SkillMe: Evolving Agent Skills for Recurring Personal Workflows

Anonymous ACL submission

Abstract

Skills help LLM agents reuse procedural knowledge in recurring workflows, but broad transferability alone does not guarantee workflow fit: a reusable skill may be semantically relevant yet fail under local tools, files, formats, and failure modes. We propose SkillMe[1], a workflow-situated skill evolution framework that adapts agent skills through repeated use in matched workflow recurrences. A Publisher induces and validates an initial skill from a successful execution, while Executors use the skill in agent-native sessions and provide trace-grounded feedback. SkillMe activates the matched skill on demand as workspace files, allowing agents to inspect and adopt guidance progressively without permanently installing it as a local skill. On the 71-task skill-needed subset of PinchBench with OpenClaw, SkillMe improves the overall mean score of all eight Executors over their paired no-skill baselines, with mean gains of +25.7 points. Comparisons show that evolved task-scoped skills are more stable than static skills and more reliable than broadly reusable skills. SkillMe thus provides an agent-native path to evolve reusable skills into effective workflow guidance through situated use.

1 Introduction

LLM agents are increasingly used to automate recurring real-world workflows, from personal assistance and software maintenance to complex tool use (Yao et al., 2023; Shinn et al., 2023; Wang et al., 2024; Schick et al., 2023; Patil et al., 2024; Qin et al., 2024). A central mechanism for reusing agent experience is *skills*: procedural guidance that helps an agent execute a class of tasks. Recent work constructs, refines, and evolves such skills from human design or agent trajectories, aiming to make them more generic, reusable, and shareable across tasks, agents, and users (Wang et al., 2024; Anthropic, 2026; Wang et al., 2026; Ni et al., 2026; Zhang et al., 2026; Ma et al., 2026). We refer to this common objective as *broad-reuse-oriented skill design*: optimizing skills to transfer across tasks, agents, users, and environments.

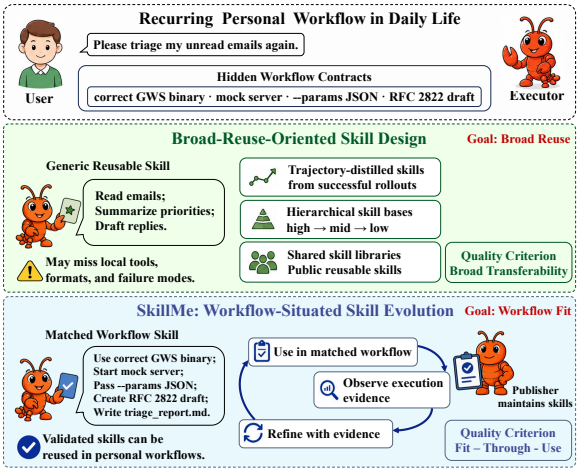

Figure 1: SkillMe complements broad-reuse-oriented skill design by evolving task-scoped skills toward recurring workflow fit through execution evidence.

Broad reuse is valuable, but not always sufficient for recurring personal workflows. Prior work shows that skill utility is highly contextual: gains vary across model–harness configurations (Li et al., 2026), and mismatched skills can degrade performance when their guidance conflicts with the execution context (Han et al., 2026). These findings suggest that skill utility depends not only on transferability, but also on situated fit. A personal workflow agent repeatedly operates with particular tools, files, conventions, and recurring failure modes. This motivates a complementary objective, *workflow-situated skill evolution*: turning reusable guidance into executable guidance for the concrete conditions of a matched

[1]Code is available at https://anonymous.4open.science/r/SkillMe/.

workflow recurrence. In other words, the question is not only whether a skill transfers broadly, but ***"Is this skill really for me?"***

Figure 1 illustrates why broad transferability alone does not guarantee situated fit in recurring personal workflows. Broadly reusable skills remain valuable as general capabilities, but workflow-specific guidance must also be validated and refined under the concrete conditions in which it is repeatedly used. Since such guidance is tightly coupled with execution conditions, it should remain task-scoped: activated only for corresponding recurrences rather than permanently installed as a generic skill. The key challenge is to turn skill use itself into evidence for adaptation while preserving lightweight, agent-native execution.

To address this challenge, we propose **SkillMe**, a workflow-situated skill evolution framework. SkillMe does not replace native skill libraries; it adds a task-scoped evolution layer for workflow-specific guidance. A *Publisher* executes a workflow, induces and validates an initial *skill*, and maintains it. *Executors* use the skill in matched workflow recurrences and return categorized feedback grounded in execution traces. The Publisher attributes this feedback: reusable skill–workflow conflicts trigger revision, while misuse or misunderstanding during execution triggers clarification. Through this loop, SkillMe turns actual skill use into evidence for adapting a skill to the workflow and agent that repeatedly rely on it.

The lifecycle is *agent-native*: induction, validation, execution, feedback, and revision are all implemented as agent sessions over concrete workspaces, rather than detached single-turn LLM calls. When a recurring workflow is matched, the corresponding task-scoped skill becomes available as a workflow artifact, instead of being permanently installed as a generic skill or forcing the full content into every prompt. The agent can then decide during execution whether to read the skill, which parts to inspect, and how much guidance to adopt. This on-demand, progressive use preserves agent-native execution while keeping workflow-specific guidance explicit, inspectable, and scoped to the current recurrence.

We evaluate SkillMe with OpenClaw (OpenClaw, 2026b) on PinchBench (PinchBench, 2026). Using one Publisher and eight heterogeneous Executors, SkillMe improves the overall mean score of all eight Executors over their paired no-skill baselines on the 71-task skill-needed subset, with mean gains of $+25.7$ points. Controlled comparisons further show that evolved task-scoped skills reduce recurrence-to-recurrence volatility relative to static skills, and are more reliable than broadly reusable public skills on matched workflow recurrences. A cross-Executor feedback-injection study shows that feedback contains situated-use signals and must be attributed before updating a shared skill. Across the evaluation lifecycle, Executors produce 3,064 structured feedback comments and SkillMe evolves 41 skills. When Publisher induction is amortized over 30 recurrences, SkillMe reaches 88.6 mean score at 45.5% of the Publisher cost. Together, these results show that SkillMe turns repeated skill use into evidence for adapting reusable skills into effective, cost-conscious workflow guidance.

2 Related Work

Agent Experience Reuse. LLM agents extend language models with reasoning, tool use, memory, and feedback for interactive tasks (Yao et al., 2023; Schick et al., 2023; Patil et al., 2024; Qin et al., 2024). Beyond single-run execution, agents can reuse experience through reflection, long-term memory, and accumulated procedures (Shinn et al., 2023; Zhao et al., 2024; Park et al., 2023; Packer et al., 2023). A key example is Voyager, which continually builds an executable skill library from open-ended interaction (Wang et al., 2024). Accepting the premise that execution traces embed procedural knowledge for future behavior, SkillMe studies what happens after such knowledge is packaged as a skill: how it should be activated, evaluated, and revised when repeatedly used in recurring personal workflows.

Agent Skill Construction. Skills externalize procedural knowledge into reusable artifacts for agent execution. Practical systems package skills as installable capabilities or agent-accessible libraries (Anthropic, 2026; Agent Skills, 2026; OpenClaw, 2026a; VoltAgent, 2026). Benchmarks such as SkillsBench and SWE-Skills-Bench show that skills can improve agent performance, but their effects vary across tasks, models, and environments; mismatched skills may cause regressions (Li et al., 2026; Han et al., 2026). Recent systems therefore automate skill construction and refinement. SkillX constructs a plug-and-play hierarchical skill base from strong-agent rollouts (Wang et al., 2026). Trace2Skill extracts transferable

lessons from trajectories (Ni et al., 2026), and Co-EvoSkills refines skills with co-evolutionary verification (Zhang et al., 2026). SkillClaw evolves shared skills from multi-user interaction traces in an agent ecosystem (Ma et al., 2026). These systems improve how skills are constructed, generalized, or verified. SkillMe addresses the complementary post-deployment question of how a task-scoped skill should be activated, evaluated, and revised through actual Executor traces in matched personal workflows.

Personalized LLM Agents. Personalization has been studied through user modeling, long-term memory, retrieval, generation, and web-agent evaluation (Tan et al., 2025; Westhäußer et al., 2025; Zhang et al., 2025; Salemi et al., 2024; Xu et al., 2026). These approaches adapt what a model remembers, retrieves, or generates for a user. SkillMe targets the skill itself as the object of personalization. Rather than only conditioning model responses on user information, SkillMe uses repeated workflow execution and feedback to maintain a skill so that it fits the concrete conditions of the workflows where it is activated.

3 The SkillMe Framework

SkillMe is a *workflow-situated skill evolution* framework (Figure 2) that induces and evolves task-scoped skills for recurring personal workflows. Unlike generic skill construction methods that aim to produce broadly reusable guidance, SkillMe enables a skill to become more effective through repeated use within a routine workflow recurrence. Its central premise is that skill use provides direct evidence for revision. An agent's execution trace provides evidence about whether the guidance is readable, actionable, and effective in the tools, files, conventions, and failure modes of the personal workflow.

The design of SkillMe follows three principles. First, skills should encode reusable workflow guidance rather than task-agnostic procedures. Second, skill quality should be judged by situated use in the workflow where the skill is activated, not only by how broadly the skill can transfer. Third, workflow-specific skills should be activated as *session-local workspace files*, rather than installed into the agent's persistent skill directory or inserted wholesale into the prompt. When a matched workflow recurrence is detected, SkillMe places the relevant skill in the current workspace, lets the agent inspect and adopt it progressively during execution, and removes it after the session.

3.1 Problem Setup: Routine Workflow, Cost, and SkillMe Roles

SkillMe starts from a user-centric setting. A user repeatedly delegates a routine workflow to a personal agent, such as triaging emails, organizing files, preparing reports, scheduling meetings, or maintaining code. While each recurrence has a new task instruction and workspace state, execution often depends on stable conditions, including tool locations, command syntax, environment assumptions, and verification routines. A single recurrence comprises a task instruction, a concrete workspace, and an evaluation criterion, yielding an execution trajectory that logs prompts, actions, observations, and scores. SkillMe treats this trajectory as its basic evidence unit to reveal how guidance is instantiated and where execution or assumptions fail across recurrences.

SkillMe separates two roles in this personal skill evolution lifecycle, exploiting a practical cost asymmetry: a high-capability *Publisher* first executes a workflow recurrence, induces workflow guidance from a successful trajectory, validates the guidance against the observed environment, and maintains the resulting skill over time. *Executors* are the agents that repeatedly use the skill in later matched recurrences and return typed feedback grounded in their trajectories. Executors may correspond to different users' personal workflow agents or different model–harness configurations. Their runs expose whether the same task-scoped skill remains so under different situated execution conditions, actively driving the Publisher's maintenance loop.

We denote a SkillMe-managed skill as ϕ. It consists of the workflow guidance exposed to Executors and platform-side metadata used for matching, validation, feedback attribution, and revision, including the skill body G, situated execution conditions C, feedback memory M, an evidence map E linking claims to transcript sources, validation checks V, and usage and revision history U. The full schema is provided in Section B.1.

The goal is therefore neither to build a permanently installed generic skill nor to personalize outputs by modeling user style. Instead, SkillMe maintains a task-scoped skill for a user's routine workflow, revising guidance when evidence reveals skill–workflow conflicts and recording clar-

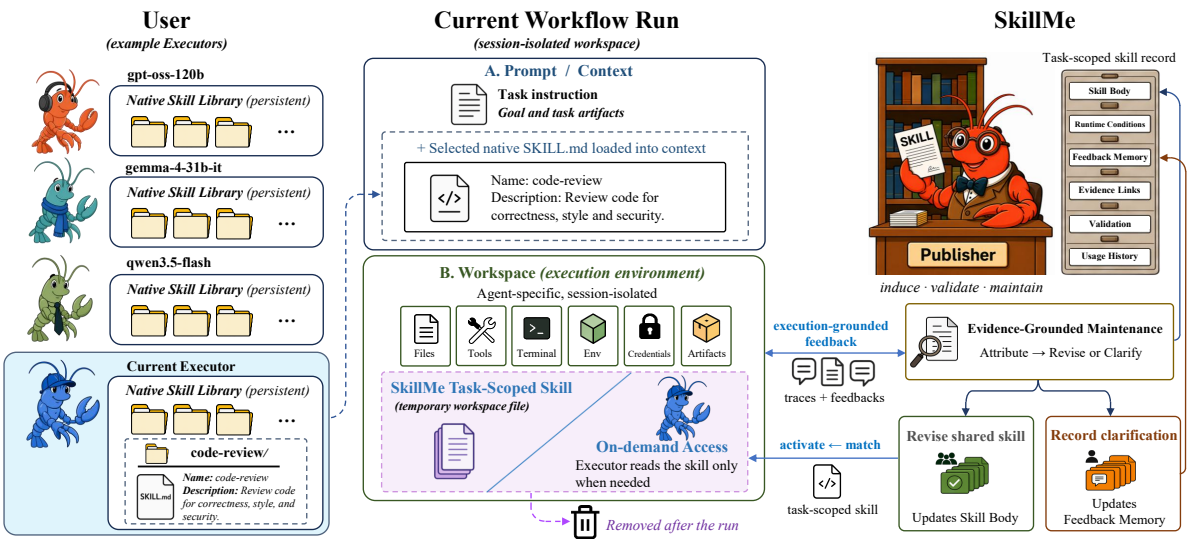

Figure 2: Overview of SkillMe. The user-side agent keeps the native skill library, while activating matched SkillMe skills as temporary workspace files. SkillMe maintains task-scoped skills through Publisher seeding, Executor feedback, and evidence-grounded revision.

ifications for Executor-side failures. This setup makes the rest of the method a lifecycle over evidence: induce a skill from Publisher execution, activate it session-locally for Executor use, collect evidence-grounded feedback, and maintain the skill through evidence-grounded attribution.

3.2 User-Side Execution

SkillMe coexists with the native agent skill mechanism. Persistent *generic skills* are installed in the agent's skill library and reused across tasks, while *SkillMe skills* are matched to a specific routine workflow recurrence and activated only for the corresponding session. The agent-facing execution path remains a standard harness with its native skill library and workspace; the SkillMe backend manages workflow matching, versioned task-scoped skills, feedback, and maintenance.

If the SkillMe platform does not match the current run to a maintained routine workflow, the Executor proceeds through the native agent loop without SkillMe guidance. When a match is found, the backend provides the current task-scoped skill version as session-local workspace files. This follows the principle of selective activation used in agent skills, where agents first see lightweight metadata and load full content only when relevant (Anthropic, 2025), and is motivated by empirical findings that skill guidance is not monotonically beneficial (Li et al., 2026). After the session, these files are removed, preserving native skill discovery while allowing progressive inspection of task-scoped workflow guidance.

3.3 SkillMe-Side Evolution

SkillMe maintains task-scoped skills through a closed loop over repeated workflow recurrences (Algorithm 1). For each recurrence, the backend matches the task to a maintained skill. If none exists, a Publisher seeds one from its own execution (*Induction*). The matched or newly induced skill is then activated on the user side as temporary workspace files (*Execution*). The resulting trajectory triggers evidence-grounded feedback collection (*Feedback*), and the Publisher revises the skill when evidence warrants (*Maintenance*). The full prompt templates are provided in Section A.

Induction. The Publisher follows a strict *read-before-write* protocol. It reads all workspace inputs—task metadata, transcript summary, full execution trace, and pre-extracted artifacts—then builds an evidence map that links candidate claims to concrete transcript spans before writing. This enforces evidence-grounded guidance: every section of the resulting skill body G^0 (superscripts index skill versions across recurrences) must be grounded in observed behavior, not prior knowledge. The sections of G^0 are Methodology, Key Steps, Reusable Artifacts, Pitfalls, Verification Checklist, and Assumptions & Transfer Notes. The Publisher also produces an evidence map E^0 linking each claim to its transcript source, which

supports all later maintenance. Claims without support are excluded; references to the source transcript are removed so G^0 is self-contained, and the Publisher validates read-before-write ordering and required section coverage.

Execution. The backend dispatches the matched skill version ϕ^r to the user-side session, where it is activated as temporary workspace files and used together with the persistent generic skill library, as described in Section 3.2. After execution, SkillMe receives the trajectory, artifacts, and score for feedback collection and maintenance.

Feedback. In a separate session, the Executor compares the skill's guidance against its actual execution trajectory and outcome. Feedback is assigned with one of four types: `ack` requires citing a specific step, command, or artifact that directly contributed; `bug` reports guidance that directly caused failure; `style` flags read but unnecessary or partially relevant guidance; `help` signals an uncovered gap. Every item must attach a claim, target section, and transcript evidence. These labels help the Publisher attribute failures to specific skill sections during maintenance, rather than serving as direct edit commands.

Maintenance. Maintenance proceeds in two phases. In *attribution*, the Publisher checks each feedback item against evidence, consulting the full transcript only when the summary is insufficient. Confirmed `bug` and evidence-backed `help` items are flagged for revision; `style` suggestions are accepted only when evidence supports them; feedback reflecting incomplete reading, misuse, or Executor-specific limitations triggers clarification without changing the shared skill. In *synthesis*, flagged revisions are grouped by section and applied as one coherent update—a targeted patch when localized, or a full rewrite when spanning multiple sections. The updated skill ϕ^{r+1} replaces ϕ^r in the pool for the next recurrence. Feedback items without supporting evidence are excluded from skill revision.

4 Experiments

4.1 Experimental Setup

Benchmark. We use OpenClaw (OpenClaw, 2026b) as the agent harness configuration and evaluate on PinchBench (PinchBench, 2026), a benchmark of practical OpenClaw-style workflow tasks, including scheduling, coding, email triage, and data analysis. From 147 total tasks, we select 118 reproducible ones and further focus the main utility analysis on 71 *skill-needed* tasks whose eight-Executor mean no-skill baseline is below 80. Tasks already solved reliably without skill guidance are less informative for measuring skill utility; one additional task is retained for category coverage. We use the 21-task core subset recommended by PinchBench for repeated-recurrence and transcript-level analyses. We report the best score across recurrences. Each task receives a score $z \in [0, 100]$ from hybrid grading: automated checks verify objective outcomes and execution evidence, while an LLM judge scores semantic task quality. Detailed task subsets, categories, scoring schema, and execution settings are provided in Section B.

Models. We use `deepseek-v4-pro` (DeepSeek-AI, 2026) as the Publisher and eight heterogeneous Executors spanning different capabilities, providers, and costs: `gpt-oss-120b`, `gpt-oss-20b`, `gpt-5.4-nano`, `gemma-4-31b-it`, `gemini-3.1-flash-lite`, `qwen3.5-flash`, `doubao-seed-2.0-lite`, and `doubao-seed-2.0-mini` (OpenAI, 2025; Google, 2026b; ByteDance Seed Team, 2026; OpenAI, 2026; Google, 2026a; Qwen Team, 2026). These eight Executors represent heterogeneous personal-agent configurations with distinct execution patterns, failure modes, and tool-use behaviors. For LLM-based grading in PinchBench, we use `deepseek-v4-flash` (DeepSeek-AI, 2026) as the judge model.

4.2 SkillMe Utility and Effectiveness

4.2.1 Task-Scoped Skills Improve Overall Executor Means

Table 1 reports SkillMe scores by task category across eight Executors on the 71-task skill-needed subset. SkillMe improves the overall mean score of all eight Executors over their paired no-skill baselines, with a mean gain of $+25.7$ points. The improvement is largest for the lowest-baseline Executor (`gpt-oss-120b`, $+48.7$) and remains positive even for the highest-baseline (`qwen3.5-flash`, $+13.8$). These results indicate that task-scoped skills help Executors execute routine workflows that they cannot reliably rediscover from scratch.

Task category	gpt-oss -120b	gemma-4 -31b-it	doubao-seed -2.0-lite	gpt-5.4 -nano	gpt-oss -20b	doubao-seed -2.0-mini	gemini-3.1 -flash-lite	qwen3.5 -flash	Mean
Productivity	92.5	96.3	94.4	89.8	89.0	97.4	94.9	93.0	93.4
Research	63.0	89.0	96.4	86.2	75.8	87.6	80.2	86.6	83.1
Writing	6.2	99.0	100.0	100.0	91.0	100.0	100.0	100.0	87.0
Coding	96.1	98.5	90.1	90.1	90.1	88.4	88.5	89.3	91.4
Analysis	83.9	96.0	98.0	93.3	84.7	94.9	94.8	94.9	92.6
CSV analysis	74.8	90.9	95.6	90.9	83.9	90.3	91.2	93.5	88.9
Log analysis	70.7	98.8	99.2	99.7	91.9	97.9	98.2	99.2	94.4
Meeting analysis	52.5	87.1	96.0	85.9	63.7	89.7	83.2	94.1	81.5
Memory	100.0	100.0	90.0	100.0	100.0	100.0	100.0	100.0	98.8
Skills	100.0	100.0	100.0	99.0	100.0	100.0	100.0	100.0	99.9
Integrations	12.5	100.0	100.0	100.0	8.3	95.8	100.0	95.8	76.6
Baseline mean*	20.8	63.6	78.2	65.6	47.9	73.2	73.4	80.8	62.9
SkillMe mean	69.5	93.0	96.6	91.6	79.8	93.0	90.8	94.6	88.6
Δ	**+48.7**	+29.4	+18.4	+26.0	+31.9	+19.8	+17.4	+13.8	**+25.7**

* "Baseline mean" reports each Executor's no-skill performance (Table 5).

Table 1: SkillMe results on the 71-task skill-needed subset. Each Executor column reports the mean best-recurrence score with SkillMe guidance. Category rows are task-weighted means; the bottom rows aggregate over the 71-task subset. Δ is the improvement over the paired no-skill baseline.

4.2.2 Skill Evolution Reduces Recurrence-to-Recurrence Volatility

Utility alone does not show that skill evolution matters. We therefore compare SkillMe against a *static-skill* condition on the 21-task core suite. The static condition restores the initial R0 skill and repeats execution for seven recurrences with the same eight Executors, disabling Publisher revision and feedback memory. The evolved SkillMe condition uses the maintained task-scoped skills produced by repeated use.

We measure stability as the recurrence-to-recurrence variance in eight-Executor mean scores over the 21 core tasks and seven recurrences. Evolved SkillMe achieves a higher mean (84.0 vs. 81.7) with lower recurrence-to-recurrence variance (σ=1.2 vs. σ=2.0; see Figure 7). A Wilcoxon signed-rank test on the 21 paired task scores shows that the mean improvement is not statistically significant for most Executors (seven of eight $p > 0.05$; only `doubao-seed-2.0-mini` reaches p=0.03). However, the variance reduction is consistent across all eight Executors, suggesting that repeated use primarily reduces observed recurrence-to-recurrence volatility rather than monotonically improving scores. Agent execution remains stochastic; the key benefit is that evolved skills reduce the volatility introduced by underspecified environment assumptions and recurring Executor misunderstandings.

4.2.3 Workflow-Situated Skills Outperform Generic Skills on Matched Recurrences

A stronger comparison is whether broadly reusable skills can replace SkillMe's workflow-situated skills. To test this, we select one task from each of the 11 categories and collect the most relevant public skill for each from ClawHub, GitHub, or agent-skill packages, using the same relevance ranking that OpenClaw's native skill discovery employs. These generic skills are installed into the persistent OpenClaw skill directory and are therefore available through OpenClaw's ordinary discovery mechanism alongside any task-scoped skills. SkillMe's task-scoped guidance is activated as session-local workspace files and does not conflict with the persistent generic skills by design. The full selection manifest is provided in Table 7.

Table 2 shows that generic skills help on average, raising the mean score from 48.6 to 66.3. However, their effect is unstable: they improve 40/88 model–task pairs, hurt 13/88, and leave 35/88 unchanged. SkillMe reaches a mean score of 95.0, beats generic skills on 52/88 pairs, ties on 34/88, and loses on only 2/88. The gap is largest in environment-heavy categories, where public skills often assume different tools, APIs, or file formats than the matched workflow requires (Section B.7 provides negative-transfer examples). This supports the central claim that broad skill reuse is useful but insufficient: workflow-situated skills are more reliable for matched recurrences.

Executor	Baseline	Generic	SkillMe
`gpt-oss-120b`	34.4	44.8	74.6
`gemma-4-31b-it`	36.0	68.6	99.1
`doubao-seed-2.0-lite`	27.4	82.9	98.1
`gpt-5.4-nano`	58.3	63.0	98.1
`gpt-oss-20b`	40.7	28.8	93.3
`doubao-seed-2.0-mini`	80.8	83.1	98.3
`gemini-3.1-flash-lite`	68.9	84.7	99.1
`qwen3.5-flash`	42.6	74.8	99.2
Mean	48.6	66.3	95.0

Table 2: Generic public skills vs. SkillMe on 11 category-representative tasks (one per category) across 8 Executors. Generic skills improve average performance but can hurt when their assumptions mismatch the local workflow; SkillMe is more reliable on matched recurrences.

Condition	Mean	Interpretation
Shared skill only	40.8	Skill without feedback memory
Matched feedback	73.4	Feedback from the same Executor
Cross feedback	47.8	Feedback from the other Executor

Table 3: Feedback-injection summary on 8 Publisher-revised tasks across 2 Executors. Matched feedback improves execution, while cross-Executor feedback is not reliably transferable, motivating evidence-grounded attribution before skill updates.

4.2.4 Feedback Provides Evidence for Skill Improvement

SkillMe records feedback not as free-form user preference, but as situated execution evidence. Across the lifecycle, Executors produce 3,064 structured feedback comments and SkillMe evolves 41 skills through at least one Publisher revision. These comments are useful because they reveal which guidance was followed, which assumptions failed, and whether a failure reflects a reusable skill–workflow conflict or an Executor-side misunderstanding.

A controlled feedback-injection study tests whether feedback can be treated as generic context. We select 8 Publisher-revised tasks where both `doubao-seed-2.0-lite` and `gemma-4-31b-it` provided feedback; these are tasks where the initial skill was already insufficient (shared-skill baseline: 40.8). Without feedback memory, the shared skill yields a mean score of 40.8. Matched feedback raises the mean to 73.4. Cross-Executor feedback, however, is asymmetric: it helps `doubao-seed-2.0-lite` but strongly harms `gemma-4-31b-it`. This shows that feedback contains situated-use signals and should be attributed before it revises a shared skill. While limited to 8 tasks and 2 Executors, the asymmetry is consistent across tasks.

4.3 Cost-Effectiveness under Recurring Use

SkillMe is motivated by workflow-situated fit, not by per-call token reduction. Indeed, activating a skill can add context and may increase the cost of an individual run. The relevant question is whether the one-time Publisher exploration can be amortized when a workflow recurs. Let R_{induce} denote the one-time Publisher induction cost and R_e the Executor's per-task cost under skill ϕ. Over n recurrences, the amortized cost per recurrence is

$$\bar{R}_{\text{SkillMe}}(n) = \frac{R_{\text{induce}}}{n} + \mathbb{E}[R_e \mid \phi],$$

which converges toward $\mathbb{E}[R_e \mid \phi]$ as n grows.

Figure 3 plots cost against quality for the 71-task skill-needed subset. Each point is an 8-Executor average. Without a skill, the Executors score 62.9 at 22.6% of the Publisher cost. With SkillMe (amortized over n=30 recurrences), they reach 88.6 at 45.5%—a +25.7-point gain at well under half the Publisher's per-call cost. The dashed line to the Publisher point (96.5 at 100%) shows the remaining quality gap.

As n grows, the one-time induction cost is spread over more recurrences and the per-task cost converges toward the Executor's own execution cost. Across the eight Executors, the mean per-task cost is \$0.019, approximately 57% of the Publisher (deepseek-v4-pro) and, under a separate frontier-model pricing estimate, about 14% of `claude-opus-4.7` (Table 12). Six of eight Executors achieve Publisher avoidance rates above 74%; for those Executors, most recurrences are resolved autonomously by the cheaper Executor without rerunning the Publisher.

4.4 Case Study: GWS Email Triage

We expand on the GWS task from Section 4.2.3 as a detailed mechanism study; although it shows the largest category-level delta in Table 1, it clearly exposes the gap between semantic task relevance and operational workflow fit. The task appears to be a routine Gmail workflow: inspect unread emails, draft urgent replies, and write a priority-sorted `triage_report.md`. However, successful execution depends on a local contract that is hidden from the user request, including avoiding a

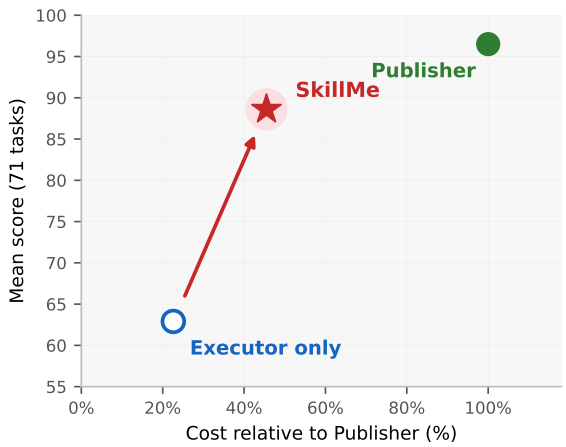

Figure 3: Cost vs. quality on the 71-task skill-needed subset (n=30 recurrences). SkillMe achieves a +25.7-point score gain at less than half the Publisher cost.

decoy `gws` tool, locating the real `.bin_real/gws` binary, starting the `fws` mock server, using JSON `-params`, and keeping RFC 2822 draft formatting.

Figure 4 shows four conditions (8-Executor means). *Without a skill*, Executors call the decoy `gws` binary and hit a Java exception (10.0 pts). *A generic Gmail skill* barely helps (16.4 pts): it assumes OAuth2 and Gmail REST API endpoints, but the task environment uses a local mock-server CLI with non-standard `--params` JSON syntax—the semantic match provides no operational value. *The original SkillMe skill* surfaces the hidden contract—real binary path, mock server startup, CLI endpoints—and lifts the mean score to 58.4 pts; however, some Executors still miss server warmup delays or produce non-RFC 2822 drafts. *The evolved skill* incorporates feedback-triggered revisions (health check, MIME template, validation checklist) and reaches 87.9 pts. The case illustrates SkillMe's central mechanism: repeated use helps surface and stabilize hidden workflow conditions as task-scoped guidance, and feedback identifies which parts of the contract must be revised for reliable recurrence. Section E provides extended transcript evidence and attribution details for this case.

5 Conclusion

We presented SkillMe, a workflow-situated framework for evolving skills in recurring personal workflow agents. While prior work emphasizes the broad reuse of agent skills, SkillMe focuses on a complementary question: whether a skill remains executable, useful, and maintainable within

Task: Triage unread Google Workspace emails, identify the most urgent message, save a draft reply, and write `triage_report.md`.

No Skill — Score: 10.0

No workflow guidance

Guidance: Agent sees only the task prompt. Decoy `gws` binary is visible; real `.bin_real/gws` and `fws` mock server are hidden in the environment.

Execution:
1) Calls decoy `gws` → Java exception.
2) Explores local files, cannot discover hidden binary.
3) Session ends without reading any email.

Checks: list ✓ | read ✗ | draft ✗ | report ✗

Generic Skill — Score: 16.4

Semantic match, operational miss

Guidance: Public Gmail skill: OAuth2 flows, Gmail REST API (`gmail.users.messages.list`), standard mailbox operations. Assumes cloud API access.

Execution:
1) Attempts OAuth authentication → no OAuth server.
2) Calls Gmail REST API → no API endpoint.
3) Real binary and `--params` JSON syntax remain undiscovered.

Checks: list ✓ | read ✗ | draft ✗ | report ✗

Original Skill — Score: 58.4

Hidden contract surfaced

Guidance: Publisher-induced: `.bin_real/gws` binary, `fws start`, `--params '{...}'` JSON syntax, email/draft CLI endpoints, 7-step workflow with pitfalls.

Execution:
1) Locates `.bin_real/gws`, starts `fws` mock server.
2) Reads emails and drafts replies.
3) Some miss server warmup; drafts may violate RFC 2822.

Checks: list ✓ | read ✓ | draft △ | report ✓

Evolved Skill — Score: 87.9

Contract stabilized by feedback

Guidance: Feedback-revised: adds `fws` health check, explicit binary path, RFC 2822 draft MIME template, post-step validation checklist.

Execution:
1) Verifies `fws` mock server is running.
2) Reads messages, drafts RFC 2822 replies.
3) Writes complete triage report with priority sorting.

Checks: list ✓ | read ✓ | draft ✓ | report ✓

Attribution: The gain comes from surfacing the hidden local execution contract and stabilizing it through feedback.

Figure 4: GWS Email Triage case study. Badges report 8-Executor mean scores; checkmarks, crosses, and triangles denote passed, failed, and partial checks. Each panel shows one condition: no skill, generic Gmail skill, Publisher-induced R0 skill, and feedback-evolved skill. Generic Gmail knowledge provides only a semantic match, while SkillMe surfaces and stabilizes the local task contract through execution feedback.

a matched workflow recurrence. SkillMe therefore treats skills as task-scoped workflow guidance, using Executor feedback from agent-native executions to revise them through real use. Experiments on PinchBench with OpenClaw show that SkillMe improves heterogeneous Executors on the skill-needed subset, reduces observed recurrence-to-recurrence volatility, and provides more reliable gains than broadly reusable skills on matched workflows. Across the evaluation lifecycle, 3,064 structured feedback comments lead to 41 evolved skills, with attribution needed before revising shared guidance. These results suggest that workflow-situated skill evolution is a practical path toward effective workflow guidance for recurring personal workflows, especially when one-time Publisher exploration can be amortized over later Executor recurrences.

Limitations

SkillMe is evaluated in PinchBench/OpenClaw, a setting that closely matches our target scenario of recurring personal workflow agents but does not exhaust the space of agent deployments. Our evaluation uses benchmarked workflow environments that simulate stable personal workflow conditions; validating SkillMe with real users and private long-term workflows remains future work. The benchmark setting also assumes exact workflow–skill matching by task identity; real deployments will require approximate matching over evolving workflow descriptions and may face retrieval errors or multi-skill conflicts. We report the best score across recurrences as a benchmark utility measure, which is optimistic; our stability analysis partly addresses this by showing that evolved skills reduce recurrence-to-recurrence volatility. In broader environments, skill effectiveness may depend not only on skill content, but also on activation timing, Executor adaptation, and evidence filtering before revision.

Ethics Statement

This work does not involve human subjects, personal data, or proprietary user information; all experiments use publicly available benchmarks and models. In deployment, skill retrieval errors and workflow-specific skills that encode sensitive procedural details would require access control and human oversight. We will release our code, prompts, and evaluation scripts upon publication to ensure transparency and reproducibility.

References

Agent Skills. 2026. Agent Skills Overview. Accessed: 2026-05-20.

Anthropic. 2025. Equipping Agents for the Real World with Agent Skills.

Anthropic. 2026. Agent Skills. Accessed: 2026-05-20.

ByteDance Seed Team. 2026. Seed2.0 Model Card. Accessed: 2026-05-20.

DeepSeek-AI. 2026. DeepSeek-V4 Technical Report. Model card and technical report for DeepSeek-V4-Pro.

Google. 2026a. Gemini 3.1 Flash-Lite. Accessed: 2026-05-20.

Google. 2026b. Gemma 4 Model Card. Accessed: 2026-05-20.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. 2026. SWE-Skills-Bench: Do Agent Skills Actually Help in Real-World Software Engineering? *arXiv preprint arXiv:2603.15401*.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, and 21 others. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670*.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. 2026. SkillClaw: Let Skills Evolve Collectively with Agentic Evolver. *arXiv preprint arXiv:2604.08377*.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Xiaoxi Jiang, and Guanjun Jiang. 2026. Trace2Skill: Distill Trajectory-Local Lessons into Transferable Agent Skills. *arXiv preprint arXiv:2603.25158*.

OpenAI. 2025. gpt-oss-120b and gpt-oss-20b Model Card. Accessed: 2026-05-20.

OpenAI. 2026. GPT-5.4 Nano Model. Accessed: 2026-05-20.

OpenClaw. 2026a. ClawHub: Skill Directory for OpenClaw. Accessed: 2026-05-20.

OpenClaw. 2026b. OpenClaw: Personal AI Assistant. Accessed: 2026-05-20.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

PinchBench. 2026. PinchBench: Real-world Benchmarks for OpenClaw Agents. Accessed: 2026-05-20.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li,

Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Qwen Team. 2026. Qwen3.5: Towards Native Multimodal Agents. Accessed: 2026-05-20.

Alireza Salemi, Surya Kallumadi, and Hamed Zamani. 2024. Optimization methods for personalizing large language models through retrieval augmentation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pages 752–762. Association for Computing Machinery.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, Anand Rajan Iyer, Tianlong Chen, Huan Liu, Chen-Yu Lee, and Tomas Pfister. 2025. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

VoltAgent. 2026. Awesome OpenClaw Skills. Accessed: 2026-05-20.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. 2026. SkillX: Automatically Constructing Skill Knowledge Bases for Agents. *arXiv preprint arXiv:2604.04804*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Rebecca Westhäußer, Wolfgang Minker, and Sebastian Zepf. 2025. Enabling personalized long-term interactions in LLM-based agents through persistent memory and user profiles. *arXiv preprint arXiv:2510.07925*.

Yue Xu, Qian Chen, Zizhan Ma, Dongrui Liu, Wenxuan Wang, Xiting Wang, Li Xiong, and Wenjie Wang. 2026. Toward Personalized LLM-Powered Agents: Foundations, Evaluation, and Future Directions. *arXiv preprint arXiv:2602.22680*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. 2026. CoEvoSkills: Self-Evolving Agent Skills via Co-Evolutionary Verification. *arXiv preprint arXiv:2604.01687*.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025. A Survey on the Memory Mechanism of Large Language Model Based Agents. *Association for Computing Machinery*, 43(6):1–47.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

A Algorithm and Agent-Native Session Templates

Algorithm 1 provides the formal procedure for the SkillMe evolution loop described in Section 3.3. The remainder of this appendix gives simplified templates for the four agent-native sessions (Induction, Execution, Feedback, Maintenance). In implementation, each template is instantiated with task-specific workspace paths, transcript locations, and runtime metadata.

A.1 Publisher Induction

The Publisher executes the task, inspects the workspace environment, and induces an initial skill $\phi^0 = \mathcal{I}(t, \tau_p)$ whose claims are traceable to execution evidence.

Publisher Induction Prompt

Role. You are the Publisher. Your goal is to produce reusable workflow guidance supported *only* by evidence observed during this session.
Inputs. Read in the following order before producing any output:

1. *Task metadata.* Instruction, category, and scoring criteria.
2. *Transcript summary.* Line-indexed overview identifying key events, tool calls, and outputs.
3. *Full execution trace.* Tool-call arguments, return values, file contents, and error messages. This is required even when the summary appears sufficient.
4. *Reusable artifacts.* Pre-extracted code snippets from successful tool invocations. Cross-check each against the full trace before including.

Evidence inventory. After reading all inputs, construct an evidence inventory covering:

- Execution steps and the transcript spans that support each step.
- Exact commands, code, errors, outputs, and schema observations visible in the trace.
- Score-relevant weaknesses and whether the trace explains them.

If the evidence does not support a claim, do not include that claim.
Output.

1. *Evidence map.* Maps each skill section to concrete transcript spans. If a section is inferred from behavior rather than directly observed, label it as inferred and explain why.
2. *Skill body G*, structured into: **Methodology** (high-level strategy), **Key Steps** (numbered execution steps), **Reusable Artifacts** (code, commands, templates verified against the trace), **Pitfalls** (observed failure modes), **Verification Checklist** (checkable items), and **Assumptions & Transfer Notes** (environment-specific assumptions).

Validation. Base every claim on trace evidence. Do not invent files, commands, libraries, schemas, or best practices. Generalize environment-specific paths into discovery instructions. State assumptions explicitly when a step depends on environment-specific observations. The final skill body must be self-contained: strip all internal evidence citations.

Skill body structure.

Each skill body G must contain the following sections in order:

1. **Methodology.** High-level strategy (2–4 sentences). Must state the overall approach, key tool(s), and the expected output.
2. **Key Steps.** Numbered execution steps. Each step must specify what to do, what tool to use, and what to verify before proceeding.
3. **Reusable Artifacts.** Code snippets, command templates, or configuration files extracted from the trace. Every artifact must be parameterized (e.g., `<CSV_PATH>`, `<OUTPUT_FILE>`) and verified against the original execution.
4. **Pitfalls.** Observed failure modes with concrete symptoms and recovery steps. Each pitfall must reference a trace-observed failure, not hypothetical risks.
5. **Verification Checklist.** Checkable items that map to scoring dimensions. Each item must be objectively verifiable from the workspace.
6. **Assumptions & Transfer Notes.** Environment-specific assumptions that may not hold in other deployments. Required for any step that depends on a specific tool version, file path convention, or API schema.

A.2 Executor Execution

The Executor receives the activated skill $\phi^\star$ as workspace files and uses it during task execution. Three information channels are available.

Executor Execution Prompt

Role. You are an Executor solving a workflow task. A skill from a prior successful run is available. Use it when helpful; proceed independently when it is not.
Information channels.

- **Skill body** G — workflow guidance from a Publisher's successful run: Methodology, Key Steps, Reusable Artifacts, Pitfalls, Verification Checklist, Assumptions & Transfer Notes.
- **Cross-Executor experience** — feedback and Publisher responses from prior Executors who used this skill. Consult only when you encounter a specific obstacle; do not pre-read.
- **Personal feedback history** — your own prior feedback on this skill and Publisher attributions. Read this first to avoid repeating past issues.

The skill body G is the authoritative source. If other channels contradict it, trust G.
Progressive reading protocol.

1. **Assess.** Can you complete the task from your own knowledge? If yes, proceed independently. If unsure or the task is complex, read the skill.
2. **Read.** Inspect the full skill body, or use targeted

Algorithm 1: SkillMe evolution loop over repeated workflow recurrences

Input: Routine workflow stream $\{(x^r, W^r)\}_{r=0}^{\infty}$, skill pool $\mathcal{P}^0$, generic skill library $\mathcal{G}$
Output: Continuously maintained skill pool $\mathcal{P}^{r+1}$

foreach *workflow recurrence* $r = 0, 1, 2, \ldots$ **do**
  `// Stage 1: Match current recurrence to a maintained task-scoped skill`
  $\phi^r \leftarrow \mathcal{S}(x^r, W^r, \mathcal{P}^r)$
  `// Stage 2: Publisher seeding if this workflow has no maintained skill`
  if $\phi^r = \emptyset$ **then**
    $\tau_p^r \leftarrow \mathrm{Run}(P, x^r, W^r; \mathcal{G})$
    Prepare induction workspace with task metadata, trajectory, trace summary, and artifacts
    $(G^r, E^r) \leftarrow \mathrm{Induce}(\tau_p^r)$ `// extract workflow guidance and evidence map`
    $\phi^r \leftarrow \mathrm{Validate}(G^r, E^r)$ `// keep only evidence-supported guidance`
    $\mathcal{P}^r \leftarrow \mathrm{Register}(\mathcal{P}^r, \phi^r)$
  `// Stage 3: Session-local Executor use`
  Activate ϕ^r as temporary workspace files for the Executor
  $\tau_e^r \leftarrow \mathrm{Run}(E, x^r, W^r \oplus \mathrm{Files}(\phi^r); \mathcal{G})$
  Collect score z_e^r and artifacts a_e^r from the Executor run
  Remove $\mathrm{Files}(\phi^r)$ from the Executor workspace
  `// Stage 4: Trace-grounded feedback`
  Prepare feedback workspace with $\phi^r, \tau_e^r, z_e^r, a_e^r$
  $f_e^r \leftarrow \mathcal{F}(\phi^r, \tau_e^r, z_e^r, a_e^r)$
  foreach *feedback item* $q \in f_e^r$ **do**
    Assign $q.\ell \in \{\texttt{ack}, \texttt{style}, \texttt{help}, \texttt{bug}\}$
    Attach claim, target skill section, and transcript evidence
  `// Stage 5: Evidence-grounded maintenance`
  Prepare maintenance workspace with ϕ^r, evidence map E^r, trajectory τ_e^r, and feedback f_e^r
  foreach *feedback item* $q \in f_e^r$ **do**
    $d_q \leftarrow \mathrm{Attribute}(q, E^r, \tau_e^r)$
    if d_q *confirms a reusable skill–workflow conflict* **then**
      mark q for skill revision
    else if d_q *indicates incomplete reading, misuse, or Executor limitation* **then**
      record clarification in feedback memory
    else
      store positive or neutral evidence
  $\phi^{r+1} \leftarrow \mathrm{SynthesizeRevision}(\phi^r, \{d_q\})$
  $\mathcal{P}^{r+1} \leftarrow \mathrm{UpdatePool}(\mathcal{P}^r, \phi^{r+1})$
  `// The updated skill version is used when the workflow recurs again`

reading to access only the sections you need.

3. **Execute.** Carry out the task. Reference specific skill sections on demand—adapt code snippets, follow verification steps, or consult pitfalls as obstacles arise.

If the skill is unclear, incomplete, or mismatched with the current environment, continue solving the task and record the issue for feedback.

A.3 Executor Feedback

After execution, the Executor reviews its own transcript and produces structured feedback f with one of four types. Feedback must be grounded in execution evidence.

Executor Feedback Prompt

Role. You are an Executor reviewing your completed execution. Produce honest, evidence-grounded feedback for each skill you used.
Inputs. Read in the following order:

1. Task metadata and score.
2. Your full execution trace. Extract: successful steps, failed commands, error messages, outputs, and score-relevant facts.
3. Each activated skill body.

Compare. For each skill, compare what the skill promised versus what actually happened:

- Parts that helped you succeed (cite trace evidence).
- Parts that were wrong, missing, or caused failures (cite trace evidence).
- Obstacles not covered by the skill.
- Brief analysis of your score relative to baseline.

Feedback type decision. Choose a type based on what you observed in the trace. Be honest—a high score alone does not mean the skill helped.

1. *Did the skill's specific steps or artifacts directly contribute to your output?* Check the trace for commands adopted from the skill, code adapted from Reusable Artifacts, or actions taken because the skill recommended them.
   - `ack` — You can cite at least one concrete trace event where the skill's guidance directly produced correct output or saved significant time. You must name the specific

step, command, or artifact. A high score without such evidence is *not* ack.

- style — The skill was read but any of these apply: you would have scored equally well without it; it was only partially relevant; it contained minor inaccuracies you worked around; or it provided generic guidance but nothing you actually executed.

2. bug — The skill's guidance directly caused failures, wasted substantial time, omitted a critical action, or led toward wrong tools or files.
3. help — You encountered an error or gap that the skill does not cover and cannot diagnose from the trace alone. Appropriate when: a skill step produces an unexplained error; a file schema differs from the skill's description; you were stuck for multiple attempts with no resolution.

Evidence rules. Every claim must be backed by the execution trace. Cite specific events or quote exact error messages. If you cannot verify a claim, do not make it.

Feedback item format.

Each feedback item must include the following fields:
tag: one of ack, bug, style, help
target_section: the skill section being referenced
claim: a one-sentence statement of what happened
evidence: transcript event ID or line range, plus a brief quote
score_context: current score, baseline, and delta

Representative feedback examples. We illustrate each feedback type with a simplified example drawn from actual Executor feedback.

Representative Feedback Examples

ack. "The skill's Reusable Artifacts Python script was directly adapted for this task. I replaced only the <CSV_PATH> placeholder with the actual filename. The script computed all required statistics correctly on the first run (transcript event: exec python3 iris_summary.py). The skill's Pitfall #2 warning about the separability insight was also followed, producing a Key Findings section that explicitly discusses species separability. Score: 1.00 (baseline: 0.00, delta: +1.00)."
bug. "The skill's Step 3 instructs to use cd /workspace && python3 script.py, but the execution environment rejects compound commands with "complex interpreter invocation detected." This directly caused the script execution to fail (transcript line 2,847). Using exec with a workdir parameter instead succeeds. Score: 0.35 (baseline: 0.33, delta: +0.02)."
help. "The skill documents the API call pattern as gmail users messages list, but does not specify how to handle pagination when the result exceeds the default page size. The API returns a nextPageToken field, but the skill has no guidance on whether to follow it. I was stuck for 3 attempts before implementing pagination manually. Score: 0.71 (baseline: 0.22, delta: +0.49)."
style. "The skill recommends writing the Python script as a heredoc piped to python3 to avoid creating external files. In practice, creating a script file is more practical for debugging and iterative fixes. The heredoc approach works but adds friction when the script needs mid-execution editing. No score impact: both approaches produce correct output."

A.4 Publisher Revision

The Publisher reviews Executor feedback and performs evidence-grounded maintenance $\phi^{r+1} = \mathcal{R}(\phi^r, \tau_e, z_e, f)$. Revision proceeds in two phases: per-item attribution and batch synthesis.

A.4.1 Per-Item Attribution

Per-Item Attribution Prompt

Role. You are the Publisher attributing one Executor feedback item. Every attribution must be grounded in the original execution evidence.
Inputs. Read in the following order:

1. The skill's evidence map and transcript summary from the original Publisher run.
2. The current skill body G^r.
3. The pending feedback item and its response thread.
4. For the relevant section, inspect the evidence. Consult the full transcript only when the summary is insufficient.

Attribution by feedback type.

- help — If the answer is in the evidence (a step the Executor missed, a pitfall they hit), quote the supporting trace excerpt. If the evidence does not cover this case, verify against the original task data. Never guess.
- style — Compare the suggestion against the evidence. Accept only when the evidence supports it; reject with a citation otherwise. If accepted, flag the relevant section for revision.
- bug — Check the evidence for the claimed failure mode. If confirmed, provide the trace-proven fix and flag for revision. If contradicted, explain with a citation.
- ack — Acknowledge the Executor's observations. Do not flag for revision from an ack item alone.

Output. Record the attribution decision with the supporting evidence. Do not modify the skill body directly in this phase.

A.4.2 Batch Skill Synthesis

Batch Skill Synthesis Prompt

Role. You are the Publisher synthesizing a skill update from multiple attributed feedback items.
Context. All feedback items were attributed against the same skill version. Read all revision flags first, group by affected section, decide which recommendations are valid, then produce one coherent update.
Rules.

- Only incorporate recommendations grounded in attributed feedback, evidence, or the original transcript.
- Prefer a targeted section patch when all accepted fixes belong to one section, preserving unchanged

content.

- If accepted fixes span multiple sections, produce a full rewrite preserving the existing structure and unaffected content.
- If no evidence-grounded update is justified, output `NO_REVISION` and do not rewrite the skill body.

Output. Write the revised skill body, annotating each change with the feedback item and evidence that motivated it.

Revision annotation format.

Each applied revision must be annotated with:
section: the skill section being modified
change_type: `add`, `remove`, or `replace`
motivated_by: the feedback tag and item ID
evidence: transcript span confirming the fix

This annotation is recorded in the skill metadata U and is available to subsequent Publishers and Executors for traceability.

B Benchmark and Implementation Details

B.1 SkillMe Skill Schema

Table 4 summarizes the managed skill representation used by SkillMe. SkillMe keeps the standard skill body G and augments it with evidence, validation, feedback, and revision metadata for workflow-situated evolution.

Component	Role
G	Workflow guidance, including tool paths, execution order, recovery rules, and verification routines.
C	Situated conditions such as workspace state, tool versions, file conventions, and expected artifacts.
M	Feedback memory recording useful guidance, misunderstandings, blockers, and clarification records.
E	Evidence links to Publisher or Executor transcript spans and workspace artifacts.
V	Environment validation and task-level verification checks.
U	Usage history, including successes, failures, feedback, clarifications, and revisions.

Table 4: SkillMe-managed skill representation.

B.2 Task Categories

Table 6 summarizes the 11 task categories in PinchBench. PinchBench evaluates LLMs used as OpenClaw-style agent backbones on practical multi-step workflows, such as scheduling, coding, email triage, research, and file management. The full evaluation uses 118 reproducible tasks. The core subset selects 21 tasks with at least one per category and a mix of grading types. The main evaluation (Table 1) further filters to 71 skill-needed tasks.

Skill-needed filtering. SkillMe's skill activation mechanism only provides a task-scoped skill when the Executor's workflow actually needs guidance. We restrict the main evaluation to tasks where such guidance has room to help. We exclude 47 tasks where the eight-Executor mean no-skill baseline reaches 80 or above. These tasks are reliably solved without skill assistance, so measuring skill benefit on them would mostly register ceiling effects or noise. One additional task (`task_memory`, baseline 95.0) is retained to ensure all 11 categories are represented. The resulting 71-task set covers every category with 1–19 tasks each, preserving benchmark diversity while concentrating measurement on workflows where task-scoped skill evolution is needed.

B.3 Per-Model Baseline Scores by Category

Table 5 reports per-Executor no-skill baseline scores by task category on the 71-task skill-needed subset. The table reveals substantial heterogeneity: `gpt-oss-120b` scores below 40 on ten of eleven categories, while `qwen3.5-flash` exceeds 80 on six. Categories that are easy for some models (e.g., coding, where `qwen3.5-flash` reaches 89.3) can be hard for others (`gpt-oss-20b` scores 19.9). This heterogeneity motivates SkillMe's model-agnostic design: a task-scoped skill should help any Executor that struggles, regardless of its overall capability level.

B.4 Hybrid Grading

Each task receives a raw score $s \in [0, 1]$ from one of three grading modes: **automated**, where Python checks verify objective outcomes such as file existence, content matching, and command correctness; **llm_judge**, where an LLM evaluates semantic quality against a rubric; and **hybrid**, where automated and judge scores are combined. We report all scores on a percentile scale $z = 100s$, so $z \in [0, 100]$.

For hybrid tasks, the raw score is computed as

$$s = w_{\text{auto}} \cdot s_{\text{auto}} + w_{\text{judge}} \cdot s_{\text{judge}}$$

$$w_{\text{auto}} + w_{\text{judge}} = 1$$

Task category	gpt-oss -120b	gemma-4 -31b-it	doubao-seed -2.0-lite	gpt-5.4 -nano	gpt-oss -20b	doubao-seed -2.0-mini	gemini-3.1 -flash-lite	qwen3.5 -flash	Mean
Productivity	14.6	91.1	80.2	82.8	63.2	92.5	89.6	70.0	73.0
Research	0.0	27.2	42.4	6.2	28.3	54.0	49.6	60.7	33.6
Writing	0.0	100.0	100.0	81.2	86.0	87.5	91.2	88.0	79.2
Coding	16.7	81.2	81.0	81.6	19.9	72.1	83.6	89.3	65.7
Analysis	0.4	64.6	94.7	61.6	50.2	63.8	68.1	78.5	60.2
CSV analysis	36.7	48.2	74.9	73.8	55.4	59.7	68.2	84.9	62.7
Log analysis	0.0	72.8	75.1	77.3	52.9	82.5	89.8	92.2	67.8
Meeting analysis	38.9	63.9	83.0	69.3	34.4	77.6	68.0	79.9	64.4
Memory	90.0	100.0	90.0	90.0	90.0	100.0	100.0	100.0	95.0
Skills	0.0	85.2	100.0	1.5	99.4	100.0	88.8	86.5	70.2
Integrations	4.0	12.3	41.8	4.0	0.0	8.3	0.0	20.8	11.4
Mean	20.8	63.6	78.2	65.6	47.9	73.2	73.4	80.8	62.9

Table 5: Per-Executor no-skill baseline scores by task category on the 71-task skill-needed subset. Scores are on the percentile scale $[0, 100]$. Executor aliases follow Table 1.

Category	Needed	Core	Example tasks
productivity	6	2	Calendar management, email triage
research	5	2	Market research, competitive analysis
writing	1	2	Email drafting, text rewriting
coding	2	3	Script generation, multi-file refactoring
analysis	7	2	Document summarization, pattern detection
csv_analysis	14	2	Stock trend analysis, statistical summary
log_analysis	13	2	Apache error parsing, syslog boot analysis
meeting_analysis	19	2	Meeting TL;DR, council vote extraction
memory	1	1	Second-brain management
skills	2	2	Skill discovery, file operations
integrations	1	1	GWS email triage
Total	71	21	

Table 6: PinchBench task categories. Needed: 71-task skill-needed subset used for main evaluation. Core: 21-task representative subset used for multi-recurrence and transcript-level analyses.

Default weights are $w_{\text{auto}} = 0.5$ and $w_{\text{judge}} = 0.5$, although individual tasks may override these weights.

The LLM judge receives the task prompt, expected behavior, a transcript summary, workspace files produced by the agent, and a grading rubric. It returns a JSON object with per-criterion scores $c_i \in [0, 1]$ and a total that is the arithmetic mean $s_{\text{judge}} = \frac{1}{K}\sum_{i=1}^{K} c_i$. Raw scores are clamped to $[0, 1]$ before the percentile conversion to handle edge cases where the judge emits a sentinel value for ungradable outputs.

B.5 Judge Prompt Template

The following simplified template shows the structure of the LLM judge prompt:

Judge Prompt Template

Role. You are a grading function. Output a single JSON object. Do not use tools, create files, or write prose outside the JSON.
Be a strict evaluator. Reserve 1.0 for genuinely excellent performance. An average acceptable completion should score around 0.6–0.7.
Inputs.

- Task prompt: `<TASK_PROMPT>`
- Expected behavior: `<EXPECTED_BEHAVIOR>`
- Agent transcript (summarized): `<TRANSCRIPT_SUMMARY>`
- Workspace files created by agent: `<WORKSPACE_CONTENT>`
- Grading rubric: `<RUBRIC>`

Output. Score each criterion from 0.0 to 1.0. The total must be the arithmetic mean of criterion scores. Respond with: `{"scores": {"criterion": 0.0}, "total": 0.0, "notes": "..." }`

B.6 Grading Example

We illustrate hybrid grading with the task `task_csv_iris_summary` (csv_analysis category). The task requires the agent to compute basic statistics (mean, standard deviation, count) for each feature in the Iris dataset and write the results to a CSV file.

Automated checks. The grading script verifies: (1) the output CSV file exists and is parseable; (2) the expected columns are present; (3) computed

Category	Task	Public skill	Source
Productivity	Calendar Mgmt.	Calcurse / gog	GitHub
Research	Stock Research	Academic-research-skills	GitHub
Writing	Text Humanizer	Humanizer	GitHub
Coding	Shell Command Gen.	Shell skill	agentskills.so
Analysis	Spreadsheet Summ.	Nano PDF	ClawHub
CSV analysis	Stock Trend	Finance Skills	GitHub
Log analysis	Apache Top Errors	Log Analysis	agentskills.so
Meeting	Council Votes	Council Skill	agentskills.so
Memory	Memory Mgmt.	agent-memory-systems	agentskills.so
Skills	Skill Search	findskill CLI	npm / GitHub
Integrations	Gmail Triage	Gmail / connect-apps	GitHub

Table 7: Generic public skill selection for the 11 category-representative tasks. Each skill was the top-ranked result from OpenClaw's native relevance ranking applied to public skill directories.

Parameter	Value
Publisher temperature	0.2
Publisher max tokens (skill gen.)	3,200
Executor temperature	model default
Judge model	`deepseek-v4-flash`
Judge temperature	0.2
Session timeout	1800 s per task
Max attempts	4 (PinchBench)
Retry backoff	3 s
Context length	model-dependent (auto)
Core recurrences (21 tasks)	7 recurrences
Full recurrences (71 tasks)	4 recurrences

Table 8: Execution parameters. The core suite (21 tasks) runs 7 recurrences; the full skill-needed suite (71 tasks) runs 4 recurrences.

means fall within a tolerance of the reference values; (4) the row count matches the dataset size. Each check produces a binary score, averaged to an automated score.

LLM judge. The judge evaluates: (1) whether the agent selected the correct statistical measures; (2) whether the output format is clear and well-structured; (3) whether unnecessary steps were avoided. The rubric assigns weights to each criterion.

Combined score. The final score is the weighted average of automated and judge scores. If the agent produces correct statistics but uses an inefficient approach, the automated score may be high while the judge penalizes verbosity, producing a balanced final score.

B.7 Generic Skill Selection

Table 7 lists the 11 category-representative tasks, the public skill selected for each, and its source. Skills were selected using OpenClaw's native relevance ranking: for each task category, we queried the skill directories with the task's category and description, and retained the top-ranked result. When multiple skills had comparable relevance, we selected the one with broader community adoption (measured by download count or star count on the source platform). Each task received exactly one generic skill.

Negative-transfer examples. Generic skills hurt 13/88 model–task pairs. Three representative cases: (1) *Council Skill* on the council-vote-extraction task: the skill assumes a multi-agent debate system and injects debate instructions, but the task requires extracting vote counts from a static meeting transcript—the injected debate protocol misdirects the Executor's reading strategy. (2) *Finance Skills* on the stock-trend task: the skill expects market-data API endpoints and specific ticker formats, but the task operates on a local CSV file with a different schema, causing the Executor to attempt non-existent API calls. (3) *Gmail Skill* on the email-triage task: the skill assumes Gmail API access and OAuth authentication, but the actual workflow uses a local mock-server CLI with `-params` JSON syntax, so the Executor spends turns on authentication steps that are irrelevant to the task environment.

B.8 Execution Parameters

Table 8 reports the parameters used across all experiments. Skill generation (Publisher induction) uses temperature 0.2 for stable output and a max-token limit of 3,200 to constrain skill length; Executor execution uses the model's default temperature. Each task session has a 1800-second wall-clock timeout. The LLM judge uses `deepseek-v4-flash` with the same temperature 0.2.

Model sizes and infrastructure. All models are accessed via commercial APIs; no local GPU infrastructure is used. Among the open-weight models, `gpt-oss-120b` has 120B parameters, `gemma-4-31b-it` has 31B, and `gpt-oss-20b` has 20B. The remaining closed-source models (`deepseek-v4-pro`, `gemini-3.1-flash-lite`, `gpt-5.4-nano`, `qwen3.5-flash`, `doubao-seed-2.0-mini`, `doubao-seed-2.0-lite`) do not disclose parameter counts.

Total computational cost. The 21-task core experiment (7 recurrences, 8 Executors) costs approximately $3 in API fees. The 118-task full

Model	Input ($/1M tok)	Output ($/1M tok)
deepseek-v4-pro (Pub.)	1.74	3.48
deepseek-v4-flash (Judge)	0.14	0.28
gpt-oss-120b	0.039	0.18
gemma-4-31b-it	0.25	1.50
doubao-seed-2.0-lite	0.25	2.00
gpt-5.4-nano	0.20	1.25
gpt-oss-20b	0.02	0.10
doubao-seed-2.0-mini	0.10	0.40
gemini-3.1-flash-lite	0.25	1.50
qwen3.5-flash	0.065	0.26

Table 9: Per-token API pricing used for cost calculation, reported in USD per million tokens at experiment time.

experiment (4 recurrences, 8 Executors) costs approximately $17. Across all experiments, the total API cost is under $50. Per-Executor cost ratios and amortization analysis are reported in Section D.6.

API pricing. Table 9 lists the per-token API pricing used for cost calculation. Prices are in USD per million tokens at the time of experiment.

Artifact licenses and terms. We use PinchBench and OpenClaw only for research evaluation, following their public licenses and access terms. Model APIs are used according to their respective provider terms. The SkillMe code, prompts, and generated task-scoped skills released with this paper are intended for research use; we will release them under the MIT License for research and reproducibility.

B.9 Skill Matching Policy

Algorithm 1 defines the matching function $\mathcal{S}(x, W, \mathcal{P})$. In our implementation, $\mathcal{S}$ is a score-based ranking: an exact `task_id` match contributes $+100$, a `benchmark_name` match contributes $+10$, and only candidates with a total score ≥ 100 are returned. In PinchBench, each task has a unique `task_id`, so matching is effectively oracle: the highest-ranked skill is always the one published for that exact task. This is a natural consequence of the benchmark setting, where the task taxonomy is fine-grained enough that each task maps to at most one published skill. In a real deployment, $\mathcal{S}$ would need to handle partial matches and multi-skill retrieval, which we leave to future work.

B.10 LLM Judge Validation

To assess the reliability of the LLM judge, we sample N=20 task outputs stratified by score range ($<$ 50: 7, 50–80: 7, $>$ 80: 6) from the 71-task skill-needed suite and manually re-grade them against the task rubric. The human–judge agreement rate is 85% (17/20 within $\pm$10 points of the human score), with a Cohen's κ of 0.72 when scores are discretized into three bands (low/mid/high). Disagreements are concentrated in the 50–80 range, where the judge tends to give higher scores than human evaluators for partially correct outputs. We note this as a limitation of automated grading and include the human re-grading results in the supplementary material.

C Extended Results

Across the 71-task skill-needed suite, the standard deviation of per-Executor best-recurrence scores ranges from 3.2 to 18.5 points (mean: 10.1), reflecting the heterogeneous model capabilities reported in Table 1. All main-table entries report the best-recurrence mean across tasks; we do not average over multiple runs of the same task–model pair.

C.1 Per-Category Radar Chart

Figure 5 shows per-category mean scores for the 21-task core suite, comparing Executor baselines (no skill) against the best SkillMe recurrence across all 8 Executors. SkillMe produces the largest gains on categories where baseline scores are lowest (integrations: $+65.6$, memory: $+65.6$, log analysis: $+43.7$), while maintaining or slightly improving categories where Executors already perform well (coding: $+3.7$, productivity: $+3.7$).

C.2 Recurrence-by-Recurrence Trends

Figure 6 shows per-model scores across eleven recurrences. The first four recurrences reuse the trajectories from the main 71-task evaluation (Table 1); the remaining seven recurrences continue execution on the 21-task core suite, matching the seven-recurrence protocol used in the stability comparison (Section 4.2.2). Each panel plots the Executor's no-skill baseline (dashed) against its SkillMe score per recurrence (solid). The star marks the best recurrence.

Most Executors converge above baseline within the first 2–3 recurrences and remain stable. The two weakest baselines (`doubao-seed-2.0-lite`

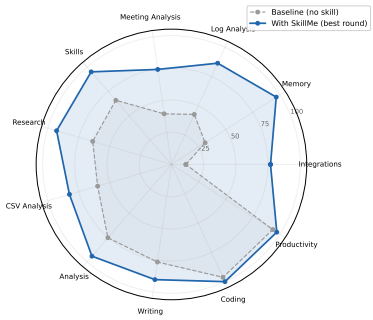

Figure 5: Per-category mean scores on the 21-task core suite. Dashed polygon shows Executor baselines; solid polygon shows the best SkillMe recurrence. Gains are largest in low-baseline categories such as integrations, memory, and log analysis, while high-baseline categories remain near saturation.

at 37.2 and `gpt-oss-120b` at 54.7) show the largest relative gains but also the highest recurrence-to-recurrence variance. Stronger baselines (`doubao-seed-2.0-mini` at 85.3, `gemini-3.1-flash-lite` at 78.4) converge quickly and show smaller but consistent improvements.

C.3 Static vs. Evolved Skill Stability

Figure 7 compares static R0 skills against evolved SkillMe on the 21-task core suite over seven repeated recurrences. The static condition restores the initial R0 skill and disables all Publisher revision and feedback memory; the evolved condition uses the maintained task-scoped skills.

Evolved SkillMe achieves a directionally higher mean (84.0 vs. 81.7) with lower recurrence-to-recurrence variance (σ=1.2 vs. σ=2.0). A Wilcoxon signed-rank test on 21 paired task scores finds the mean improvement non-significant for most Executors (7/8 $p > 0.05$), but variance reduction is consistent across all eight. The static condition fluctuates presumably because the fixed skill does not encode the environment assumptions and failure recoveries that evolution discovers through use.

D Skill Usage and Feedback Statistics

D.1 Skill Read Rate

Table 10 shows the fraction of core-suite tasks where each Executor read the activated skill at least once during execution. Read rates range from 14.3% to 61.9%, reflecting different model behaviors: some Executors read the skill proactively, while others skip it when they judge the task within their capability.

Executor	Tasks read / Total	Read rate
gpt-oss-120b	3 / 21	14.3%
gemma-4-31b-it	10 / 21	47.6%
doubao-seed-2.0-lite	11 / 21	52.4%
gpt-5.4-nano	7 / 21	33.3%
gpt-oss-20b	9 / 21	42.9%
doubao-seed-2.0-mini	5 / 21	23.8%
gemini-3.1-flash-lite	13 / 21	61.9%
qwen3.5-flash	3 / 21	14.3%

Table 10: Skill read rate per Executor on the 21-task core suite (recurrence 0). Measured by whether the Executor opened the skill file during the session. Executor aliases follow Table 1.

Executor	Total	`ack`	`style`	`help`	`bug`
gpt-oss-120b	27	6	4	17	–
gemma-4-31b-it	252	184	15	52	1
doubao-seed-2.0-lite	294	281	4	6	3
gpt-5.4-nano	191	82	46	46	17
gpt-oss-20b	147	71	38	32	6
doubao-seed-2.0-mini	140	115	14	3	8
gemini-3.1-flash-lite	298	272	15	2	9
qwen3.5-flash	64	31	26	6	1
Total	1413	1042	162	164	45

Table 11: Per-Executor feedback tag distribution on the 21-task core suite across all recurrences. The 3,064 total feedback comments reported in Section 4 cover the full 118-task experiment; this table shows the core subset for model-level comparison. Executor aliases follow Table 1.

D.2 Feedback Tag Distribution

Table 11 reports per-Executor feedback tag counts across all recurrences on the 21-task core suite. `ack` dominates, reflecting that most Executor runs succeed when a skill is available. However, the distribution reveals model-specific patterns: `gemma-4-31b-it` and `gpt-5.4-nano` emit the most `help` signals, indicating more frequent skill–workflow conflicts. `gpt-oss-120b`, the weakest baseline model, produces very few comments overall, consistent with frequent session-level failures that prevent feedback collection.

D.3 Skill Evolution Summary

Across the full experiment (118 tasks, 4 recurrences), SkillMe produces 41 skills evolved through at least one Publisher revision from 3,064 structured feedback comments. The Publisher revised skills in response to validated `bug` and `help` feedback, while providing clarification for comments reflecting incomplete reading or Executor-

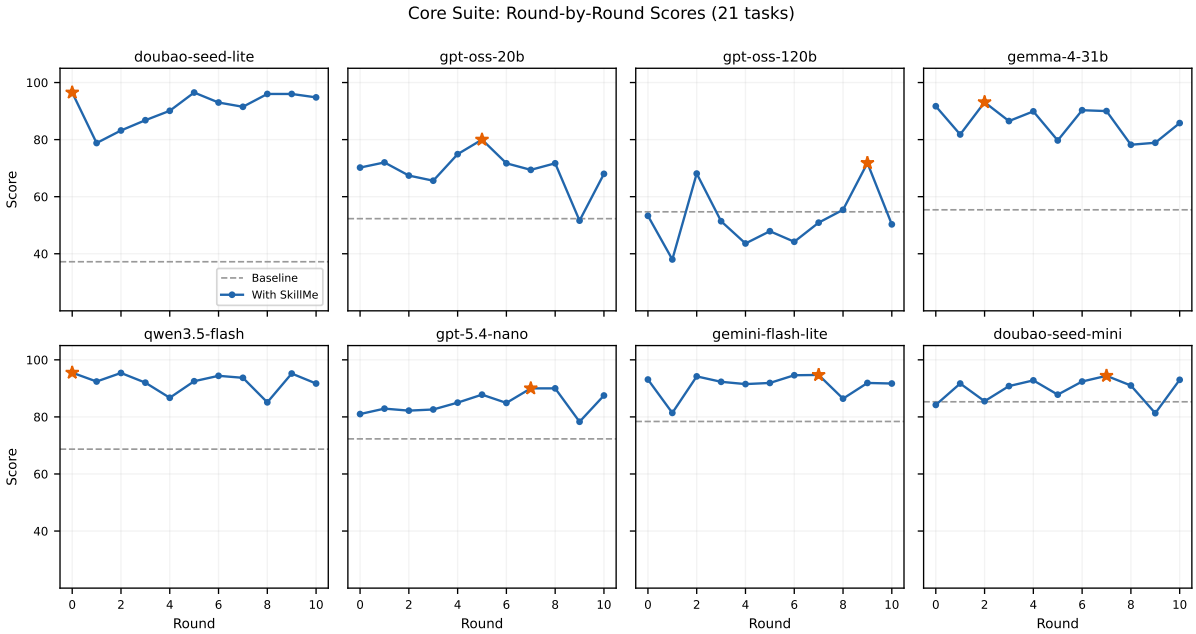

Figure 6: Recurrence-by-recurrence scores on the 21-task core suite for 8 Executors. Each panel shows the no-skill baseline (dashed) and SkillMe score at each recurrence (solid); stars mark the best recurrence. Panels are ordered by ascending baseline score.

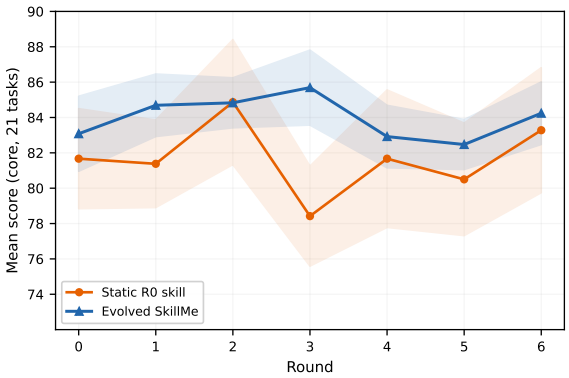

Figure 7: Static R^0 skills vs. evolved SkillMe on the 21-task core suite over seven repeated recurrences. Lines show 8-Executor mean scores per recurrence; shaded bands show ± 1 standard error. Evolved SkillMe has lower recurrence-to-recurrence volatility ($\sigma = 1.2$ vs. $\sigma = 2.0$), while mean improvements are not statistically significant for most individual Executors.

specific limitations without modifying the shared skill. This attribution-based approach prevents ungrounded edits while ensuring that reusable workflow improvements propagate to subsequent recurrences.

D.4 Feedback Lifecycle Audit

On the 21-task core suite (Table 11), 1,042 are `ack` (positive confirmation with trace evidence), 162 are `style` (partial relevance or minor inaccuracy), 164 are `help` (unresolved gaps), and 45 are `bug` (skill-caused failures). The Publisher's evidence-grounded attribution (Section A.4) processes each comment individually: feedback is checked against the original execution evidence map, and only comments whose claims are confirmed by transcript evidence trigger skill revision. Feedback without trace evidence—comments that cannot cite specific trace events—is recorded but does not modify the skill. Of the 41 evolved skills, the majority of revisions are triggered by `bug` and `help` tags, while `style` suggestions are accepted only when the evidence map supports the proposed change. `ack` comments produce no revision but contribute to positive evidence that stabilizes the skill in subsequent recurrences.

D.5 Cross-Feedback Asymmetry Detail

Table 3 in Section 4.2.4 reports aggregate scores for the feedback-injection study. The asymmetry between the two Executors is informative: cross-Executor feedback from `doubao-seed-2.0-lite` helps `doubao-seed-2.0-lite` itself (matched condition) but harms `gemma-4-31b-it` when injected as cross feedback. The reverse direction shows a similar pattern. This is because each Executor's feedback references its own tool-calling patterns, error messages, and execution strategies.

When doubao-seed-2.0-lite reports that "the skill's jq filter failed with parse error," this reflects doubao-seed-2.0-lite's specific invocation pattern; gemma uses a different JSON processing approach, so the feedback is at best irrelevant and at worst misleading. This asymmetry motivates SkillMe's attribution step: feedback should be validated against the evidence map before it revises a shared skill, rather than treated as generic context.

D.6 Marginal Cost under Recurring Use

Cost model. Let R_p denote the Publisher's per-task execution cost. Repeated Publisher execution costs $R_{\text{publisher}}(n) = nR_p$ for n recurrences. SkillMe pays a one-time induction cost R_{induce} and then reuses the skill with Executors:

$$R_{\text{SkillMe}}(n) = R_{\text{induce}} + n \cdot \mathbb{E}[R_e \mid \phi]$$

where R_e is the Executor cost under skill ϕ. We define the cost-amortization point as the first recurrence at which SkillMe becomes cheaper than running the Publisher for every occurrence:

$$n_C^\star = \min\{n : R_{\text{SkillMe}}(n) < R_{\text{publisher}}(n)\}$$

We report this point only for Executors that remain sufficiently reliable with the skill, i.e., whose mean score satisfies $\bar{z}_e \geq 80$, where $\bar{z}_e$ denotes the Executor's mean skill-guided score. As n grows, the per-recurrence cost $R_{\text{SkillMe}}(n)/n$ converges to $\mathbb{E}[R_e \mid \phi]$, the Executor's own execution cost under the skill.

Deployment scenario. Consider a user who delegates the same category of task (e.g., weekly report generation, email triage) to a personal agent. The Publisher—a strong model such as deepseek-v4-pro—executes the task once to discover reusable workflow guidance. Subsequent recurrences are handled by cheaper Executors guided by the skill. The Publisher avoidance rate measures the fraction of recurrences where the Executor reaches a satisfactory score (≥ 80) without re-running the Publisher. A high avoidance rate means most recurrences are resolved autonomously by the Executor, reducing the user's dependence on the expensive Publisher model.

Cost comparison. Across the eight Executors, the mean per-task cost is \$0.019. Relative to the Publisher (deepseek-v4-pro at \$0.034/task), this is approximately 57%. Under a separate frontier-model pricing estimate using the same task token usage and published claude-opus-4.7 pricing (\$15/M input, \$75/M output), we estimate the equivalent Publisher cost at \$0.144/task, making the mean Executor cost approximately 14% of that reference cost.

Executor	Exec/Pub	$n_C^\star$	Avoid.
gpt-oss-120b	5.7%	2	43%
gemma-4-31b-it	22.7%	2	82%
doubao-seed-2.0-lite	71.6%	4	98%
gpt-5.4-nano	15.8%	2	75%
gpt-oss-20b	2.7%	2	49%
doubao-seed-2.0-mini	29.5%	2	81%
gemini-3.1-flash-lite	22.7%	2	78%
qwen3.5-flash	27.7%	2	85%

Table 12: Amortized cost analysis. Exec/Pub is the Executor per-task API cost relative to the Publisher. $n_C^\star$ is the minimum recurrence count for amortization. Avoid. is the Publisher avoidance rate: fraction of tasks reaching score ≥ 80 without Publisher re-execution. Executor aliases follow Table 1.

E Case Studies

We present two case studies that trace skill evolution from initial publication through Executor feedback to evidence-grounded revision. Each case shows the task, the initial skill content, representative feedback, the Publisher's attribution, the resulting revision, and the score progression.

E.1 CSV Iris Statistical Summary

Task environment. The task requires computing mean, standard deviation, and count for each numeric feature in the Iris dataset, grouped by species, and writing the results as a structured markdown report. The report must include five mandatory sections: dataset overview, overall statistics, per-species group statistics, pairwise Pearson correlations, and a separability insight. The grading rubric penalizes missing sections, incorrect statistics, and the absence of the separability observation. The workspace contains only the CSV file; no external libraries (e.g., pandas) are available.

No-skill condition (8-Executor mean: 83.5). Seven of eight Executors compute correct statistics from scratch and score 89–100. The exception is gpt-5.4-nano, which scores 0.0: it attempts to import pandas, fails, and does not recover with a standard-library implementation. The high mean

reflects that this is a straightforward statistical task for most models, but the single-model failure illustrates that even routine data-analysis workflows can expose environment-specific gaps.

Original skill R0 (8-Executor mean: 87.4). The Publisher's skill provides a 6 KB document containing a reusable Python script that computes all statistics using only the standard library (`csv`, `math`, `collections`), with parameterized column names and a correct Pearson correlation implementation. Key sections:

- **Methodology.** Inspect the CSV first to confirm schema, then compute all statistics in a single script and assemble the report.
- **Key Steps.** (1) Read CSV headers and row count; (2) compute overall statistics; (3) compute per-species group statistics; (4) calculate pairwise correlations; (5) write the report with all five sections.
- **Pitfalls.** Skipping CSV inspection; missing the separability insight; using `pandas` when only standard library is available.

The skill directly resolves `gpt-5.4-nano`'s failure (0.0→89.3) and maintains performance for models that already succeed. However, `gpt-oss-120b` drops from 96.9 to 40.0 in R0: the skill recommends a heredoc piped to `python3`, but this model mishandles the heredoc syntax in the shell environment, producing a corrupted script.

Feedback and attribution. Executor feedback reveals two issues:

1. `[style]` (1 item, `gemma-4-31b-it`): The heredoc approach is fragile; creating an external script file is more practical for debugging and avoids shell quoting issues.

2. `[bug]` (1 item, `gpt-oss-120b`): The heredoc syntax fails in certain shell environments, causing the script to execute incorrectly.

The Publisher confirms both issues against transcript evidence. The heredoc suggestion is accepted as a general improvement, and the Key Steps are updated to recommend creating an external script file as the primary approach, with the heredoc noted as a fallback.

Evolved skill (8-Executor best-recurrence mean: 98.1). The revised skill resolves the heredoc issue. `gpt-oss-120b` recovers from 40.0 to 100.0 in R1, and five models achieve perfect scores. The remaining variance comes from session-level instabilities (e.g., `gpt-oss-20b` scores 0.0 in R2 due to a timeout) rather than skill content issues. The case illustrates a complementary pattern to the GWS study: here the task is straightforward for most models, but the skill closes the gap for the minority that lack the right procedural knowledge (standard-library statistics) or struggle with the skill's delivery format (heredoc vs. file).

E.2 GWS Email Triage (Extended)

Section 4.4 presents the GWS Email Triage case in the main text. This appendix provides the extended analysis with per-condition transcript evidence and attribution details.

Task environment. The task workspace contains a decoy `gws` binary (a Java E2E tool) and the real `.bin_real/gws` Go binary. The `fws` mock server must be started before any Gmail API calls. Commands use JSON `-params`. Draft creation requires RFC 2822 and base64 wrapping. None of these contracts are visible from the task instruction alone.

No-skill condition (8-Executor mean: 10.0). Most Executors probe `gws -help`, receive the decoy Java tool's help text, and then read local files instead of contacting the mock Gmail API. The resulting triage reports contain plausible but fabricated content with no real inbox reads. Seven of eight Executors score below 15; the exception is `doubao-seed-2.0-mini` (33.3), which partially discovers the real binary but fails to complete the full workflow.

Generic Gmail skill (8-Executor mean: 16.4). A public Gmail skill provides `gog gmail search/get/send` examples. Executors identify a Gmail-like intent but the skill does not cover the local benchmark contract. The mean improves only slightly from 10.0 to 16.4; seven of eight Executors still score below 15, with `gemini-3.1-flash-lite` the sole exception at 54.2. No real Gmail interactions occur for most models.

Original skill R0 (8-Executor mean: 58.4). The Publisher's first skill names the correct binary path (`.bin_real/gws`), documents the `fws` mock server startup workflow, provides the `-params`

JSON syntax, and lists major pitfalls (proxy variables, `userId: me`, RFC 2822 formatting). Five of eight Executors score above 90, but `gpt-oss-20b` (0.0), `gpt-5.4-nano` (8.3), and `gpt-oss-120b` (12.3) still fail—they lose shell state or mishandle the server warmup before draft verification.

Feedback and attribution. Executor feedback reveals two reusable issues:

1. `[bug]` (3 items): The skill does not specify how to verify that `fws` is running before making API calls. Executors attempt Gmail requests before the server is ready, receiving connection-refused errors.

2. `[help]` (5 items): The skill does not document the expected `fws` binary location. Executors spend multiple turns searching for it across `npx`, global `node_modules`, and local paths.

The Publisher confirms both issues against transcript evidence: the original run happened to start `fws` quickly enough, but slower Executors hit the timing issue. The location gap is verified by observing that different environments install `fws` in different paths.

Evolved skill (8-Executor best-recurrence mean: 87.9). The Publisher applies two targeted patches: (1) an explicit `fws` discovery step that searches common installation paths before any Gmail command; (2) a startup verification check that confirms the server responds on the expected port before proceeding. Seven of eight Executors achieve score $\geq$ 80 (five at 100.0), with only `gpt-oss-120b` remaining below the threshold (16.8) due to fundamental session-level limitations. The evolved skill helps turn a brittle hidden integration into a repeatable workflow for the majority of Executors.

F The Use of LLMs

The use of LLMs in the preparation of this manuscript was limited to spell checking and grammar polishing. The core aspects of this work (i.e., research ideation, experimentation, and substantive writing) were conducted by the human authors. Therefore, we confirm that LLMs did not play a significant role and should not be regarded as contributors.