THE CONTRACT–INJECTION MISMATCH: AN EXECUTION-CONDITIONED AUDIT OF SKILL SELF-EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving skill frameworks write a reusable skill document from an agent's own task experience, and are accepted by a single average success rate: measured under one benchmark, one base model, and one deployment configuration, the evolution counts as effective once the number goes up. Whether this score represents the skill—whether it holds outside the configuration that produced it, and what it conceals within—has never been examined, yet self-evolved skills are being certified, compared, and deployed on it. We therefore conduct an execution-conditioned audit of SkillOpt, a representative self-evolution system: we compare skill and no-skill conditions on the same task pool, repeat every task five times, and read every rule evolution wrote. The audit finds that the score is a property of the execution condition: $+13.4$ points on SearchQA become -6.2 on OfficeQA, flipping or splitting again when only the base model or the deployment harness changes; even within the original configuration, the score is the residual of 315 tasks improved against 116 harmed. Separating the two sides: the gains are stabilization—turning intermittently correct tasks into always-correct ones—while the costs land on tasks that never failed, where injection introduces error types that did not exist before. The text explains both sides: evolution writes operating rules with preconditions, not new knowledge, yet injection places every rule on every task. Because a rule's preconditions can only reveal themselves in execution, the audit's rule of use is: do not inject by default; retry once with the skill on failure—outperforming both no-skill and full injection on all three benchmarks. In simpler terms: **rules written for some executions are deployed into all of them, and whether they help is only knowable after execution.**

1 INTRODUCTION

Self-evolving skill frameworks write a reusable skill document from an agent's own task experience (e.g., Voyager, SkillOpt, and Trace2Skill (Wang et al., 2023; Yang et al., 2026; Ni et al., 2026)). Acceptance rests on a single average success rate, measured under one benchmark, one base model, and one deployment configuration: if the number goes up, the evolution counts as effective. The accepted document is then injected in full into every execution. The conditions under which the score was obtained do not travel with the score.

Scrutiny of skills themselves has begun to appear. Demystifying Agent Skills codes 8,135 paired trials in controlled experiments across benchmarks, harnesses, and LLMs, finding that a given skill mostly stabilizes existing behavior and that retrieval is an independent bottleneck (Jiang et al., 2026). Agent Skills Can Be Harmful traces task failures back to specific skills after the fact (Dong et al., 2026). OpenSkillEval synthesizes tasks from the open skill ecosystem and finds that having a skill is not the same as using it (Ying et al., 2026). ACES builds repository-native infrastructure for continuous skill-level evaluation (Kevin et al., 2026). Crucially, all of these audits study *given* skills—written by humans or imported from elsewhere—at the granularity of *single* executions: the text is treated as known, and only its consumption is examined. They do not ask where the skill came from—the text a self-evolution process writes is unknown until it is read—or whether the aggregate score that certifies it holds outside its configuration—what the mean conceals is unknown until it

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

is recomputed. Yet self-evolved skills are being certified, compared, and deployed on exactly this unread text and unchecked score.

This paper does precisely that for SkillOpt: an execution-conditioned audit that reads the text and recomputes the score. The audit holds the task pool, the executor, and the evaluation protocol fixed, and runs skill and no-skill conditions on the same tasks, pairing outcomes task by task—so that outcome differences are attributable to the skill text alone. Every task is executed five times—separating lucky successes from reproducible ones. Official outcomes are consulted only after the protocol is frozen—keeping post hoc selection out of the procedure. Before any comparison, we verified how the system actually deploys its product: in all 105 examined cells, the evolved document is injected in whole into every execution, so the audit examines the deployed practice itself, not a weakened configuration.

Under this protocol, the first observation is that the score changes with its execution condition, and not at the margin. The same skill earns $+13.4$ points on SearchQA and -6.2 on OfficeQA, with all five training seeds on each side agreeing in sign. The task axis is not special: holding the benchmark fixed on OfficeQA and changing only the base model turns the estimate to $+9.9$ on Kimi-K2.6; holding the system fixed and changing only the deployment harness splits SearchQA into $+12.5$ and -5.5. Finally, even within the configuration where it was obtained, $+13.4$ is not the whole story: it is the residual of 315 tasks improved against 116 harmed.

That residual is precisely where both sides of the skill's effect hide. We therefore separate gains from costs, reading the text and the paired outcomes side by side. The gains are stabilization: nearly all tasks that become always-correct on SearchQA come from the band that was already intermittently correct (256 tasks), while 15 of 497 never-correct tasks are recovered. The costs land on tasks that never failed: among tasks the no-skill executor solved five out of five times, 9.28 percentage points fall out of correctness into an error type absent without injection (value or evidence mismatch, $0\% \rightarrow 9.0\%$). One text explains both sides: evolution writes operating rules with preconditions—answer formats, evidence checks—not new knowledge, yet injection places every rule on every task. We call this the *contract–injection mismatch*: rules written for some executions are deployed into all of them.

Under the mismatch, deployment becomes a decision: inject, or not. The decision cannot be made before execution—the task text, execution features, and per-task history all fall short—while the pass/fail outcome after one execution settles it. The resulting rule—do not inject by default; retry once with the skill on failure—beats both extremes on all three benchmarks: on SearchQA it reaches 59.1%, from 37.0% without the skill and 50.4% under full injection.

The rest of this paper is structured as follows. Appendix A situates our audit among work on skill self-evolution and skill auditing. Section 2 describes the audited system and the protocol. Sections 3.1–3.4 answer the four questions the mismatch forces: whether the score travels beyond its configuration and what the mean conceals within it (Section 3.1); what evolution wrote (Section 3.2); where gains and harms land, and through which execution stage (Section 3.3); and when the skill should be used (Section 3.4). Section 4 distills recommendations for evaluators, method builders, and deployers; Section 5 concludes. Strategies that failed validation are reported alongside those that passed.

2 Skill Self-Evolution: Common Interface and Representative Methods

To audit skill self-evolution, we first fix its object and interface. This section introduces the skill document through a pair of real artifacts, formalizes the execution interface shared by existing methods (Shinn et al., 2023; Zhao et al., 2023; Wang et al., 2025; 2023; Ouyang et al., 2025; Ni et al., 2026; Yang et al., 2026), lists the five skill conditions under evaluation ordered by how the document is produced, and then walks through the self-evolution loop that is the primary object of our audit.

2.1 The Skill Document and the Execution Interface

Figure 1 shows a pair of real documents. The *human skill*—a document written by hand—consists of a title and one placeholder sentence. The *self-evolved skill* produced by the same system is instead

Human skill (in full)	**Self-evolved skill (excerpts)**
# Question Answering Skill (No learned rules yet. Rules will be added through the reflection process.)	– Match the grammatical number (singular/plural) and article usage of the answer to the question's phrasing. If the question uses a plural noun (e.g., 'Bagpipes'), answer with the plural form. – If the question asks for a property (e.g., 'higher in this vitamin'), answer with the property name alone ('C', not 'Vitamin C'). – For well-known entities, use the standard name ('Lake Superior', not just 'Superior').

Figure 1: A pair of real skill documents (SearchQA). Left: the human skill in full. Right: excerpts from the self-evolved skill produced by the same system; all content consists of answer-format and extraction rules. The placeholder sentence of the human skill remains in the body of the evolved document.

a set of operational rules: which answer form to extract under which question type, how to match grammatical number and articles, when to use the canonical name rather than the full form. None of this content comes from an external corpus; it is written by the evolution process out of execution experience. It is the entire text injected at deployment and consumed per task, and the entire object of our audit.

Let the executor model M have frozen weights, the benchmark task pool be $\mathcal{D} = \{t_i\}_{i=1}^{N}$, and the deployment harness be h (direct chat or an agentic harness). The skill document S is placed verbatim into a fixed section of the system prompt; execution and scoring on task t are

$$(\hat{y}_t,\ \tau_t)\ \sim\ \pi_M\big(\cdot\ \big|\ \underbrace{P_{\text{sys}}(h) \oplus \text{Skill}(S)}_{\text{system prompt}} \oplus t\big), \qquad c_t = \text{Eval}(\hat{y}_t,\ y_t) \in \{0, 1\}, \tag{1}$$

where $\oplus$ denotes concatenation, τ_t the full execution trajectory, and c_t the hard correctness returned by the benchmark evaluator. The same document serves every task in the pool, with no per-task selection or rewriting; model weights, tools, and the execution flow are fixed. The skill document is therefore the only source of behavioral differences across conditions, task scores are jointly determined by the tuple $(S, M, h, \mathcal{D})$, and the promise of any self-evolution method can be stated as: find S that raises the expected pool accuracy $\mathbb{E}_{t\sim\mathcal{D}}[c_t]$.

2.2 A Spectrum of Skill Conditions

The conditions differ only in how S is produced. Let S_{h} denote the human skill, G a one-shot generator, and the iterative training loop be Eqs. (3)–(5). The five evaluated conditions are

$$\underbrace{S = \varnothing}_{\text{noskill}}, \qquad \underbrace{S = S_{\text{h}}}_{\text{human}}, \qquad \underbrace{S = G(\text{spec})}_{\text{skillcreator}}, \qquad \underbrace{S = S^{\star}_{\text{official}},\ S^{\star}_{\text{self-trained}}\ (\text{Eq. 6})}_{\text{skillopt}}. \tag{2}$$

noskill and human give the capability floor and the human-written baseline. The skillcreator condition is written in one shot by a strong model from the benchmark specification and then frozen; in our audit it enters as a fixed external file, its generation process lies outside the audited system, and the comparison target is the document itself. The skillopt conditions rewrite S repeatedly from execution experience, starting at the human skill; they are the primary object of the audit, and their two instances—official release and self-trained—are described next.

2.3 SkillOpt: The Audited Self-Evolution Loop

SkillOpt (Yang et al., 2026) treats the skill document as a trainable parameter and organizes its rewriting with the discipline of neural-network training: the executor M is frozen, a separate optimizer model $\mathcal{O}$ produces the edits, and the task pool is split before training into a training set $\mathcal{D}_{\text{train}}$ and a validation subset $\mathcal{V}$. One training step proceeds in four beats. The executor runs every task in the current batch $B_k \subset \mathcal{D}_{\text{train}}$ under S_k, leaving trajectories and scores. The optimizer reads these trajectories, attributes common causes among failures and extracts generalizable patterns from successes, and emits bounded add/delete/replace edit patches

$$\{\delta_j\} = \mathcal{O}\Big(S_k,\ \{(\tau_i, c_i)\}_{t_i \in B_k}\Big). \tag{3}$$

The patches are semantically merged, ranked by relevance, and at most ℓ of them are applied to S_k, yielding a candidate

$$S'_k = \mathrm{Apply}\Big(S_k,\ \mathrm{Top}_\ell\big(\mathrm{Aggregate}(\{\delta_j\})\big)\Big), \tag{4}$$

where ℓ acts as a textual learning rate with a decay schedule (default batch size 40, ℓ=4). The candidate is then scored on the validation subset and accepted only on strict improvement:

$$S_{k+1} = \begin{cases} S'_k, & J_{\mathcal{V}}(S'_k) > J_{\mathcal{V}}(S_k), \\ S_k, & \text{otherwise.} \end{cases} \tag{5}$$

At each epoch boundary, the system compares the previous and current skills on identical items, injects longitudinal guidance into the document, and maintains a cross-epoch strategy memory consulted by later reflection. Training terminates at

$$S^\star = \arg\max_k\ J_{\mathcal{V}}(S_k), \tag{6}$$

the document with the highest validation score; deployment involves no optimizer calls. We run this loop with the method's official codebase, unchanged: among the five conditions, skillopt_ckpt is the officially released skill document and skillopt_self is trained by us with the same implementation; both are instances of the same method and differ only in training provenance.

The five conditions share Eq. (1) and differ only in the provenance of S. The interface folds $(S, M, h, \mathcal{D})$ into a single average score that carries neither the configuration that produced it nor the within-pool effect of S. Testing both is a prerequisite for any comparison made through this interface.

3 Experiments

Our evaluation proceeds in four steps. We first test whether a single average score can represent the skill, and find its effect flipping sign across tasks, base models, and deployment conditions (Section 3.1). Opening the skill text then shows what evolution actually writes—operational conventions tied to the evolver and the target model (Section 3.2). Because these conventions carry premises, the same text helps some tasks and harms others; we locate both effects on the task map and trace their mechanisms (Section 3.3). Finally, because the gains and the damage leave observable traces, we turn this analysis into a deployment policy: when to use the skill (Section 3.4). All numbers are recomputed from logged paired executions without model calls; each table's caption states its pool, conditions, and statistic. System, benchmarks, models, harnesses, training configuration, statistics, and the deterministic audits behind every content-level table are given in Appendix B, where Table 7 maps each table to its exact pool and protocol.

3.1 Can a Single Average Score Represent a Self-Evolved Skill?

The evaluation practice for self-evolved skills measures one average success rate under one benchmark, one base model, one deployment configuration, and a handful of training seeds. This subsection asks two questions: does the score travel beyond the configuration in which it was obtained (*transferability*), and within the original configuration, is it complete (*what the average summarizes*)?

Transferability: three condition axes. All tests use a same-pool paired design: the skill condition and the no-skill condition run on the same task set, the same executor, and the same evaluation protocol, and we report per-task paired accuracy differences Δ in percentage points. As a prerequisite, we verified whole-document injection in every examined cell (105/105 single-hash injections; Appendix B), which removes selective per-task rule loading as a confound.

The first test of transferability is the task axis (Table 1). Within one system, all five seeds are positive on SearchQA ($+11.6$ to $+14.2$, all $p \leq 0.001$) and all five are negative on OfficeQA (-3.5 to -7.6); the pooled estimates are $+13.4$ [11.9, 15.0] and -6.2 [-9.3, -3.4]. The within-benchmark seed standard deviations are 0.7 and 1.1 points, an order of magnitude smaller than the 19.6-point gap between the two columns; the reversal of direction therefore belongs to the task axis. A score measured on SearchQA carries limited information about the skill's behavior on OfficeQA.

Table 1: The task axis. Skill sources are five independently self-evolved skills (training seeds s1–s5); the baseline is the no-skill condition on the same task pools (SearchQA and OfficeQA; task ID sets identical to the baseline within each column). Reported are per-task paired hard-accuracy differences Δ (percentage points) and the counts of improved/harmed tasks. Single-seed rows use the exact McNemar test; the pooled row uses a task-cluster bootstrap (5,000 resamples) 95% CI with Holm correction within each column. The human row is the human-written seed skill as a control. $^{*}p \leq 0.05$, $^{**}p \leq 0.01$, $^{***}p \leq 0.001$.

Seed	SearchQA Δ (improved/harmed)	OfficeQA Δ (improved/harmed)
s1	$+14.2^{***}$ (307/108)	-4.7 (6/14)
s2	$+14.2^{***}$ (315/116)	-7.6^{*} (6/19)
s3	$+11.6^{***}$ (292/130)	-7.6^{**} (4/17)
s4	$+13.1^{***}$ (315/132)	-7.6^{**} (4/17)
s5	$+14.1^{***}$ (319/122)	-3.5 (9/15)
Pooled	$\mathbf{+13.4}$ **[11.9, 15.0]*****	$\mathbf{-6.2}$ **[**-9.3**,** -3.4**]****
Human control	-1.9^{*} $[-3.2, -0.7]$	-6.1

The second test fixes the task pool and varies the base model (Table 2). The four systems split in direction: Qwen2B is significantly negative (-6.2), Kimi-K2.6 significantly positive ($+9.9$), while DeepSeek-V3.2 and Qwen3.6 have intervals covering zero. Direction diverges from the no-skill baseline: Qwen2B and Kimi-K2.6 start from similarly low baselines (13.1% and 17.4%), yet end on opposite sides. A score measured on OfficeQA carries limited information about another base model.

The third test fixes the system and varies the deployment condition (Table 10). Without a harness the effect is near zero ($+0.7$ $[-2.4, +3.8]$; seeds alternate in sign). Once a harness is introduced, the two harnesses give opposite pooled effects on SearchQA ($+12.5$ $[+5.0, +20.5]$ vs. -5.5 $[-11.5, -0.5]$), and the counts of positive seeds on OfficeQA and SpreadsheetBench flip between them (3/1 vs. 1/5; 1/5 vs. 3/5). A score measured under one deployment carries limited information about another deployment.

Table 2: The base-model axis. The benchmark is fixed to OfficeQA (full same-pool tasks; task ID sets within each row identical to the baseline). Rows compare four self-evolution systems, each a base model with its own evolved product (Qwen2B, Qwen3.6, DeepSeek-V3.2, Kimi-K2.6; same local chat harness). The per-seed column lists paired differences under independent training seeds; the pooled Δ is the seed mean. The Qwen2B row's CI and p come from a task-cluster bootstrap (Holm-corrected); the Qwen3.6 row uses a seed-level bootstrap approximation; single-seed (†) rows use the exact McNemar test.

Base model	Per-seed Δ	Pooled Δ [95% CI]	p
Qwen2B (5 seeds)	-4.7 / -7.6 / -7.6 / -7.6 / -3.5	$\mathbf{-6.2}$ $[-9.3, -3.4]$	0.0014 (Holm)
Qwen3.6 (5 seeds)	-2.3 / $+2.3$ / $+5.2$ / $+2.9$ / -4.7	$+0.7$ $[-2.7, +3.7]$	— (CI covers 0)
DeepSeek-V3.2 †	—	-1.2	0.84
Kimi-K2.6 †	—	$+9.9$	0.0046

Is the average complete within its own configuration? Transferability concerns whether the score travels; within the original configuration, the average still carries limited information. The improvement/harm counts in Table 1 show that SearchQA's $+14.2$ net is the residue of 315 tasks improved against 116 harmed: about 31% of the pool changed, $2.2\times$ the net. The human control row reveals the residue's composition: on OfficeQA the evolved product's harm nearly equals the human seed's (-6.2 vs. -6.1), while on SearchQA the evolved product exceeds the human seed by $+15.3$. The net gain concentrates on particular task types, and their composition is lost in the average.

Stage summary. Evidence on three condition axes points to the same place: the $+13.4$ measured on SearchQA becomes -6.2 on OfficeQA (Table 1), $+9.9$ on Kimi-K2.6 (Table 2), and splits into $+12.5$ and -5.5 across deployment conditions alone (Table 10). The value and the sign of an

average score change with the conditions that produced it; a score should be cited together with its configuration. Within the original configuration, the number is still the residue of 315 improvements against 116 harms: it overstates the skill's transfer range and omits its action surface. A sound skill evaluation must therefore carry two pieces of information: the configuration it was measured in, and which tasks improved and which were harmed within it. To understand why the same skill helps and hurts under different conditions, one must open the skill text and examine what evolution actually wrote.

3.2 What Does Self-Evolution Write into the Skill?

Section 3.1 put the score back into its configuration; the inconsistencies within a configuration all land on the same object, the skill text itself. Skills are injected as whole documents into every task: every rule written in the text is present on every task. This subsection examines three questions in turn: what content evolution writes; whose execution needs this content serves; and whether a skill produced by a different model under the same evolution method still fits.

What is written: operational conventions. The 189 rules are operational conventions: where to read, how to compute, what format to deliver, what to do on error (Table 11); domain facts are absent from the set. The two largest categories are output formatting (43 rules) and evidence verification (35), and they are rediscovered in nearly every training run (11–12 of 14, spanning all models and benchmarks). Retrieval, navigation, and fallback appear in 3–4 runs, bound to specific tool interfaces. The themes differ by benchmark: answer formatting on SearchQA, header location and aggregation on the spreadsheet benchmark, evidence-verification discipline on OfficeQA (Appendix C, Table 18). By sentence form, most rules are conditionals of the kind "under condition X, do Y" (Appendix C, Table 19). That the written content is operational contracts raises the next question: whose execution needs do these contracts serve?

Whom the contracts serve: the target model's failure gaps. How much is written tracks model maturity (Table 12). The smallest model, Qwen3.6, exhausts its budget in all three runs (proposal rate 100%) and on SearchQA—whose seed spans 104 characters—accepts two revisions that grow the text more than forty-fold (4,502 and 5,617 characters). The more mature Kimi-K2.6 accepts once in three runs, its single write extending an 887-character seed to 1,675. The direction of writing is consistent: near-empty seeds are heavily expanded, while on OfficeQA, whose 887-character seed is already a complete human workflow, Qwen3.6 and DeepSeek-V3.2 exhaust their budgets with acceptance rate 0, and the accepted revision is Kimi-K2.6's small addition. Six of the ten runs changed the text. That the contracts follow the target model invites a sharper test: keep the evolution method fixed and let a different model run it—does the produced skill still fit?

A stronger evolver: lower success for Qwen2B, higher cost. The two skills share one evolution method and differ in the model that ran it (Table 13). On behalf of Qwen2B, a stronger model (GPT-5.5) produced a text more than twice as long (9,675 vs. 4,502 characters); the extra length subdivides the same answer-form topics into rarer cases—quotation style, abbreviation copying, hard letter counts. The longer text leaves Qwen2B worse off: its success and stable-success rates (47.8%, 18.4%) fall below the version Qwen2B evolved itself (50.4%, 25.6%), at 5.5$\times$ the cost per correct outcome. The deficit lies in delivery: the self-evolved text contains the rule "put the final answer in the <answer> tag, without explanations, citations, or repeating the question"—the only clause specifying how to hand in an answer—whereas the GPT text leaves delivery unspecified; the corresponding average answer lengths are 11.6 vs. 73.5 words. Constraints travel with the evolver: GPT wrote out the subdivisions it cares about, and in this cell the constraints the small model needed came from the evolution aimed at it.

Stage summary. The three tables answer one question at three levels. Evolution writes operational conventions: reading, computation, delivery, and error handling (Table 11). The conventions correspond to the target model's own failure gaps: most runs terminate early, roughly nine in ten candidate revisions are rejected, and the smallest base model receives the most writing (Table 12). When a stronger model, GPT-5.5, runs the same evolution method, the resulting skill brings no gain to Qwen2B at higher cost (Table 13). The skill text therefore records constraints written by a particular evolver for a particular execution condition: whole-document injection makes these constraints

Table 3: Proximate-cause attribution of 343 paired harm events ("same task, same conditions: no skill passes, the skill fails"). For each event we check the official evaluator signature and the execution log to determine the proximate cause of failure—a deterministic determination "→" separates no-skill from self-evolved answers; quotes are verbatim excerpts (. . . marks truncation).

Proximate cause	Events	Share [95% CI]	Verbatim pairs
Value taken from the wrong location or basis	187	54.5% [49.2, 59.7]	"What is the page number of the U.S Treasury Bulletin from July 1946 . . . ": 42 → 72. "What was the federal government's interest cost for the calendar year . . . ": 93,349 → 28,279. "The one in Central America": Belize → San Salvador
Result not materialized in evaluator-readable form	115	33.5% [28.7, 38.7]	Evaluator record: expected '0', got '=SUMPRODUCT((B2:G2=MAX(B2:G2)))'. Plus 9 undelivered output workbooks
Other observable mismatch	41	12.0% [8.9, 15.8]	—

present on every task—when the execution condition falls within a constraint's premise, intermittently correct behavior is fixed into reliable behavior, and when it falls outside, the constraint idles or alters behavior that was already correct. This is the content-side explanation for the score reversals of Section 3.1. To determine which rules cause harm on which tasks, the per-task failure events must be aligned with the contracts that fired.

3.3 Where Do Gains and Harms Arise, and Why?

Section 3.1 measured score reversals; Section 3.2 read the contracts written into the text. This subsection decomposes the net score, asking in turn: what is the nature of the gain, where does the cost land, and at which contract stage does harm appear. The instrument is five-seed paired execution: each task runs five times under the no-skill and the skill condition, and we compare per-task outcomes (execution protocol in Appendix B).

Nature of the gain: fixing occasionally-correct into always-correct. Under no skill, over half of SearchQA (54.2%) is unstable across five executions: the same task draws 3.34 distinct answers on average, and 28.7% of unstable tasks were solved only once (Table 14). Self-evolution raises *always correct* from 12.2% to 27.0%; almost all newly stable-correct tasks come from the mixed band (256 tasks), while 15 of the 497 stable failures are rescued. On OfficeQA the composition barely moves—stable failure actually rises slightly (55.4% vs. 57.8%). Across three benchmarks, stable-failure sets overlap more than stable-correct sets: every method is stuck on the same tasks, while the tasks each method solves differ. The nature of the gain is thus clear: fixing occasionally-correct tasks into always-correct; unsolvable tasks remain unsolvable.

Location of the cost: new errors on tasks the no-skill run always solves. Change splits by band (Table 15). Self-evolution raises *always correct* by 18.05 points on the hard band and 14.55 on the mixed band; on the easy band—138 tasks the no-skill run solves five out of five—it knocks 9.28 points out of correctness. The human row moves in the same direction, further (-23.33): injection itself interferes with mature execution. The net average is the residue of these offsetting directions. What kind of change this is, the composition comparison answers.

The composition comparison establishes the direction of causality (Table 16). On the easy band, value/evidence mismatch appears from nothing (0% → 9.0%): this error type records zero occurrences under no skill, so injection introduces it. The hard band is the opposite: no skill already mismatches on 92.1%, and the skill converts part of that to correct while the error composition stays within existing modes. A compatible explanation for the easy-band harm is that the injected text imposes extra constraints: tasks that were answerable directly are detoured by these constraints and land on value/evidence errors, while on tasks that were always wrong the same text acts as a corrective.

Table 4: Default off, retry on failure, against the two extremes. The rule in one sentence: by default do not use the skill; if the no-skill execution fails, retry once with the skill. The rule is fixed before scoring and operates without gold answers. SearchQA and OfficeQA are paired executions of the self-evolved skill (7,000 / 860 rollouts; CIs from task-cluster bootstrap); ALFWorld is a 160-task paired reuse audit of the seed skill (point estimates).

Cell (skill)	Default off	Always inject	**Default off, retry**	vs. default off	vs. always inject
SearchQA (self-evolved)	37.0%	50.4%	**59.1%**	+22.1 [+20.9, +23.4]	+8.7 [+7.9, +9.5]
OfficeQA (self-evolved)	13.1%	7.0%	**16.5%**	+3.4 [+2.1, +4.9]	+9.5 [+6.5, +12.8]
ALFWorld (seed skill)	31.3%	13.1%	**35.0%**	+3.8	+21.9

Where harm appears: evidence verification and result delivery. Every event in Table 3 is a paired case where no skill passed and the skill failed under identical conditions, so the reversal is attributable to the presence of the skill text; the table reports the proximate cause given directly by evaluator signatures and logs: a wrong page (42 answered as 72), a wrong basis (interest figures from a different statistical basis), a substituted neighbor entity (Belize replaced by San Salvador), formulas the evaluator cannot read, and undelivered files. Two bundles cover over 80% of the harm: wrong-value retrieval (54.5%) and non-materialized delivery (33.5%). The mix follows the benchmark's delivery contract: all 30 SpreadsheetBench events are non-materialized delivery, and all occurred on tasks the no-skill run could deliver correctly; OfficeQA is dominated by wrong-value retrieval (59.8%). Harm takes the form the benchmark demands—the same two operation categories that evolution writes most densely (Table 11).

Stage summary. Gains and harms each have a location and a mechanism. The gain is stabilization: almost all newly stable-correct tasks on SearchQA come from the mixed band (256 tasks), while 15 of 497 stable failures are rescued (Table 14). The cost is intervention: the easy band loses 9.28 points of correctness, all turning into value/evidence mismatches—an error type introduced by injection (Tables 15 and 16). Harm has a location: over 80% appears as wrong-value retrieval or non-materialized delivery, with composition following each benchmark's delivery contract (Table 3). Together with Section 3.2—contracts carry premises and are present on every task—premise-satisfying tasks turn occasional successes into reliable ones, and premise-violating tasks turn correct executions into wrong ones. The remaining question is: at which point in the information stream can a deployer tell whether an injection will help or harm?

3.4 Should the Skill Be Used at All?

The harms of Section 3.3 concentrate on tasks the no-skill run already solves: injecting the skill everywhere carries a cost. The deployer thus faces a binary decision: use the skill on this task, or hold it back. This subsection first tests whether the decision can be made before execution (Table 17), then moves the decision to the point where it can be made reliably, and measures the payoff (Table 4).

The use/no-use decision stays open before execution. The three pre-execution sources leave the decision open (Table 17). Question text is one-up one-down across benchmarks and pooled ineffective: routing by surface similarity learns dataset topics in place of a skill's applicability conditions. Structured features add insignificant positive signals. Same-task history is significant on rerun tasks ($+1.53$, $+1.44$) but requires the task to have been run before; spread over all tasks, its benefit falls to $+0.63$ ($p = 0.080$). The fourth row defers the decision to the end of execution: once pass or failure is determined, both benchmarks rise to $+9.5$ and $+8.7$. Before execution the decision remains open; once the outcome arrives, it closes. The reasonable policy therefore skips the skill by default and decides on a retry after seeing the outcome.

Default off, retry on failure. All three benchmarks agree, and the policy beats both extremes (Table 4). On OfficeQA the ordering is complete: always injecting falls below the default-off extreme (7.0% vs. 13.1%), and default-off with retry is highest (16.5%). On SearchQA it lifts the no-skill 37.0% to 59.1% while skipping 37% of skill calls. On ALFWorld the seed skill transfers severely negatively (13.1%); the policy is unaffected (35.0%). The payoff comes from the mechanisms of the previous two subsections: the policy keeps every no-skill success—including the easy tasks the skill

would break—and recovers the no-skill failures the skill can solve. As contrasts, alternative policies that decide at information-poor points—triggering on the response text, or routing by execution path—all score below always injecting (Appendix C, Table 22).

Stage summary. Whether to inject a skill remains open before execution: question text, execution features, and history carry limited information, while the pass/fail outcome of the execution itself settles it. The resulting policy—default off, retry once with the skill after a failed no-skill attempt—beats both the default-off and the always-inject extremes on all three benchmarks: it keeps every no-skill success, recovers the failures the skill can repair, and avoids the harm the skill would introduce. These results complete the analysis. What self-evolution writes is a set of operational contracts with premises; where the task's execution conditions fall within those premises, the skill converts unstable successes into reproducible outputs, and where they fall outside, the same text damages correct executions. Because the premises hold or fail within a concrete execution, the sensible deployment applies a skill only after verification fails.

4 Implications for Evaluators, Builders, and Deployers

The audit's findings translate into three recommendations, one for each audience the mismatch touches.

For evaluators. Never cite a skill score apart from the configuration it was measured in: the same skill spans $+13.4$ (SearchQA) to -6.2 (OfficeQA), $+9.9$ (Kimi-K2.6) to -6.2 (Qwen2B), and $+12.5$ versus -5.5 across harnesses alone (Tables 1–10). Alongside the mean, report the paired action surface—improvement and harm counts (315 versus 116), band-level transitions, and stable-success claims only together with their $k/5$ definition. Finally, keep a human-seed or empty control in the loop: on OfficeQA the evolved product's harm nearly equals its human seed's (-6.2 vs. -6.1), so without such a control the cost of injection is silently booked as a gain of evolution.

For method builders. The acceptance gate optimizes the very mean this audit questions: nine in ten candidate revisions are rejected (pooled acceptance rate 10.7%), and what survives is a set of operational conventions tied to the evolver and the target model (Tables 12 and 11). Skills do not automatically cross the evolver–target boundary: the checkpoint evolved by GPT-5.5 leaves the target model worse than its own self-evolution at $5.5\times$ the cost per correct outcome (Table 13). Skills should therefore be evaluated on the model they were evolved for, and the benefit of evolution should be reported as what it is—stabilization of the intermittently correct band, not new capability.

For deployers. Whole-document injection is a choice, and the choice cannot be made before execution: question text, execution features, and same-task history all fail to settle it, while one pass/fail outcome does (Table 17). The deployable default is therefore a policy, not a predictor: keep the skill off, and retry once with it after a failure. This dominates both extremes on all three benchmarks—59.1% vs. 37.0%/50.4% on SearchQA, 16.5% vs. 13.1%/7.0% on OfficeQA, and 35.0% vs. 31.3%/13.1% on ALFWorld (Table 4)—while every strategy that decides earlier loses to always injecting (Table 22).

5 Conclusion

In this work, we audit SkillOpt, a representative self-evolving skill system, under an execution-conditioned protocol. The audit's conclusion comes down to one sentence: a skill score is not a property of the skill but of its execution condition. The root cause is the contract–injection mismatch—operating rules written with preconditions are deployed into every execution, and whether a precondition holds can only reveal itself in execution. We also reported the roads that do not work: no pre-execution information suffices to decide whether to inject, and every alternative strategy we tested loses to full injection. What remains is one deployable default—do not inject by default; use the skill after a failure—and one request to evaluators, the other half of the same coin: never cite a skill score apart from the conditions under which it was obtained.

REPRODUCIBILITY STATEMENT

Upon acceptance we will release the deterministic audit scripts (one entry point per audit), the archived inputs and intermediate CSVs behind every content-level table, per-run configurations (optimizer model, rounds, edit budget), execution logs, the skill documents and human seeds, and all statistical analysis code. The supplement records the audited system, benchmarks, models, harnesses, training configuration, statistical procedures, and a table-by-table provenance map (Table 7).

AI USE STATEMENT

In this work, we used generative AI tools for [tasks with required disclosure]. We have not used generative AI tools for [other tasks with required disclosure], and [the rest of the required disclosure tasks] are not applicable to this work. Additionally, we used generative AI tools for [tasks with recommended disclosure]. We have reviewed all AI-assisted work. [Elaborate.] We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Zhiyu Chen, Zihan Guo, Bo Huang, Bingwei Lu, Jianghao Lin, Yuanjian Zhou, and Weinan Zhang. Skilljuror: Measuring how agent skill organization changes runtime behavior. *arXiv preprint arXiv:2606.11543*, 2026.

Gen Dong, Yanjie Gao, Liqun Li, Tianyin Xu, Yu Hua, and Fan Yang. Agent skills can be harmful: An empirical study of skill-induced failures in llm agents. *arXiv preprint arXiv:2608.11888*, 2026.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Xue Yang, Dongdong Chen, Xiaoqing Zheng, and Chong Luo. From raw experience to skill consumption: A systematic study of model-generated agent skills. *arXiv preprint arXiv:2605.23899*, 2026.

Zhiyuan Jiang, Fangrui Huang, Hanwen Xing, Xander Wu, Yipeng Gao, Rui Cao, Mengdi Wang, Shilong Liu, and Yijiang Li. Demystifying agent skills: Why they work-until they don't. *arXiv preprint arXiv:2608.14036*, 2026.

Christopher Kevin, Narendran Raghavan, Jean-Francois Puget, Roshni Malani, Meghana Puvvadi, Moshe Abramovitz, Mohit Gupta, Rama Akkiraju, Subodh Prabhu, Yogesh Dangi, Wei Luo, and Seong Hee Lee. Evaluating skills, not just agents: Agentic continuous evaluation of skills. *arXiv preprint arXiv:2608.20614*, 2026.

Tu Lan and Chaowei Xiao. Runtime skill audit: Targeted runtime probing for agent skill security. *arXiv preprint arXiv:2606.11671*, 2026.

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

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *arXiv preprint arXiv:2406.14991*, 2024.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, Michael Bendersky, Matei Zaharia, and Xing Chen. Officeqa pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2023.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 63897–63911, 2025.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Jiahao Ying, Boxian Ai, Wei Tang, Siyuan Liu, and Yixin Cao. Openskilleval: Automatically auditing the open skill ecosystem for llm agents. *arXiv preprint arXiv:2605.23657*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023.

A Extended Related Work

A growing toolbox lets agents accumulate skills from their own experience. Reflexion stores verbal feedback (Shinn et al., 2023), ExpeL distills lessons from training tasks (Zhao et al., 2023), AWM induces reusable workflows (Wang et al., 2025), ReasoningBank distills reasoning strategies (Ouyang et al., 2025), and Voyager accumulates executable code skills (Wang et al., 2023); SkillOpt and Trace2Skill go further and optimize a textual skill directly under validation feedback (Yang et al., 2026; Ni et al., 2026). The methods keep multiplying, but the acceptance test has hardly changed: evolve an artifact, run it on a benchmark, and check how much the average score moves. This paper audits that practice itself. We take SkillOpt as the system under audit, first checking how its evolved products are actually deployed—in every evaluation cell we observe, the product is one document injected in full into every execution (Section 2)—and then comparing paired outcomes with and without the skill, under a fixed task pool, executor, and protocol. We do not propose another evolution method; we report what an audit of one finds: what evolution changed relative to its human seed (Section 3.2), where the stabilization it buys pays off and where it costs (Sections 3.1 and 3.3), and how much of these patterns can be exploited when only partial information is available (Section 3.4).

Closest to us, one line of work audits skills themselves. Demystifying Agent Skills runs controlled experiments across benchmarks, harnesses, and LLMs and codes 8,135 paired trials, finding that a given skill mostly stabilizes existing behavior rather than adding missing facts, with retrieval as an independent bottleneck (Jiang et al., 2026). Along the same direction, Agent Skills Can Be Harmful traces task failures back to specific skills (Dong et al., 2026); OpenSkillEval synthesizes tasks from the open skill ecosystem and reports that, in aggregate, having a skill is not the same as using it (Ying et al., 2026); SkillJuror measures how skill organization changes runtime behavior (Chen et al., 2026); ACES builds repository-native infrastructure for continuous skill-level evaluation (Kevin et al., 2026); a systematic study of model-generated skills finds that how plausibly a skill is written does not predict whether it helps (Huang et al., 2026); and Runtime Skill Audit probes skill security at runtime, orthogonal to our outcome-and-content ledger (Lan & Xiao, 2026). What this line shares is the object of study: *given* skills—authored by humans or imported from elsewhere—measured over *single* executions. Our object is the evolution process itself: the unit of analysis is what one evolution run changed relative to its seed, and the instrument is paired outcomes over repeated executions of the same tasks, not coded single runs. And the ledger has two sides, so we record both: what the skill brings in, and what it gives back—regressions on easy tasks, and rescue/harm that flips with seed and harness (Sections 3.1 and 3.3).

B Experimental Setup and Reproducibility Details

Audited system. The audited system is SkillOpt, a skill self-evolution framework. Each training round executes the current skill document on training tasks, reflects on the executed trajectories to produce patch candidates (patches are drawn from both successful and failed trajectories), merges the patches into one complete candidate document, and submits it to a validation gate: the gate evaluates the candidate on held-out validation tasks and either accepts it (rewriting the current document) or rejects it (keeping the previous text). Updates are patch-based, bounded by an edit budget of four items per round, and the gate is always enabled. At test time the final document is injected as a whole into every task: we verified single-hash whole-document injection in 105/105 examined cells, so no per-task or per-rule selective loading exists.

Table 5: Skill sources compared throughout the paper. Sizes are the injected text lengths (characters) on the named benchmark.

Source	Definition	Size (SearchQA)
No skill	Empty file	38
Human seed	Human-written seed document, one per benchmark	104
SkillCreator	Skill produced by the SkillCreator system	2,877
Official checkpoint	Released checkpoint skill, evolved by GPT-5.5 with the same method	9,675
Self-evolved	Produced by the target model itself (SkillOpt-self)	4,502

Benchmarks. SearchQA (Dunn et al., 2017): short-answer QA, scored by exact match; single-turn execution (`max_turns`=1). OfficeQA (Opsahl-Ong et al., 2026): QA over office documents, scored by exact match. SpreadsheetBench (Ma et al., 2024): workbook manipulation, scored by reading stored cell values with openpyxl—live formulas do not count as materialized results. ALFWorld (Shridhar et al., 2021): embodied household tasks, scored by task completion. SkillsBench (Li et al., 2026): general skill-execution tasks (used only in the writing audit). Task pools differ across analyses because each analysis uses the tasks for which every condition it compares has complete logged results; the pool and its size are stated in each table's caption, and Table 7 maps every table to its pool.

Table 6: Base models, harnesses, and execution. All models are referred to by public names; inference used hosted endpoints with versions frozen as of 2026-07-23.

Base models	Qwen2B, Qwen3.6, DeepSeek-V3.2, Kimi-K2.6; skill evolver for the official checkpoint: GPT-5.5
Harnesses	Direct chat (skill appended to the system prompt); Claude Code and Codex CLI (skill written as a workspace `SKILL.md` with each harness's own tools, retries, and repair loop)
Execution	The same task ID runs under both conditions with identical seed labels; outcomes are hard verdicts from the official evaluator
Repetitions	Five seeds (1–5) per condition unless stated otherwise

Evolution runs. Configured training rounds are 40 (SearchQA), 8 (OfficeQA), and 4 (SpreadsheetBench, ALFWorld); runs may terminate before the configured budget, and proposal rates in Table 12 use the configured rounds as denominator. Training uses seed 42 and the per-benchmark train split (e.g., 400 SearchQA training tasks). Ten formal runs are analyzed (Table 12); two additional runs (Qwen3.6×SkillsBench and a second Qwen3.6×SpreadsheetBench run) exist in the logs and are excluded from that table's scope.

Statistics. Paired single-seed comparisons use the exact McNemar test; pooled paired deltas use a task-cluster bootstrap with 5,000 resamples (cluster = task), with Holm correction inside predeclared families (benchmark columns in Table 1; band×source cells in Table 15). Proportions carry Wilson 95% CIs; the class-vs-class comparison in Table 19 uses an exact binomial test; set overlap is Jaccard ($|\cap|/|\cup|$) and cell-level association is Spearman rank correlation; selector CIs

in Tables 17–4 are seed-level or task-cluster bootstraps as marked. Significance marks: $^{*}p \leq 0.05$, $^{**}p \leq 0.01$, $^{***}p \leq 0.001$.

Table 7: Table-by-table provenance: the pool, compared sources, executor, and statistic behind every table in the paper. "Direct" is the direct-chat harness.

Table	Pool (tasks)	Sources compared	Executor / harness	Statistic
Table 1	SearchQA 1,400; OfficeQA 172	No skill vs. self-evolved (s1–s5); human	Qwen2B / direct	McNemar; cluster bootstrap; Holm
Table 2	OfficeQA 172	No skill vs. four systems	Each model / direct	McNemar; cluster bootstrap
Table 10	Per-row benchmark pools	Qwen3.6 system across harnesses	Qwen3.6 / direct, Claude, Codex	Sign test; McNemar
Table 11	14 final documents	Learned-only rules (text audit)	—	Deterministic classification
Table 12	10 formal runs	Evolution logs	—	Rates; Wilson CI
Table 13	SearchQA 1,400×5	Five skill sources	Qwen2B / direct	Descriptive; text profile
Table 14	SearchQA 1,480; OfficeQA 374	No skill vs. self-evolved	Qwen2B / direct	Composition; transitions; Jaccard
Table 15	SearchQA 1,400 (bands)	No skill vs. self-evolved vs. human	Qwen2B / direct	Δ with CI; Holm
Table 16	Same as Table 15	Same	Same	Shares
Table 3	343 paired harm events	Cross-benchmark cells	Official evaluator + logs	Bootstrap CI
Table 17	Held-out splits per benchmark	Four information sources vs. always inject	Qwen2B	McNemar; bootstrap
Table 4	SearchQA 7,000 rollouts; OfficeQA 860; ALFWorld 160 pairs	Default off / always inject / default-off retry	Qwen2B	Cluster bootstrap
Table 20	SearchQA 1,400×5	Threshold sweep k=3, 4, 5	Qwen2B / direct	Replacement rate
Table 21	Strict common coverage: 7,600 / 2,402 / 434 rollouts	No skill + three skill sources	Qwen2B	Oracle; stability
Table 22	OfficeQA Qwen2B pool	Rejected policies vs. always inject	Qwen2B	Accuracy

Deterministic audits. Every content-level table in the paper is produced by a released, deterministic script: no model calls, no LLM judges, and no manual labeling enters any reported number. Each script reads only archived artifacts (skill documents, per-run configurations, and execution logs), writes its intermediate tables to archived CSVs, and is released with a single entry point per audit. Table 8 lists each audit, its input, its procedure, and the tables it serves.

Table 8: Deterministic audits behind the paper's tables. All scripts are released; all inputs and outputs are archived CSVs and documents.

Audit	Input	Procedure	Serves
Rule extraction & categorization	Final skill documents + human seed documents	Split into rule lines; normalize (case, punctuation); *learned-only* = present in the final document and absent from the seed; one category per rule via ordered regular-expression patterns, with a designated fallback class	Table 11; Tables 20 and 21
Text composition profile	Two injected skill texts	Segment into rule lines (drop headings and scaffolding); count marker lexicons (broad advice, hard constraints, delivery contract, task/artifact); normalize per 1k chars	Table 13 (bottom)
Five-seed stability & bands	Per-task×seed outcomes	Outcome strings → always-correct / always-wrong / mixed; bands by no-skill correct count; terminal states mapped from evaluator outputs	Tables 14–16; Table 20
Harm attribution	Paired rollouts (evaluator signatures + logs)	Pair by task and seed; assign proximate cause only when the signature or a literal log cue supports it; otherwise a reserved residual class	Table 3
Selector & policy validation	Logged rollouts	Task-level train/test splits; feature extraction from no-skill traces; fixed policies scored post hoc	Tables 17–4; Table 22
Oracle	Cross-source outcomes	Per-task maximum over skill sources	Table 21

Two procedures carry interpretive weight and are specified in full. *Rule categorization* (Table 11) assigns each learned-only rule to exactly one of nine categories through an ordered list of released regular-expression patterns, applied in fixed priority order; rules matching no pattern fall into a reserved ninth class. The patterns, the raw extracted rules, and the per-rule assignments are all released, so any alternative grouping can be recomputed from the same 189 items. *Harm attribution* (Table 3) reports the *proximate cause* of each paired failure and nothing more: a cause is assigned only when the official evaluator signature or a literal execution-log cue supports it, events without such support fall into the reserved residual class (12.0%), and log cues are treated as evidence of where the failure appears, not as rule-level causation. Whole-document injection (single-hash verification, Section 3.2) rules out selective loading, but it also means no per-rule treatment exists; we therefore never attribute a failure to an individual rule.

Archive and scope. Per-run configuration files (including optimizer model, rounds, and edit budget) and all execution logs are archived and released with the code. Older runs do not carry complete immutable serving-side manifests (e.g., decoding temperature and parser limits); all comparisons in this paper are therefore paired reuse audits of logged outputs under identical task and seed labels, not newly fingerprinted rollouts. Hosted-inference version drift is a residual risk. Code, logs, and per-run configurations will be released upon publication.

C SUPPLEMENTARY TABLES

This section collects the full-length tables relocated from the main text for space, in paper order and grouped by the section they support, followed by the additional registered analyses. Every table keeps its original caption, pool, and statistic; Table 7 maps each table to its exact pool and protocol.

C.1 TABLES FOR THE METHOD (SECTION 2)

Table 9: The five skill conditions. All conditions share the execution interface of Eq. (1) and the evaluation protocol; each (model, benchmark) pair is trained once, direct-chat and both agentic-harness evaluations reuse the same $S^\star$, and same-cell evaluations are repeated over multiple evaluation seeds.

Skill condition	Document source S	Role in the audit
noskill	$S = \varnothing$ (empty document)	no-skill baseline
human	human skill S_{h}	human-written baseline
skillcreator	one-shot generated, frozen external file	one-shot control
skillopt_ckpt	official SkillOpt release (GPT-5.5 optimizer)	external artifact
skillopt_self	$S^\star$ self-trained in the same codebase (fixed training seed)	primary audit object

C.2 TABLES FOR SCORE TRANSFERABILITY (SECTION 3.1)

Table 10: The deployment axis. The base model is fixed to Qwen3.6; rows compare the same self-evolved skill system across deployment conditions (direct chat = no harness, Claude Code, Codex CLI) against the no-skill condition, per-task paired. Per-seed Δ lists paired differences under 5 training seeds; pooled Δ reports a task-cluster bootstrap 95% CI for the direct-chat row and seed means for harness rows; p is an exact sign test (direct row) or per-seed exact McNemar (harness rows; $^{*}p \le 0.05$, $^{**}p \le 0.01$). † single-seed cells with incomplete task ID sets, reported as directional only.

Benchmark	Deployment	Per-seed Δ	Pooled Δ
OfficeQA	Direct (no harness)	-2.3 / $+2.3$ / $+5.2$ / $+2.9$ / -4.7	$+0.7$ [-2.4, $+3.8$]
OfficeQA	Claude Code	-16.7† / $+25$ / $+30$ / -15 / $+15$	$+7.7$ (seed mean)
OfficeQA	Codex CLI	-3.3† / $+5$ / -10 / -20 / -15	-8.7 (seed mean)
SearchQA	Claude Code	$+15$ / $+25^{**}$ / $+10$ / $+12.5$ / 0	$+12.5$ [$+5.0$, $+20.5$]
SearchQA	Codex CLI	$+2.5$ / -12.5 / -7.5 / -5 / -5	-5.5 [-11.5, -0.5]
SpreadsheetBench	Claude Code	-5 / $+15$ / -10 / -15 / -20	-7.0 (seed mean)
SpreadsheetBench	Codex CLI	$+5$ / -5 / $+10$ / -5 / $+30^{*}$	$+7.0$ (seed mean)

C.3 TABLES FOR WHAT EVOLUTION WRITES (SECTION 3.2)

Table 11: Categories of model-added rules. Rules are extracted from the final skill documents of self-evolution training: 189 model-added rules, i.e., sentences present in the final document but absent from the corresponding human seed text, pooled over the 14 documents that contain any addition. Each rule is assigned to one category; "training runs" counts runs containing at least one rule of the category (out of 14). Examples are excerpts of original rules. Extraction and categorization are deterministic scripts; see Appendix B.

Category	Rules	Runs	Example (excerpt)
Output format & canonicalization	43	11	"When the question asks for a **type** or **category** . . . , provide the common noun without extra modifiers."
Evidence scope & verification	35	12	"After a promising match, read only a small surrounding span and verify it matches the requested year, basis, and unit."
Other specific / low coverage	32	8	". . . the answer is the noun phrase that completes the reference."
Safe execution & artifact	22	8	"Excel error values like #N/A appear as strings starting with # . . . "
Computation & aggregation	18	10	"When the question names a specific formula (e.g., arc elasticity, CAGR), compute exactly the expression described."
Retrieval & search	18	4	"When parsed pages are provided in the prompt . . . , extract values from them directly."
Dynamic navigation & schema	12	4	"Build a lookup dictionary from the source sheet using a composite key (e.g., (date, name, id))."
Error handling & fallback	5	4	". . . fill remaining target cells with `None`."
Priority & ordering	4	3	"First collect all target row numbers."

Table 12: The writing funnel over ten formal self-evolution runs (three base models × four benchmarks). Each run is configured with a number of training rounds and may terminate before exhausting the budget; each round proposes one complete candidate document, which a fixed validation gate evaluates—acceptance rewrites the current text, rejection keeps it. Proposal rate = proposed rounds / configured rounds; acceptance rate = accepted versions / proposed rounds; the pooled acceptance rate has Wilson 95% CI [5.7%, 19.1%].

Run	Proposal rate	Acceptance rate	Seed → final (chars)
Qwen3.6 × SearchQA	100%	5.0%	104 → 4,502
DeepSeek-V3.2 × SearchQA	32.5%	23.1%	104 → 5,617
Kimi-K2.6 × SearchQA	2.5%	0%	104 → 104
Qwen3.6 × OfficeQA	100%	0%	887 → 887
DeepSeek-V3.2 × OfficeQA	100%	0%	887 → 887
Kimi-K2.6 × OfficeQA	25.0%	50.0%	887 → 1,675
Qwen3.6 × SpreadsheetBench	100%	25.0%	1,594 → 6,451
DeepSeek-V3.2 × SpreadsheetBench	75.0%	33.3%	1,594 → 6,091
Kimi-K2.6 × SpreadsheetBench	100%	0%	1,594 → 1,594
DeepSeek-V3.2 × ALFWorld	25.0%	100%	2,966 → 5,855
Total	**52.5%**	**10.7%**	—

Table 13: The same evolution method, two evolvers: execution outcomes (top) and text composition (bottom). Top: same-pool comparison on Qwen2B×SearchQA, 7,000 executions per source (1,400 tasks × 5 repetitions). "5/5 stable success" is the fraction of tasks answered correctly in all 5 repetitions; "cost per correct" is the average response characters spent per correct outcome. The official checkpoint skill and the self-evolved skill come from the same evolution method: the former was produced by running evolution with a stronger model (GPT-5.5), the latter by the target model itself under identical settings. Bottom: a deterministic composition profile of the two injected texts (this cell only); "rule lines" are clauses excluding headings and scaffolding, and marker counts follow the registered text audit.

Source	Chars	Success	5/5 stable	Avg. answer words	Cost per correct
No skill	38	37.0%	9.9%	89.1	1,541
Human seed	104	35.1%	8.2%	84.9	1,541
SkillCreator	2,877	49.7%	25.9%	9.6	144
Checkpoint skill (GPT-evolved)	9,675	47.8%	18.4%	73.5	984
Self-evolved (target model)	4,502	**50.4%**	**25.6%**	11.6	**179**

Composition	Checkpoint (GPT)	Self-evolved
Length (chars)	9,675	4,502
Rule lines	38	22
Characters per rule line	≈255	≈205
Rule lines / 1k chars	3.9	4.9
Broad-advice markers per rule	2.11	1.86
Delivery-contract markers / 1k chars	8.4	10.2

C.4 Tables for Where Gains and Harms Arise (Section 3.3)

Table 14: Five-seed stability. Qwen2B execution, each task repeated 5 times; *always correct* / *always wrong* / *mixed* are defined by the 5 outcomes. Panel A: task composition (SearchQA pool 1,480 and OfficeQA pool 374, the stability audit's own five-seed common task sets). Panel B: SearchQA transitions from no-skill to self-evolution, and the answer churn of no-skill's unstable tasks. Panel C: Jaccard overlap (intersection/union) between no-skill and self-evolution stable sets on each benchmark's common tasks.

Panel A: task composition (%)

Benchmark	Source	Always correct	Always wrong	Mixed
SearchQA	No skill	12.2	33.6	54.2
SearchQA	Self-evolved	27.0	23.7	49.3
OfficeQA	No skill	17.4	55.4	27.3
OfficeQA	Self-evolved	16.0	57.8	26.2

Panel B: SearchQA, gains and where they come from

Quantity	Value
Distinct answers drawn by no-skill on unstable tasks (of 5)	3.34
Unstable tasks solved only once (1/5), no skill	28.7%
Mixed → always correct (self-evolved)	256 tasks
Always wrong → always correct (of 497 stable failures)	15 tasks
Always correct → always wrong (self-evolved)	1 task

Panel C: Jaccard overlap, no skill vs. self-evolved

Benchmark	Stable correct	Stable wrong
SearchQA	0.285	0.488
OfficeQA	0.645	0.839
SpreadsheetBench	0.333	0.667

Table 15: Terminal-state changes by difficulty band. Qwen2B×SearchQA, the same 1,400-task pool, banded by no-skill performance over 5 repetitions: easy band (5/5 correct, 138 tasks), mixed band (1–4/5, 770), hard band (0/5, 492). Δ is the change in terminal-state composition relative to no skill (percentage points), 95% CI; for the correct row, p is Holm-corrected within band and all $p < 0.001$. The human row is the human-seeded skill as control.

Band	Source	Always correct Δ	Value/evidence mismatch Δ
Hard	Self-evolved	$+18.05$ [$+15.8$, $+20.4$]	-11.06
Hard	Human	$+6.18$ [$+5.0$, $+7.4$]	-4.76
Mixed	Self-evolved	$+14.55$ [$+12.3$, $+16.8$]	-11.61
Mixed	Human	-3.25 [-5.2, -1.3]	$+3.22$
Easy	Self-evolved	$\mathbf{-9.28}$ [-12.3, -6.5]	$\mathbf{+8.99}$ [$+6.2$, $+12.2$]
Easy	Human	-23.33 [-27.4, -19.7]	$+22.03$

Table 16: Terminal-state composition by band, same experiment and bands as Table 15. "→" separates no-skill (left) from self-evolved (right) shares.

Terminal state	Easy band	Mixed band	Hard band
Always correct	100% → 90.7%	49.3% → 63.9%	0% → 18.0%
Value/evidence mismatch	**0% → 9.0%**	47.3% → 35.7%	92.1% → 81.0%
Format/exact-match	0% → 0.3%	3.0% → 0.4%	7.4% → 0.9%

C.5 Tables for the Deployment Policy (Section 3.4)

Table 17: Delegating the use/no-use decision to different information sources. Numbers are the resulting performance relative to always injecting the skill. The first two sources exist before execution: question text and structured execution features (train/test split by task; p from the exact McNemar test). The third is the same task's own history: seeds 1–3 predict seeds 4–5, evaluated only on tasks the no-skill run fails, with the all-task result in parentheses. The fourth source is the pass/fail outcome of the current execution—retrying with the skill only after a failure; CI from task-cluster bootstrap. All rules operate without gold answers.

Information source	OfficeQA	SearchQA	Verdict
Question text	$+0.99$pp	-1.27pp	ineffective
Structured execution features	$+2.31$pp ($p = 0.075$)	$+0.35$pp	not significant
Same-task history	$+1.53$pp	$+1.44$pp ($p = 0.049$); all-task $+0.63$pp ($p = 0.080$)	restricted to seen tasks
Pass/fail of the current execution	$\mathbf{+9.5}$ **[**$+6.5$**,** $+12.8$**]**	$\mathbf{+8.7}$ **[**$+7.9$**,** $+9.5$**]**	**effective**

C.6 Additional Registered Analyses

Table 18: What evolution writes, by benchmark (overview). The 14 final documents containing additions are grouped by benchmark; "docs with additions" counts documents with at least one model-added rule. Themes are summaries of the registered audit.

Benchmark	Human seed	Docs with additions	What evolution mainly writes
SearchQA	104 chars, near empty	2	Answer delivery format: common nouns over proper names, exact phrases, singular/plural, articles
OfficeQA	887 chars, mature workflow	3	Evidence discipline: read near the hit, verify year/basis/unit, use formulas by name
SpreadsheetBench	1,594 chars	6	Spreadsheet operation chain: locate headers, dictionary aggregation, recognize #N/A, preserve state
ALFWorld	2,966 chars	1	Action protocol: search order, container states, avoiding loops
SkillsBench	1,130 chars	1	No dominant theme

Table 19: Sentence forms of rules that appear in only one training run. 166 of the 189 model-added rules occur in a single run; shares are Wilson 95% CIs; p is the exact test of the top class against the second.

Form	Rules	Share [95% CI]	Example
Conditional / fallback	83	50.0% [42.5, 57.5]	If the target range already contains sample rows, infer the format from them first; otherwise use the default
Domain / harness special case	25	15.1% [10.4, 21.3]	In ALFWorld, use the `look` command to see a room
Single-run experience	24	14.5% [9.9, 20.6]	After a hit, re-check the year and unit
Likely redundant	17	10.2% [6.5, 15.8]	"move X to Y is the standard placement action" (already in the seed)
Safety / artifact guard	17	10.2% [6.5, 15.8]	On code failure, remove duplicate and unused variables before patching

Table 20: Threshold sensitivity of "stable success." Qwen2B×SearchQA, the same 1,400-task pool. "Stable success" is defined as answering correctly in k of 5 repetitions; k is the analyst's choice. The table reports the stable-task share of three sources at $k = 3, 4, 5$, and the replacement rate between the self-evolved and SkillCreator stable sets ($1-$ intersection/union).

Definition	No skill	Self-evolved	SkillCreator	Replacement
3/5	37.0%	50.4%	50.6%	27.4%
4/5	24.5%	39.1%	40.9%	34.7%
5/5	9.9%	25.6%	25.9%	55.6%

Stricter thresholds shrink every stable set and increase the share of tasks on which the two sources disagree (27.4% to 55.6%); re-pairing the five seeds into subsets raises the replacement rate monotonically as well (40.0% to 55.6%). Which tasks count as "stable successes" therefore depends on the definition itself, and any reported stable-success rate must carry its $k/5$ definition.

Table 21: The oracle upper bound and its two discounts. Panel A: on tasks where the no-skill condition and all three skill sources (self-evolved, official checkpoint, SkillCreator) have complete results, the oracle takes the correct answer per task post hoc—an undeployable upper bound. Panel B: the 2,372 no-skill failures (seeds 4–5) decomposed into capability limits and random luck; the winner-stability row asks whether the oracle's best skill on a task is the same across two test seeds.

Panel A: the value of always choosing the right skill				
Benchmark	No skill	Best fixed skill	Oracle	Selection space
SearchQA	39.99%	52.59%	70.24%	+17.64
OfficeQA	32.89%	31.43%	43.34%	+11.91
SpreadsheetBench	23.50%	23.96%	32.26%	+8.29
Total	37.67%	46.47%	62.47%	+15.99
Panel B: what discounts the bound				
Discount				Value
Capability limit: executions every skill fails				52.57%
Random luck: same-task best skill stable across two test seeds				only 54.31%

Selection space exists: pooled over the three benchmarks, always choosing the right skill would raise 46.47% to 62.47%, with per-benchmark gaps of 8.3–17.6 points; on OfficeQA all three skills fall below no skill (31.43% vs. 32.89%), yet the oracle is still 11.91 points above the best fixed skill—they harm different tasks. The bound itself is discounted twice: over half (52.57%) of no-skill failures are failed by every skill, a capability limit no selector can repair; and the oracle counts successes that are merely lucky, with the per-task best skill changing across test seeds 45.7% of the time. The space that can be learned stably is far smaller than 62.47%.

Table 22: Alternative policies that failed validation. All are evaluated on the same task pool; the reference is always injecting the skill in that cell (50.41%).

Alternative policy	Score	Verdict
Trigger injection from the model's response text	47.11%	rejected
Route operation regimes by execution path	49.99%	rejected

Both policies try to decide at information-poor points, and both lose to always injecting. Together with the question-text selector of Table 17, the failure pattern is consistent: each of these decisions happens where sufficient information is still unavailable. Default-off retry works precisely because it defers the decision to the point where information is richest.