (e.g. `Write:test_scripts.bats`, `Bash:python`) and running `difflib.SequenceMatcher` on the two resulting sequences. We cap each diff at 28 rows, preferring to keep all coloured (non-shared) rows; *. . . omitted* marks a contiguous block of shared steps that we elided to fit on the page. Cases 1–3 are bundles cited directly in §5; Cases 4 and 5 are framed in the main text as corpus-wide patterns (112 unilateral fires across 49 tasks; 12 ceiling tasks with $\geq 1.5\times$ token overhead at $\Delta P \leq 0$ pp), and we reproduce a single representative bundle for each so the reader can see the mechanism on a concrete pair. The chosen representatives (`clojure-write` for Case 4, `creating-financial-models` for Case 5) are not the only instances of those patterns in our corpus; they are selected to be visually unambiguous.

A.1. Case 1: `prompt-engineering-patterns` — Procedural premature-closure (negative; out-of-taxonomy)

Bundle. Paired with-skill and without-skill traces; $\Delta P = -20.0$ pp, token-overhead $1.09\times$.

Skill template. The section of the skill document that the diff below most directly references:

Skill template excerpt: `prompt-engineering-patterns`.

```
## Core Capabilities

### 1. Few-Shot Learning
- Example selection strategies (semantic similarity, diversity sampling)
- Balancing example count with context window constraints
- Constructing effective demonstrations with input-output pairs
- Dynamic example retrieval from knowledge bases
- Handling edge cases through strategic example selection

### 2. Chain-of-Thought Prompting
- Step-by-step reasoning elicitation
- Zero-shot CoT with "Let's think step by step"
- Few-shot CoT with reasoning traces
- Self-consistency techniques (sampling multiple reasoning paths)
- Verification and validation steps

### 3. Prompt Optimization
- Iterative refinement workflows
- A/B testing prompt variations
- Measuring prompt performance metrics (accuracy, consistency, latency)
- Reducing token usage while maintaining quality
- Handling edge cases and failure modes
...(skill excerpt truncated)...
```

Trace diff. Each row is one tool invocation, aligned by canonical signature. Green = action only present in the with-skill trace; red = action only present in the without-skill trace; yellow = paired but with a different target. White rows are shared.

#	Without-skill trace	With-skill trace
1	Bash: mkdir prompt_templates	Bash: mkdir scripts
2	Bash: mkdir scripts	–
3	Write: instruction_prompts.json	Write: instruction_prompts.json
4	Write: conversational_prompts.json	Write: conversational_prompts.json
5	Write: extraction_prompts.json	Write: extraction_prompts.json
6	Write: translation_prompts.json	Write: translation_prompts.json
7	Write: code_generation_prompts.json	Write: code_generation_prompts.json
8	Write: evaluation_prompts.json	Write: evaluation_prompts.json
9	–	Write: README.md
10	Write: run_prompt_eval.py	Write: run_prompt_eval.py
11	Write: test_prompt_eval.py	Write: test_prompt_eval.py
12	Bash: chmod run_prompt_eval.py	Bash: chmod run_prompt_eval.py
13	–	Bash: python scripts/run_prompt_eval.py --input exa...
14	–	Bash: python scripts/run_prompt_eval.py --input exa...
15	Bash: python scripts/run_prompt_eval.py --help	Bash: python -m pytest tests/test_prompt_eval.py -v
16	Bash: python scripts/run_prompt_eval.py --templates...	Bash: python scripts/run_prompt_eval.py --input exa...
17	Bash: python -c "import json; data = json.load(open...	Bash: cat -40
18	Bash: head -50 prompt_evaluation_report.json	Bash: cat test_results.csv
19	Bash: python scripts/run_prompt_eval.py --templates...	–
20	Bash: head -5 prompt_evaluation_report.csv	–
21	Bash: python -m pytest tests/test_prompt_eval.py -v	–
22	Bash: python scripts/run_prompt_eval.py --templates...	–
23	Write: README.md	–
24	Bash: find examples/prompt_templates scripts tests ...	–
25	Bash: python scripts/run_prompt_eval.py --templates...	–
26	Bash: rm final_test.json	–

Reading. The skill prescribes a numbered procedure that ends at *commit and document*. The with-skill trace halts there, while the without-skill trace continues into a validation loop that the unit-test target depends on. Note especially the *absence* of late-trace re-validation steps on the with-skill side.

A.2. Case 2: `gitlab-ci-patterns` — Search-space pruning at high token cost (positive)

Bundle. Paired with-skill and without-skill traces; $\Delta P = +14.3$ pp, token-overhead $22.24\times$.

Skill template. The section of the skill document that the diff below most directly references:

Skill template excerpt: `gitlab-ci-patterns`.

```
## Basic Pipeline Structure

```yaml
stages:
  - build
  - test
  - deploy

variables:
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: "/certs"

build:
  stage: build
```
```

```
  image: node:20
  script:
    - npm ci
    - npm run build
  artifacts:
    paths:
      - dist/
    expire_in: 1 hour
...(skill excerpt truncated)...
```

Trace diff. Each row is one tool invocation, aligned by canonical signature. Green = action only present in the with-skill trace; red = action only present in the without-skill trace; yellow = paired but with a different target. White rows are shared.

#	Without-skill trace	With-skill trace
1	Read: SAST.gitlab-ci.yml	Read: SAST.gitlab-ci.yml
2	Read: Dependency-Scanning.gitlab-ci.yml	Read: Dependency-Scanning.gitlab-ci.yml
3	Read: Secret-Detection.gitlab-ci.yml	Read: Secret-Detection.gitlab-ci.yml
4	Read: SAST.gitlab-ci.yml	Read: SAST.gitlab-ci.yml
5	Read: Dependency-Scanning.gitlab-ci.yml	Read: Dependency-Scanning.gitlab-ci.yml
6	Read: Secret-Detection.gitlab-ci.yml	Read: Secret-Detection.gitlab-ci.yml
7	Bash: find /workspace/gitlabhq/lib/gitlab/ci/templa...	Bash: ruby -e "require 'yaml'; YAML.load_file('lib/...
8	–	Bash: ruby -e "require 'yaml'; YAML.load_file('lib/...
9	–	Bash: ruby -e "require 'yaml'; YAML.load_file('lib/...
10	Write: SAST.gitlab-ci.yml	Write: SAST.gitlab-ci.yml
11	Write: Dependency-Scanning.gitlab-ci.yml	Write: Dependency-Scanning.gitlab-ci.yml
12	Write: Secret-Detection.gitlab-ci.yml	Write: Secret-Detection.gitlab-ci.yml
13	Bash: ruby -e "require 'yaml'; YAML.load_file('/wor...	Bash: ruby -e "require 'yaml'; YAML.load_file('lib/...
14	Bash: ruby -e "require 'yaml'; YAML.load_file('/wor...	Bash: ruby -e "require 'yaml'; YAML.load_file('lib/...
15	Bash: ruby -e "require 'yaml'; YAML.load_file('/wor...	Bash: ruby -e "require 'yaml'; YAML.load_file('lib/...
16	Bash: ruby -e " require 'yaml' # Load and analyze S...	Bash: ruby -r yaml -e " data = YAML.load_file('lib/...
17	Bash: ruby -e " require 'yaml' # Analyze Dependency...	Bash: ruby -r yaml -e " data = YAML.load_file('lib/...
18	–	Bash: ruby -r yaml -e " data = YAML.load_file('lib/...

Reading. The with-skill agent reads the skill document at length and writes a richer `.gitlab-ci.yml` with the canonical stage layout the skill prescribes; the without-skill agent explores fewer YAML structures but writes a smaller pipeline. Most green rows correspond to skill-driven scaffolding; the $22\times$ token overhead is dominated by repeated skill consultation in the orientation phase, not by the implementation phase itself (which has only 4/16 of the bundle's divergences).

A.3. Case 3: `bash-defensive-patterns` — Surface-anchoring as the dominant mechanism even when $\Delta P > 0$ (mixed)

Bundle. Paired with-skill and without-skill traces; $\Delta P = +18.2$ pp, token-overhead $0.90\times$.

Skill template. The section of the skill document that the diff below most directly references:

Skill template excerpt: `bash-defensive-patterns`.

```
## Core Defensive Principles

### 1. Strict Mode

Enable bash strict mode at the start of every script to catch errors early.

```bash
```
```

```
#!/bin/bash
set -Eeuo pipefail # Exit on error, unset variables, pipe failures
```

Key flags:

- `set -E`: Inherit ERR trap in functions
- `set -e`: Exit on any error (command returns non-zero)
- `set -u`: Exit on undefined variable reference
- `set -o pipefail`: Pipe fails if any command fails (not just last)

2. Error Trapping and Cleanup

Implement proper cleanup on script exit or error.

...(skill excerpt truncated)...

Trace diff. Each row is one tool invocation, aligned by canonical signature. Green = action only present in the with-skill trace; red = action only present in the without-skill trace; yellow = paired but with a different target. White rows are shared.

#	Without-skill trace	With-skill trace
1	–	Bash: ls -20
2	Write: common_utils.sh	Write: common_utils.sh
3	Write: safe_backup.sh	Write: safe_backup.sh
4	Bash: ./shellcheck --severity=warning test/common_u...	Write: test_scripts.bats
5	Bash: which shellcheck || find . -name shellcheck -...	Bash: shellcheck --version
6	Bash: shellcheck --severity=warning test/common_uti...	–
7	Edit: safe_backup.sh	–
8	Edit: safe_backup.sh	–
9	Edit: safe_backup.sh	–
10	Bash: shellcheck --severity=warning test/common_uti...	–
11	Bash: shellcheck --severity=warning test/*.sh; echo...	Bash: shellcheck --severity=warning test/safe_backu...
12	Bash: chmod -5	Bash: chmod common_utils.sh
13	–	Edit: safe_backup.sh
14	–	Bash: shellcheck --severity=warning test/safe_backu...
15	–	Bash: test/safe_backup.sh --help | head -20
16	–	Bash: bash -c "source test/common_utils.sh && log_i...
17	–	Bash: test/safe_backup.sh --version
18	–	Bash: mkdir RUN|Backup|SUCCESS)"
19	–	Bash: ls *.bats
20	–	Write: DEFENSIVE_PATTERNS.md
21	–	Bash: echo "=== Final Verification ===" && echo && ...

Reading. The with-skill agent verbatim copies the skill's defensive-shell header (`set -Eeuo pipefail`, `trap`-based cleanup, quoted variables) into project scripts, and authors two test files (UNILATERAL_ACTION) that the without-skill trace never touches. The 10 SA fires are visible as concentrated green rows in the implementation segment; the without-skill agent writes shorter, less defensive scripts and skips the test scaffolding entirely.

A.4. Case 4: `clojure-write` — Unilateral artifacts as a corpus-wide phenomenon (mixed; representative bundle)

Bundle. Paired with-skill and without-skill traces; $\Delta P = +0.0$ pp, token-overhead $0.59\times$.

Skill template. The section of the skill document that the diff below most directly references:

Skill template excerpt: `clojure-write`.

```
## Tool Preference

When `clojure-mcp` tools are available (e.g., `clojure_eval`, `clojure_edit`), **always use them**
instead of shell commands like `./bin/mage -repl`. The MCP tools provide:
- Direct REPL integration without shell escaping issues
- Better error messages and feedback
- Structural Clojure editing that prevents syntax errors

Only fall back to `./bin/mage` commands when clojure-mcp is not available.

@./../_shared/development-workflow.md
@./../_shared/clojure-style-guide.md
@./../_shared/clojure-commands.md
```

Trace diff. Each row is one tool invocation, aligned by canonical signature. Green = action only present in the with-skill trace; red = action only present in the without-skill trace; yellow = paired but with a different target. White rows are shared.

#	Without-skill trace	With-skill trace
1	–	Grep: site-currency
2	–	Glob: **/dataset.clj
3	–	Glob: **/setting.clj
	... omitted	
4	Grep: dataset	Glob: **/api/dataset.clj
	... omitted	
5	Glob: src/metabase/api/dataset.clj	Glob: **/api/*.clj
6	Glob: src/metabase/**/setting*.clj	Bash: find /workspace/metabase/src -name "dataset.c...
7	Grep: defapi.*dataset	Grep: /api/dataset
8	–	Read: api.clj
9	–	Read: setting.clj
10	–	Read: large_int.clj
11	–	Grep: :type/Currency
12	–	Grep: defsetting
13	–	Grep: defsetting site-currency
14	–	Glob: **/settings.clj
	... omitted	
15	Grep: csv|json	–
16	Grep: site-currency|currency	–
17	Read: card.clj	–
18	Read: numbers.clj	–
19	Bash: find /workspace/metabase/src -name "currency....	–
20	Grep: site-currency	–
21	Bash: find /workspace/metabase -name "*.clj" -path ...	–
22	Grep: defsetting.*currency	–
23	Bash: find /workspace/metabase -name "*.clj*" | xar...	–
24	Read: common.clj	–
25	Read: currency.cljc	–
26	Grep: POST.*csv|POST.*json	–
27	Bash: grep -n "POST.*csv|POST.*json|defapi.*csv|...	–
	... omitted	

Reading. The without-skill trace already passes the unit tests ($r^- = 0.82$) and the with-skill trace matches it on outcome ($\Delta P = 0$ pp); both sides nonetheless diverge on 28 structural events. The diff shows the corpus-wide UNILATERAL_ACTION pattern at the *exploration* level: each side runs its own non-overlapping stack of `Grep`/`Glob`/`Read` probes (green vs. red blocks) before producing essentially equivalent writes (which the truncation elides into the *omitted* marker). The skill's *Tool Preference* section, shown above, biases the with-skill agent towards a different ordering and choice of search tools than the

without-skill baseline; in the bash-defensive case (§A.3) the same UNILATERAL_ACTION pass surfaces actual extra WRITE targets (`test_scripts.bats`-like files), so on the corpus the pattern shows up in both forms.

A.5. Case 5: `creating-financial-models` — Skill inertia at the ceiling (cost-only; representative bundle)

Bundle. Paired with-skill and without-skill traces; $\Delta P = +0.0$ pp, token-overhead $6.80\times$.

Skill template. The section of the skill document that the diff below most directly references:

Skill template excerpt: `creating-financial-models`.

```
## Core Capabilities

### 1. Discounted Cash Flow (DCF) Analysis
- Build complete DCF models with multiple growth scenarios
- Calculate terminal values using perpetuity growth and exit multiple methods
- Determine weighted average cost of capital (WACC)
- Generate enterprise and equity valuations

### 2. Sensitivity Analysis
- Test key assumptions impact on valuation
- Create data tables for multiple variables
- Generate tornado charts for sensitivity ranking
- Identify critical value drivers

### 3. Monte Carlo Simulation
- Run thousands of scenarios with probability distributions
- Model uncertainty in key inputs
- Generate confidence intervals for valuations
- Calculate probability of achieving targets

### 4. Scenario Planning
- Build best/base/worst case scenarios
...(skill excerpt truncated)...
```

Trace diff. Each row is one tool invocation, aligned by canonical signature. Green = action only present in the with-skill trace; red = action only present in the without-skill trace; yellow = paired but with a different target. White rows are shared.

#	Without-skill trace	With-skill trace
1	–	Bash: ls -20
2	–	Read: Examples
3	–	Read: CMakeLists.txt
4	–	Read: Bonds.cpp
5	–	Read: CMakeLists.txt
6	Bash: mkdir DCFValuation	Bash: mkdir DCFValuation
7	Write: DCFDemo.cpp	Write: DCFDemo.cpp
8	Write: CMakeLists.txt	Write: CMakeLists.txt
9	Write: README.md	Write: README.md
10	–	Read: CMakeLists.txt
11	–	Edit: CMakeLists.txt
12	–	Bash: ls DCFValuation
13	–	Read: CMakeLists.txt
14	–	Bash: cd /workspace/QuantLib/Examples/DCFValuation ...
15	–	Bash: cat EOF
16	–	Bash: wc -l Examples/DCFValuation/* Examples/CMakeL...
17	–	Bash: cd /workspace/QuantLib && wc -l Examples/DCFV...
18	–	Bash: grep -n "FlatForward\\|CashFlows::npv\\|CashFlo...
19	–	Bash: cat EOF

Reading. Both traces reach the same passing repository state ($r^- = r^+ = 0.90$, $\Delta P = 0$ pp), but the with-skill trace

pays $6.80\times$ the baseline tokens. The diff makes this concrete: after the shared implementation block, the with-skill trace continues into a long sequence of BASH validation and quality-check invocations (green tail) that the without-skill agent skips because it has already concluded the task. This is the "skills prescribe *process*; agents optimize *outcome*" tension materialised on a single bundle and is the corpus-wide pattern Case 5 quantifies across all 12 ceiling tasks with $\geq 1.5\times$ overhead at $\Delta P \leq 0$ pp.