SKILLOPTIMIZER: Agent Skill Optimization Through Subskills Without Task Supervision

Anonymous Author(s)
Affiliation
Address
email

Abstract

Agent skills have become a primary mechanism for extending LLM agent capabilities by providing raw instructions, resources, and domain knowledge. Yet skill quality in the wild is uneven: skills do not always deliver on all the workflows promised by their descriptions, and existing methods for improving skills automatically require downstream task data and verifiers that are often unavailable in practice. We present SKILLOPTIMIZER, a pipeline that improves skills without consulting any downstream task or verifier. SKILLOPTIMIZER decomposes each skill into a directed acyclic graph of subskills, synthesizes coverage-driven prompts from the DAG structure, and runs iterative optimization cycles to improve each subskill. Applied to 192 skills spanning 83 tasks, SKILLOPTIMIZER improves task pass rate over raw baselines across three agent families, establishing that current skills leave coverage value that structured synthetic optimization can recover. [1]

1 Introduction

LLM-powered agents combine reasoning with tool use, memory, and iterative planning [Yao et al., 2022] to tackle complex tasks across software engineering [Jimenez et al., 2023] and web navigation [Zhou et al., 2023a]. A key challenge in deployment is equipping them with the right domain-specific knowledge, e.g., how to use a particular API, navigate a codebase, or follow a workflow. These agents increasingly rely on agent skills [Anthropic, 2025, Jiang et al., 2026]—structured folders of instructions, scripts, and references that encode domain-specific procedural knowledge—as a primary mechanism for extending their capabilities. Public registries host thousands of community-contributed skills, and frameworks such as Claude Code [Anthropic, 2026a] allow agents to install and invoke them directly. Existing skills have the potential to substantially improve agent reliability [Li et al., 2026]: across SKILLSBENCH tasks spanning 11 domains, existing skills associated with the tasks consistently increase performance across harnesses.

A crucial part of a skill is the `SKILL.md` file, which consists of a YAML frontmatter with a required name and description (i.e., what the skill entails and when to invoke it) [Anthropic, 2026d]. This description defines a contract, which is the full scope of what the skill promises to teach. While individual skills can be organized well, there is no guarantee of their quality, nor their *coverage*, i.e., whether their content fully delivers on that contract systematically.

Specifically, Han et al. [2026] find that 39 of 49 SWE skills do not show improvement and Li et al. [2026] find that self-generated skills often do not provide benefits. The bottleneck is not the concept of skills but the depth of their coverage: without a structured way to enumerate what the frontmatter description promises and verify that the content delivers on it, skills are often written inconsistently, with implicit assumptions and gaps between the description and the actual skill.

[1] We release our code here: https://anonymous.4open.science/r/SkillTrainer-2F32/README.md

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

The natural fix is to find downstream tasks addressing the skill and improve the skill automatically against those tasks. However, skills can cover many workflows; the choice of downstream tasks, such as from existing benchmarks, can cause over-optimization in some areas, causing other areas promised by the description to remain stagnant or even become worse. While this benefits those specific tasks, it deprioritizes the general capabilities of the skill. Unlike benchmark settings, real-world tasks rarely have a ground-truth verifier: there is no oracle to tell the system whether the skill improvement helped. Even when some downstream signal is available, optimizing against a narrow task set risks overfitting: the skill becomes more brittle, better for those instances and worse for the broader distribution it is supposed to cover. What is needed is a way to improve skills in the background, systematically, without waiting for downstream tasks, and without a verifier.

We present SKILLOPTIMIZER, a pipeline that improves skills without consulting any downstream task or verifier. SKILLOPTIMIZER first decomposes every skill from its frontmatter description into a directed acyclic graph (DAG) of *subskills*. This decomposition makes coverage a structural property: every workflow implied by the description maps to at least one subskill. SKILLOPTIMIZER then generates synthetic prompts to span the graph and runs optimization cycles in which agents answer the prompts, their answers are graded, and the feedback gets propagated to each relevant subskill in the graph. Applied to 192 skills in SKILLSBENCH, SKILLOPTIMIZER improves task pass rate over raw baselines across three agent families.

We provide the following contributions:

1. A structured skill format centered on a subskill DAG, making coverage of the frontmatter description a measurable structural property and inherent in the design of the skill (§2.1).
2. A synthetic, downstream-free optimization pipeline that improves skill quality through generating synthetic prompts and doing iterative cycles of answering the prompts, grading the answers, and revising the skill.
3. Evidence that SKILLOPTIMIZER improves over raw skills on SKILLSBENCH across three agent families, suggesting that current skills leave coverage value that structured synthetic optimization can recover (§4).

We position SKILLOPTIMIZER as a base layer for background skill improvement: a step toward agents that continuously refine their knowledge during downtime, without requiring a specific task distribution to trigger improvement.

2 The SKILLOPTIMIZER Pipeline

SKILLOPTIMIZER takes a raw skill as input and produces an optimized skill folder ready to be loaded by an agent (Figure 1). The pipeline proceeds in three stages: (1) *Initialization*: decompose the frontmatter into a subskill DAG, initialize the DAG from the raw skill, and build a negative sample; (2) *Prompt Generation*: synthesize coverage-driven prompts from the DAG; (3) *Optimize*: run K atomic cycles of drafting, panel grading, targeted authoring, and validation, then ship the clean skill tree.

2.1 Initialization

The pipeline begins by creating a DAG structure to ground the rest of the skill. We define the raw skill as the input.

Creation of Subskill DAG Template A structure agent reads the frontmatter and decomposes the skill into a directed acyclic graph of *subskills*, where each node represents a discrete, teachable aspect of the domain. The motivation behind this is to continue the practice of progressive disclosure: since skills often encode many workflows, they may be missing a detailed explanation of each one; to fit this deeper explanation without increasing bloat, we need to separate out each part. This decomposition operationalizes the frontmatter contract: every workflow the description implies maps to a path in the DAG, making coverage a structural property rather than an editorial judgment.

The DAG encodes a natural planning order: a node cannot fire until its parents have been addressed, so an agent loading the skill can traverse the graph top-down to build an ordered plan before executing any step. Second, optional activation predicates allow nodes to fire conditionally on task context, enabling the agent to skip irrelevant subskills and concentrate effort on the paths the prompt actually

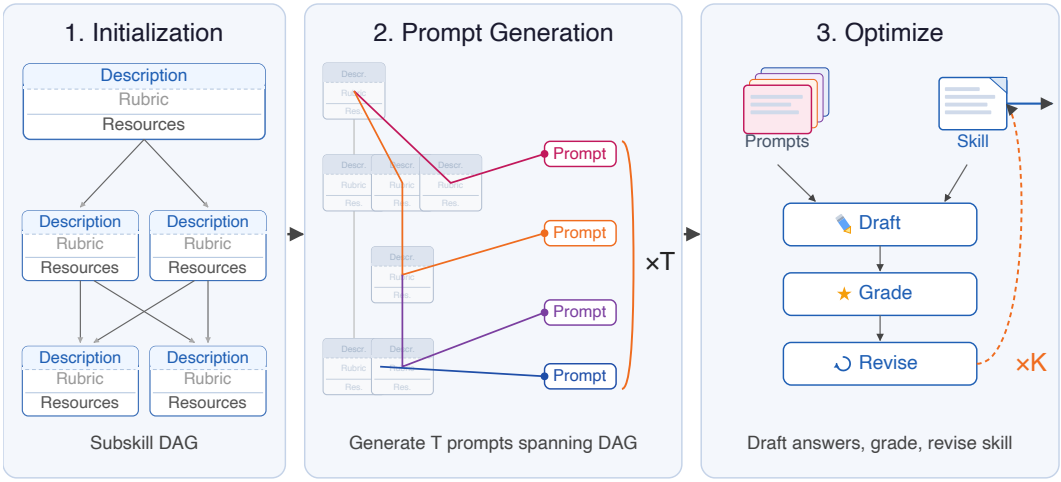


Figure 1: **The SKILLOPTIMIZER pipeline.** *Input*: a raw skill. *(1) Initialization*: decompose into a subskill DAG, initialize the DAG structure from the raw skill, and use it to generate baseline skills. *(2) Prompt Generation*: synthesize T prompts spanning DAG paths, aiming to cover all leaves. *(3) Optimize*: K cycles of drafting against a panel (no-skill, negative sample, raw, prior, current), rubric-grounded panel grading, per-node rewriting, and validator audits. *Output*: a plain skill folder usable by any agent.

activates. Third, artifact nodes, which directly generate a deliverable, carry a RUBRIC: a set of scored criteria anchored at three quality levels. After completing a subskill, the agent can check its output against the RUBRIC and iterate before moving to the next node, turning the skill into a built-in verification scaffold rather than a linear read-once document.

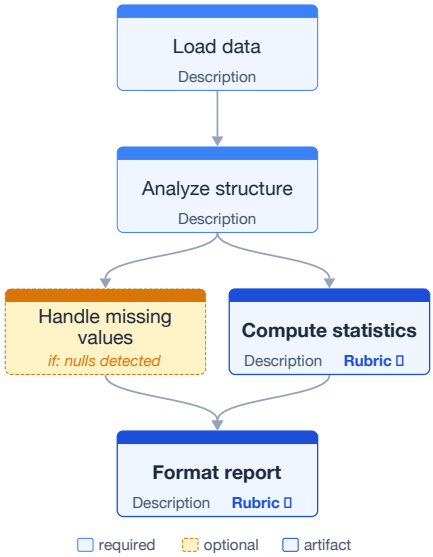
The structure agent produces empty subskill shells and cross-cutting content zones for the root `SKILL.md` but does not author content or write rubrics directly; this is left to future stages. Notably, we fix the structure of the DAG at the beginning of the pipeline, not allowing the optimization process to add or remove nodes.

DAG Initialization Before any optimization cycle, two reference skills are built concurrently and serve as anchors throughout the pipeline. The negative sample is the highest-quality skill a model can produce from the frontmatter and DAG alone, without probes, research, or access to the raw skill, and captures the model-prior baseline we want to escape. Content that merely restates the negative sample is either obvious enough that the model already knew it, or it is the model's wrong default in new vocabulary. Meanwhile, the raw skill is read end-to-end, section spans are routed into the DAG and top-level `SKILL.md` zones, resources are copied with provenance, and source material is preserved at equal-or-stronger force. This gives the optimizer a complete, shippable starting skill rather than an empty scaffold.

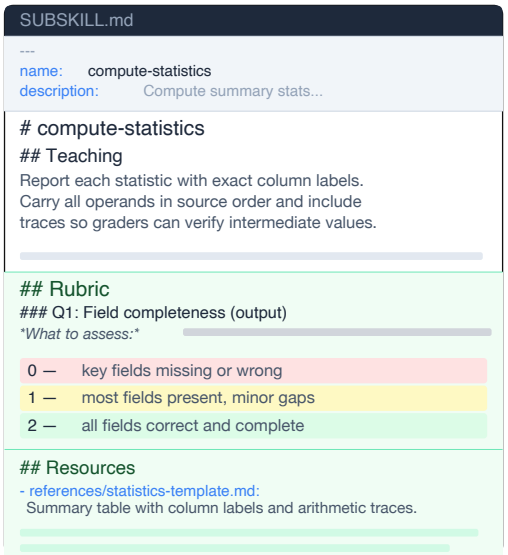
RUBRIC criteria are part of the skill parameters being optimized. The structure phase may declare which nodes need plan-, output-, or verification-shaped rubric questions, but the concrete questions and level descriptions are authored and sharpened through initialization and cycle findings. Non-discriminative criteria—those that produce no spread across panel drafts—are flagged by the validator and passed to the next authoring cycle.

2.2 Prompt Generation

At the start of cycle 1, the orchestrator stamps a seed table of exactly T rows, one per DAG path, pre-assigning diversity axes—domain, user posture, prompt style, and execution complexity—so the prompt set spans the skill's coverage space rather than clustering on typical cases. Min-path-cover rows come first (every node hit at least once, optional activations covered in both states); diversity variants pad to T. Agents then write one realistic end-to-end task per seed row.



(a) Subskill DAG for a data-analysis skill. Required nodes (blue) encode prerequisites; the optional node (amber, dashed) fires only when its activation predicate holds; artifact nodes (dark blue) carry a RUBRIC and ship a deliverable.



(b) An example `SUBSKILL.md`: frontmatter, TEACHING prose, and a RUBRIC question with 0/1/2 scoring levels and a RESOURCES entry.

Figure 2: Subskill DAG and node structure.

A coverage rule is enforced: every required subskill must appear in at least one prompt, and every optional activation must appear in both fired and suppressed states. Paths and baseline drafts are cached after cycle 1 and reused across all cycles, so the prompt distribution is held fixed and cycle-over-cycle changes are attributable to the skill, not the prompts. No prompts are drawn from a downstream benchmark.

2.3 Optimize

SKILLOPTIMIZER runs K optimization cycles, each atomic and evaluating against a frozen copy of the current skill.

Drafting. At cycle 1, four drafts are produced per path: a draft with no skill loaded, a draft loading the negative sample, a draft loading the raw skill, and a draft loading the current optimized skill. At cycles 2+, only the current-cycle draft is re-run; cached baselines are read in place. The previous-cycle draft joins the panel from cycle 2 onward, giving the grader a direct before/after comparison. Each draft's trajectory is persisted and treated as evidence, alongside any plan, verification, and final deliverable the prompt naturally calls for.

Grading. A grader receives the prompt, the RUBRIC criteria, and a role-keyed panel alongside each draft's full execution trajectory. It walks the path in reverse topological order—artifact nodes first, then their prerequisites—scoring every draft against the rubric with quoted evidence. Rankings are in-context panel comparisons; scores are not propagated across cycles to avoid calibration drift. Findings target the current skill only: either a specific line that misled the draft, or a concrete missing slot whose addition would have lifted it. Generic critique without a named target section is not a valid gradient and is rejected.

Revising. For each node with findings or unresolved validator rows, a per-node author rewrites Teaching, RUBRIC questions, and Resources through explicit add, sharpen, or drop edits at named target sections. New material is compared qualitatively against the negative sample to confirm it adds genuine teaching rather than restating the model prior in different words. The raw skill is a positive anchor: edits may extend source material but should not weaken or silently drop it. Web access is

finding-gated: external sources are fetched only when a finding requires knowledge not addressable by internal restructuring or source preservation.

Validation. After authoring, validators audit each touched artifact without reverting edits. They check qualitative negative-sample overlap, panel-ranking goodness (the current skill should beat no-skill, negative sample, previous cycle, and raw skill), non-discriminative rubric questions, prompt-vocabulary leaks, and missing resource provenance. Failures are written into a file and become binding inputs to the next cycle. Thus the optimizer carries failures forward rather than rolling back to a previous snapshot.

After the final cycle, the pipeline finalizes the skill tree. Finalize promotes shared resources, drops resources without references, resolves cross-node conflicts, resolves remaining regressions, runs a hard source-coverage audit in source mode, performs a whole-tree negative-sample sweep, and removes path-specific prompt tokens that leaked into shared content. Frontmatter is preserved byte-for-byte from input through ship. The result is a clean skill folder—`SKILL.md`, subskill files, Dockerfile, and bundled resources—ready to load by any agent, with all build-time logs retained outside the shipped surface.

3 Experimental Setup

3.1 Benchmark and Dataset

We evaluate on SKILLSBENCH [Li et al., 2026], a benchmark where each task ships a deterministic verifier and a per-task skills bundle holding the raw skills the task expects. Tasks span 11 domains, with 1–4+ skills per task. Main results use the full SKILLSBENCH paper set of 83 tasks. Two tasks are excluded per the original SKILLSBENCH paper (`mhc-layer-impl` requires a GPU; `fix-visual-stability` exceeds the verifier timeout), and one additional task is excluded due to requiring heavy setup and memory usage (`fix-druid-loophole-cve`).

For the metric, we report mean task score: the verifier returns a numeric reward $\in [0, 1]$ per trial; we average over all trials for a condition. Each (condition, task, model) triple is evaluated over $n{=}5$ independent trials (415 trials per condition per model).

3.2 Conditions

We evaluate two primary conditions:

- RAW: the agent runs each task with the original SKILLSBENCH skill bundle, unmodified. This is the baseline representing raw skill quality.
- SKILLOPTIMIZER: the agent runs each task with the SKILLOPTIMIZER-optimized skill bundle ($K{=}1$, $T{=}4$, web research on). Because optimized skills use a structured subskill DAG format that current agents are not pre-trained to navigate, a short usage preamble is injected at inference time, describing how to read the subskill structure and apply the RUBRIC criteria. This preamble is fixed and not tuned per task or per run.

3.3 Agent Harnesses and Models

We evaluate across three agent harnesses and three corresponding models: for harnesses, OpenCode [OpenCode, 2025], Codex [OpenAI, 2025], and Claude Code [Anthropic, 2026a]; for models, MiniMax-M2.7 [MiniMax, 2026], GPT-5.4 mini [OpenAI, 2026a], and Claude Sonnet 4.6 [Anthropic, 2026b]. Table 1 lists the harnesses and their pinned versions and inference models. All models were run with their default inference parameters.

Optimization. The SKILLOPTIMIZER optimization pipeline is modeled as an agent skill, allowing for easy invocation. All skills are optimized using Codex (GPT-5.5 [OpenAI, 2026b] as the orchestrator session with medium reasoning; GPT-5.4 [OpenAI, 2026a] and GPT-5.4 mini for subagent roles).

The resulting output is a skill directory with subskills. Optimization is one-time per skill and amortized across all tasks that depend on it.

Table 1: Inference agents used for downstream evaluation and models used by SKILLOPTIMIZER to produce the optimized skill. The optimization pipeline is run in Codex using the same version as in inference.

(a) Inference agents			*(b) Optimization pipeline*	
Harness	**Version**	**Model**	**Role**	**Model**
OpenCode	1.14.35	MiniMax-M2.7	Orchestrator	GPT-5.5
Codex	0.124.0	GPT-5.4 mini	Subagent	GPT-5.4
Claude Code	2.1.128	Claude Sonnet 4.6	Subagent (light)	GPT-5.4 mini

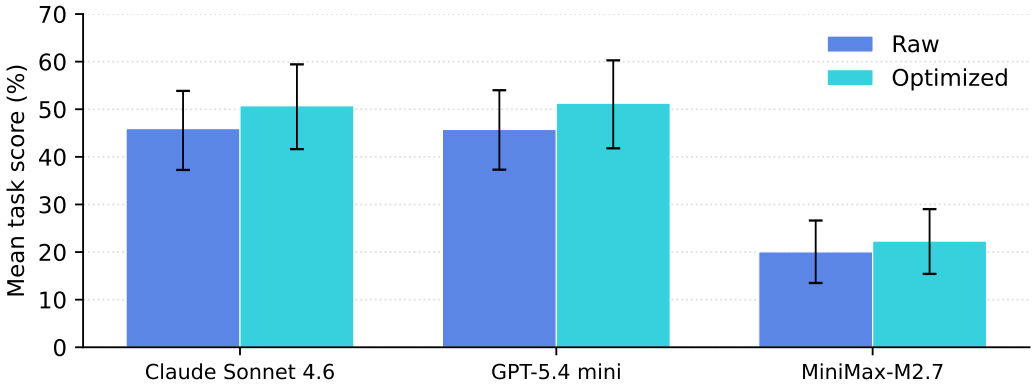


Figure 3: **Mean task score per harness, RAW vs. SKILLOPTIMIZER.** On SKILLSBENCH (83 tasks, n=5 trials), with 95% percentile bootstrap CIs (n=1000 resamples, per-task units). SKILLOPTIMIZER improves all three evaluated harnesses.

4 Results

4.1 Raw vs. Optimized Skills

We evaluate SKILLOPTIMIZER against RAW skills across three agent harnesses on SKILLSBENCH (n=5 trials per task, errors counted as zero). The story across Figures 3–4 and Tables 2–4 is consistent: optimization helps across harnesses, the gain is broad-based across the tasks where skill quality actually matters, and its magnitude is largest on hard tasks and on domains that admit synthetic coverage.

Overall improvement. Figure 3 compares RAW and SKILLOPTIMIZER per harness. SKILLOPTIMIZER improves all three evaluated harnesses: Claude Sonnet 4.6 (+4.8pp), GPT-5.4 mini (+5.5pp), and MiniMax-M2.7 (+2.3pp).

How broadly the gain holds. Figure 4 decomposes per-harness performance into wins, losses, and ties at the task level. Roughly half of all tasks are ties — typically tasks where the harness either fails ($0/5$ in both conditions) or fully solves ($5/5$ in both) regardless of skill version. Among the *decisive* tasks where skill quality actually moves the score, Claude Sonnet 4.6 wins on $27/42$ (64%), GPT-5.4 mini on $27/47$ (57%), and MiniMax-M2.7 on $18/28$ (64%). The improvement from SKILLOPTIMIZER is broad: many small wins rather than a few outsized ones.

By domain. Table 2 breaks the per-task delta down by task domain. Gains concentrate in *Manufacturing* (+17.8pp average across harnesses), *Finance* (+11.7), *Cybersecurity* (+11.4), *Mathematics* (+10.3), and *Robotics* (+6.7). The two negative domains are *Energy* (−4.4) and *Natural Science* (−4.2).

We hypothesize this pattern reflects how amenable each domain is to synthetic optimization. Manufacturing, Finance, Robotics, Cybersecurity, and Mathematics admit relatively self-contained workflows — a scheduling rule, a financial calculation, a planning step, a threat-model checklist, or a derivation

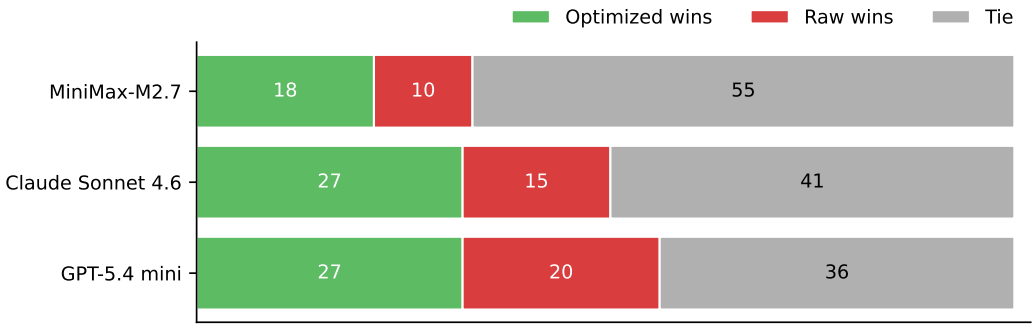


Figure 4: **Wins, losses, and ties per harness across the 83 SKILLSBENCH tasks.** A "tie" is a task where SKILLOPTIMIZER and RAW produce the same per-task mean — usually $0/5$ or $5/5$ across both conditions, where the skill version doesn't matter. Sorted top-to-bottom by wins fraction.

Table 2: Per-domain score delta (SKILLOPTIMIZER − RAW, percentage points). Sorted by mean Δ across harnesses. Claude = Claude Sonnet 4.6, GPT = GPT-5.4 mini, MiniMax = MiniMax-M2.7.

Domain	**Tasks**	**Δ (SKILLOPTIMIZER − RAW, pp)**			**Mean**
		Claude	**GPT**	**MiniMax**	
Manufacturing	3	0.0	+**46.7**	+6.7	+**17.8**
Finance	8	+15.0	+10.0	+10.0	+11.7
Cybersecurity	7	+4.8	+19.1	+**10.4**	+11.4
Mathematics	2	+5.5	+25.5	0.0	+10.3
Robotics	5	+**16.0**	0.0	+4.0	+6.7
Media & Content	11	+6.8	+8.0	+1.2	+5.3
Software Engineering	16	+8.1	+0.1	+2.9	+3.7
Office & White Collar	14	+4.3	+1.4	−2.9	+1.0
Healthcare	2	+1.0	−7.3	+6.3	−0.0
Natural Science	12	−7.7	−1.8	−3.0	−4.2
Energy	3	−6.7	−6.7	0.0	−4.4

— that can be specified declaratively, and synthetic prompts can cover this kind of content faithfully. Natural Science tasks, in contrast, often require ground-truth scientific facts or knowledge of specific datasets, instruments, or analysis pipelines (e.g., gravitational-wave templates, earthquake plate parameters). Optimizing a skill via a synthetic prompt loop without being provided grounded external sources can substitute generic structure for factual specificity, hurting performance.

By difficulty. Table 3 breaks the same delta down by task difficulty. Gains scale monotonically with difficulty: +4.5pp on *hard* tasks, +4.3 on *medium*, and +1.9 on *easy*. This is consistent with optimization being most useful where the raw skill has gaps that bottleneck the agent — on easy tasks the raw skill already suffices.

Table 3: Per-difficulty score delta (SKILLOPTIMIZER − RAW, percentage points). Claude = Claude Sonnet 4.6, GPT = GPT-5.4 mini, MiniMax = MiniMax-M2.7.

Difficulty	**Tasks**	**Δ (SKILLOPTIMIZER − RAW, pp)**			
		Claude	**GPT**	**MiniMax**	**Mean**
Easy	6	+**5.0**	+2.8	−2.2	+1.9
Medium	53	+5.0	+**6.4**	+1.7	+4.3
Hard	24	+4.4	+4.3	+**4.7**	+**4.5**

Largest gains and regressions. Table 4 surfaces the three tasks where SKILLOPTIMIZER helped most and hurt most, averaged across the evaluated harnesses. The same domain pattern holds at the per-task level: two of the top three are Finance tasks (`sec-financial-report`, `weighted-gdp-calc`)

and the third is a planning task (`virtualhome-agent-planning`); two of the bottom three are Natural Science tasks (`earthquake-plate-calculation`, `gravitational-wave-detection`). Synthetic optimization helps where workflows are codifiable and hurts where success requires real-world facts.

Table 4: Top-3 and bottom-3 tasks by avg-over-harnesses Δ (SKILLOPTIMIZER $-$ RAW, percentage points). Top tasks are where SKILLOPTIMIZER helps consistently across the evaluated harnesses; bottom tasks are where it consistently regresses.

Task	**Domain**	**Difficulty**	**Avg Δ (pp)**
Top 3 (SKILLOPTIMIZER helps most):			
`sec-financial-report`	Finance	hard	$+53.3$
`weighted-gdp-calc`	Finance	medium	$+46.7$
`virtualhome-agent-planning`	Robotics	medium	$+40.0$
Bottom 3 (SKILLOPTIMIZER hurts most):			
`earthquake-plate-calculation`	Natural Science	medium	-66.7
`sales-pivot-analysis`	Office & White Collar	medium	-20.0
`gravitational-wave-detection`	Natural Science	medium	-20.0

Cost. SKILLOPTIMIZER adds modest per-trial overhead at eval time — mean wall-clock per trial moves by a single-digit-percent across harnesses and input/output tokens grow 39–55% / 19–34% respectively, against high prompt-cache hit rates ($\approx 99\%$ Claude Sonnet 4.6, $\approx 92\%$ GPT-5.4 mini) that hold the billable delta well below the headline input growth. Optimization itself is one-time per skill (median 35.0 min wall-clock, ≈ 20.8M input / ≈ 444K output / ≈ 223K reasoning tokens, $192/192$ skills succeeded), and amortizes across every task that uses the skill. Full breakdowns are in Appendix D (eval-time, Table 5) and Appendix E (optimization, Table 6).

5 Case Study

To make the mechanism concrete, we examine `sec-financial-report` — the task with the largest gain in Table 4 ($+53.3$pp averaged across harnesses). The task asks four questions about hedge-fund 13F filings, each requiring a different one of three workflow patterns: (Q1, Q2) one-quarter fund summary, (Q3) quarter-to-quarter holdings comparison, (Q4) ranking funds by their holding of one security.

What changed in the skill. The raw `13f-analyzer` skill is a flat 41-line `SKILL.md` that lists three usage examples, one per workflow, then leaves the rest implicit. SKILLOPTIMIZER restructured this into a router subskill (`select-analysis-mode`) and three leaf subskills, one per workflow (Figure 5, left). The root `SKILL.md` shrinks to 30 lines, but each leaf gains 32–54 lines of teaching text and explicit RUBRIC criteria covering the failure modes the raw examples never named — e.g., for `analyze-fund-quarter`: "the printed summary from `one_fund_analysis.py` is the deliverable, not a starting point for your own calculations" and "Keep the script's stock filter and formatting intact."

What that change did to behavior. The script `one_fund_analysis.py` prints two related lines: `Total number of holdings: 3457` (all positions, including bonds and options) and `Number of stock holdings: 2451` (filtered through a `TITLEOFCLASS` allow-list). Question Q2 asks for stocks. Across 5 trials with the raw skill, Claude Sonnet 4.6 ran the right script, observed both lines, and reported 3457 in 4 of 5 trials — in one case the model even noted parenthetically that "2,451 are pure equity/stock holdings" before still answering 3457. The raw `SKILL.md` only said the script "will print out several basic information of a given fund… including total number of holdings, AUM, total number of stock holdings, etc." — the labels "total number of holdings" and "total number of stock holdings" sit one comma apart, which the model effectively conflated, treating the larger 3457 figure as the answer rather than the separately-labelled 2451. With the optimized skill, the same model returned 2451 in 4 of 5 trials. The `analyze-fund-quarter` subskill explicitly tells the agent to treat the printed summary as the deliverable and to keep the script's stock filter intact; this single teaching, surfaced because the workflow now has its own subskill rather than being one of three flat

usage examples, is enough to flip the answer. Aggregated across all three harnesses, this task moves from $6/15$ passing trials with raw skills to $14/15$ passing trials with optimized skills.

6 Related Work

6.1 Agent Skills

Jiang et al. [2026] systematize the agent-skill lifecycle, and [Anthropic, 2026e, Thariq Shihipar, 2026] provide explicit workflows or guidance towards creating better skills. SKILLSBENCH [Li et al., 2026] provides over 84 tasks with deterministic verifiers across 11 domains and establishes that raw skills measurably help agent performance, while self-generated skills do not. SWE-Skills-Bench [Han et al., 2026] reaches a similar conclusion in software engineering. EvoSkill [Alzubi et al., 2026] and CoEvoSkills [Zhang et al., 2026] improve skills through iterative failure analysis against benchmark instances. Tan et al. [2026] learn repository-specific skill files for coding agents using execution-grounded feedback, achieving large gains on per-repository tasks. These methods are powerful but all share the same constraint: they require a downstream task distribution with verifiers: skills are evolved against pass/fail signal from real instances. SKILLOPTIMIZER differs: it improves skills using only synthetic prompts derived from the skill's own structure, without any downstream task or verifier, enabling optimization to run in the background. Concurrent work has also attempted skill optimization without task-specific supervision [Tian et al., 2026, Maple, 2026]. However, we ground our optimization in a DAG allowing for better coverage over the contract made by the skill description, and more structured prompt generation and optimization.

6.2 Automatic Instruction Optimization

A substantial line of work studies automatic prompt and instruction optimization as search over discrete, human-readable text artifacts rather than gradient-based weight updates. APE [Zhou et al., 2023b] induces instructions through black-box search over LLM-proposed candidates; OPRO [Yang et al., 2024] conditions on prior candidate-score pairs to elicit stronger ones. EvoPrompt [Guo et al., 2024] and Promptbreeder [Fernando et al., 2024] bring evolutionary search to prompt optimization through iterative mutation and selection. DSPy [Khattab et al., 2024] and MIPRO [Opsahl-Ong et al., 2024] extend this to modular LM programs, jointly optimizing instructions and few-shot demonstrations. A closely related thread folds critique and revision into the loop itself: Self-Refine [Madaan et al., 2023] alternates generation, feedback, and revision with a single model; Reflexion [Shinn et al., 2023] stores linguistic reflections in episodic memory to improve future trials; GEPA [Agrawal et al., 2026] combines natural-language diagnosis with Pareto-guided search over candidate prompts. SKILLOPTIMIZER's optimization cycles instantiate this class on a structured, decomposed skill document rather than a flat prompt, targeting criteria-fit on synthetic prompts rather than scalar reward from a task, making it viable without any downstream signal.

7 Conclusion

We present SKILLOPTIMIZER, a pipeline for improving agent skills without downstream supervision. By decomposing each skill's frontmatter into a subskill DAG, then running iterative optimization cycles with comparative grading, SKILLOPTIMIZER recovers coverage value that raw skills leave on the table, as shown by improvements over raw baselines on SKILLSBENCH across three agent families. No downstream task data or verifier is consulted during optimization.

Future work can extend SKILLOPTIMIZER to be used as a base layer for background skill improvement: agents can optimize their skill coverage during downtime, without waiting for a specific task to trigger improvement, and without overfitting to any particular task distribution. We see this as a first step toward agents that continuously improve their skill coverage over time, analogous to how a practitioner reviews and deepens their knowledge outside of active work.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen,

Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated Skill Discovery for Multi-Agent Systems, March 2026. URL http://arxiv.org/abs/2603.02766. arXiv:2603.02766 [cs].

Anthropic. Equipping agents for the real world with Agent Skills, October 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Anthropic. Claude code overview. https://code.claude.com/docs, 2026a. Official Claude Code documentation. Accessed May 06, 2026.

Anthropic. System card: Claude sonnet 4.6. https://www-cdn.anthropic.com/bbd8ef16d70b7a1665f14f306ee88b53f686aa75.pdf, 2026b. Anthropic system card. Accessed May 06, 2026.

Anthropic. Claude code routines. https://code.claude.com/docs/en/routines, 2026c. Official Claude Code documentation. Accessed May 06, 2026.

Anthropic. Specification. https://agentskills.io/specification, 2026d. Accessed: 2026-05-08.

Anthropic. Skill Creator. https://github.com/anthropics/skills/blob/main/skills/skill-creator, 2026e. GitHub repository file: anthropics/skills, path skills/skill-creator/SKILL.md.

Cursor. Cursor cloud agents. https://cursor.com/docs/cloud-agent, 2026. Official Cursor documentation. Accessed May 06, 2026.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 13481–13544. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/fernando24a.html.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=ZG3RaNIsO8.

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

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. SoK: Agentic Skills – Beyond Tool Use in LLM Agents, February 2026. URL https://arxiv.org/abs/2602.20867v1.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *CoRR*, abs/2310.06770, 2023. doi: 10.48550/ARXIV.2310.06770. URL https://doi.org/10.48550/arXiv.2310.06770.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Letta. Sleep-time agents. https://docs.letta.com/guides/agents/architectures/sleep time/, 2025. Official Letta documentation. Accessed May 06, 2026.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han-chung Lee. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks, March 2026. URL http://arxiv.org/abs/2602.12670. arXiv:2602.12670 [cs].

Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E. Gonzalez. Sleep-time compute: Beyond inference scaling at test-time. *CoRR*, abs/2504.13171, 2025. doi: 10.48550/ARXIV.2504.13171. URL https://doi.org/10.48550/arXiv.2504.13171.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Simon Maple. Stop guessing whether your Skill works: skill-optimizer measures and improves it, April 2026. URL https://tessl.io/blog/stop-guessing-whether-your-skill-works-skill-optimizer-measures-and-improves-it/.

MiniMax. MiniMax M2-7. https://www.minimax.io/news/minimax-m27-en, 2026. Official MiniMax blog post. Accessed May 06, 2026.

Nous Research. Hermes agent. https://github.com/nousresearch/hermes-agent, 2026. Open-source always-on AI agent framework. MIT License.

OpenAI. Chatgpt codex. https://chatgpt.com/codex, 2025. Official Codex product page. Accessed May 06, 2026.

OpenAI. GPT-5.4 thinking system card. https://deploymentsafety.openai.com/gpt-5-4-thinking/gpt-5-4-thinking.pdf, 2026a. OpenAI system card. Accessed May 06, 2026.

OpenAI. GPT-5.5 system card. https://deploymentsafety.openai.com/gpt-5-5/gpt-5-5.pdf, 2026b. OpenAI system card. Accessed May 06, 2026.

OpenCode. OpenCode. https://opencode.ai/, 2025. Official product website. Accessed May 06, 2026.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 9340–9366. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.525. URL https://doi.org/10.18653/v1/2024.emnlp-main.525.

Replit. Introducing agent 3: Our most autonomous agent yet. https://blog.replit.com/introducing-agent-3-our-most-autonomous-agent-yet, 2025. Replit blog post. Accessed May 06, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann,

Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314, 2024. doi: 10.48550/ARXIV.2408.03314. URL https://doi.org/10.48550/arXiv.2408.03314.

Peter Steinberger and OpenClaw Contributors. OpenClaw: Personal AI assistant. https://github.com/openclaw/openclaw, 2026. Open-source multi-channel AI assistant gateway. MIT License.

Shangyin Tan, Lakshya A. Agrawal, Rohit Sandadi, Dan Klein, Koushik Sen, Alexandros G. Dimakis, and Matei Zaharia. Automatically Learning Skills for Coding Agents. Technical report, UC Berkeley, February 2026. URL https://gepa-ai.github.io/gepa/blog/2026/02/18/automatically-learning-skills-for-coding-agents/.

Thariq Shihipar. Lessons from Building Claude Code: How We Use Skills, March 2026. URL https://x.com/trq212/status/2033949937936085378.

Yu Tian, Jiawei Chen, Lifan Zheng, Mingxiang Tao, Xinyi Zeng, Zhaoxia Yin, Hang Su, and Xian Sun. Skills-Coach: A Self-Evolving Skill Optimizer via Training-Free GRPO, April 2026. URL http://arxiv.org/abs/2604.27488. arXiv:2604.27488 [cs].

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *CoRR*, abs/2210.03629, 2022. doi: 10.48550/ARXIV.2210.03629. URL https://doi.org/10.48550/arXiv.2210.03629.

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. Co-EvoSkills: Self-Evolving Agent Skills via Co-Evolutionary Verification, April 2026. URL http://arxiv.org/abs/2604.01687. arXiv:2604.01687 [cs].

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *CoRR*, abs/2307.13854, 2023a. doi: 10.48550/ARXIV.2307.13854. URL https://doi.org/10.48550/arXiv.2307.13854.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023b. URL https://openreview.net/forum?id=92gvk82DE-.

A Limitations

SKILLOPTIMIZER uses synthetic prompt generation, so is limited in scope to what it can improve on, e.g., related to live, stateful environments. Additionally, a core foundation is that the frontmatter is treated as an accurate contract. If the frontmatter is incomplete or misleading, the subskill DAG inherits those gaps and the optimized skill reflects them. Improving frontmatter generation is orthogonal and is left to future work. To maximize usability, we modeled SKILLOPTIMIZER as a skill itself, meaning an orchestrator is responsible for executing the entire pipeline as described. While this allows for more flexibility, it also allows certain parts of the pipeline to be ignored or underspecified. One other aspect is that there are a set of hyperparameters in SKILLOPTIMIZER that are taken as fixed. Changing these hyperparameters would give a better sense of the stability and impact of the method. Additionally, due to the cost of SkillsBench, extensive model evaluation is limited; we hope to run through more models to discover when the optimized skills help—and possibly harm—performance. Finally, GPT-5.5 in Codex was used as the orchestrator for optimizing skills using SKILLOPTIMIZER, so using another model backend or harness could affect the optimized skill quality. Additionally, one model used for inference is GPT-5.4 mini, which is also used as a subagent during optimization so could gain a higher score relative to other models. This however we believe also showcases the method in a realistic sense, as people are likely to optimize skills on models they are actually using.

B Ethics Statement

LLMs were used during paper drafting, prototyping, the optimization pipeline itself, and for running experiments. All evaluated skills come from SkillsBench [Li et al., 2026], an already publicly available dataset. The synthetic prompt sets generated during optimization are derived from skill definitions and external research; no proprietary or private data is consumed.

C Additional Related Work

Scaling test-time compute optimally has emerged as a standard lever for improving LLM performance without scaling model parameters [Snell et al., 2024]. Sleep-time compute [Lin et al., 2025] extends this idea by decoupling reasoning from the query: rather than spending compute only when a user asks a question, the model pre-processes available context while idle, producing an enriched representation that makes test-time reasoning cheaper. This motivates a broader class of approaches that do useful work during downtime. For example, Letta's sleep-time agents [Letta, 2025] run memory consolidation asynchronously between conversations, while Hermes Agent [Nous Research, 2026], OpenClaw [Steinberger and OpenClaw Contributors, 2026], Claude Code Routines [Anthropic, 2026c], Cursor cloud agents [Cursor, 2026], and Replit's Max Autonomy [Replit, 2025] all run extended autonomous tasks in the background via cron-style triggers, continuing without user input until completion. Hermes Agent [Nous Research, 2026] goes a step further — after complex tasks it autonomously writes structured skill documents in the background, and a periodic curator consolidates and prunes them. We see SKILLOPTIMIZER as being one of the possible workflows a background agent could commit to doing to expand model capabilities outside of explicit user-defined tasks.

D Eval-time Cost

SKILLOPTIMIZER adds extra subskill files to the agent's context, which inflates per-trial input tokens. Across the two harnesses that report token counts (Claude Sonnet 4.6 via Claude Code, GPT-5.4 mini via Codex), mean input tokens per trial grow 39–55% from RAW to SKILLOPTIMIZER, and output tokens grow 19–34% (Table 5). The billable delta is much smaller: prompt-cache hit rates are high ($\approx 99\%$ for Claude Sonnet 4.6, $\approx 92\%$ for GPT-5.4 mini) in both conditions, so uncached input is only a small fraction of the input column. Wall-clock per trial is essentially flat: Claude Sonnet 4.6 $+1.2$ min, GPT-5.4 mini $+1.0$ min, MiniMax-M2.7 $+0.1$ min. MiniMax-M2.7 runs through the OpenCode harness, which does not surface token counts; only wall-clock is available there.

Table 5: **Per-trial cost: RAW vs. SKILLOPTIMIZER.** Mean per-trial agent wall-clock and input/output tokens across all SKILLSBENCH trials. Input tokens are harness-emitted totals including prompt-cache hits; cache-hit rates vary by harness, so uncached input is a smaller fraction of the headline column. The MiniMax-M2.7 harness does not surface token counts; only wall-clock is available.

	Wall-clock / trial		Input tokens / trial		Output tokens / trial	
Harness	RAW	SKILLOPTIMIZER	RAW	SKILLOPTIMIZER	RAW	SKILLOPTIMIZER
Claude Sonnet 4.6	7.4 min	8.6 min	1.17M	1.81M	39.2K	52.7K
GPT-5.4 mini	4.9 min	5.9 min	1.47M	2.04M	23.3K	27.7K
MiniMax-M2.7	6.9 min	7.0 min	–	–	–	–

E Optimization Cost Breakdown

A single SKILLOPTIMIZER optimization run with default knobs (K=1 cycles, T=4 prompts, web research on) takes a median of 35.0 min wall-clock in container and runs a median of 42 sub-agent rollouts (drafter, grader, and author waves $\times$ prompts $\times$ cycles plus init/validation; mean 41.5, $\sigma = 9.3$, p10–p90 36–49, range 5–82). It consumes $\approx$ 20.8M input tokens ($\approx$ 92% cache-hit), $\approx$ 444K output tokens, and $\approx$ 223K reasoning tokens per skill (Table 6).

Table 6: **Optimization cost per skill** (K=1, T=4, web research on). Aggregated over the 192 skills. Wall-clock is end-to-end container time; tokens are summed across the subagents that one optimization run uses (drafters, graders, authors, init/validation). Optimization is one-time per skill and amortizes across every task that uses it.

	Mean	**Median**	**p90**
Wall-clock per skill	35.1 min	35.0 min	46.9 min
Input tokens (incl. cache)	20.82M	—	—
Cache-hit tokens	19.21M	—	—
Output tokens	444.5K	—	—
Reasoning tokens	223.1K	—	—

Amortisation. Optimization is one-time per *skill*, not per task, so its cost amortizes across every downstream task that uses the skill. As a reference: optimizing one skill consumes roughly 10$\times$ the input tokens of one SKILLOPTIMIZER eval trial under Claude Sonnet 4.6, and $\approx$ 8$\times$ the output tokens. There are high-leverage skills in SKILLSBENCH (`xlsx`, `pdf`, `docx`) that each cover multiple tasks (though there may be different versions of skills with the same name); for a skill that covers many tasks evaluated by many harnesses, the per-task amortized optimization cost shrinks, which is the intention of our pipeline and its main use during inference, outside of the benchmarking scenario where each set of skills is only applied to one or few tasks.

F Case Study

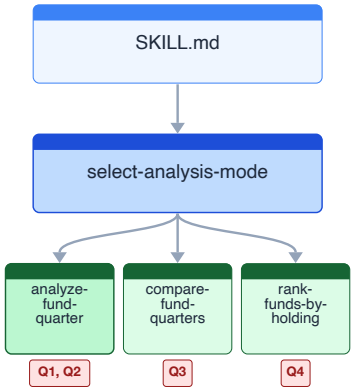
We include a visualization for the case study in Figure 5.

G SKILLOPTIMIZER Algorithm

The pseudocode for SKILLOPTIMIZER is in Eqn 1.

H Additional Results

We also report results with Gemini 3 Flash (Gemini CLI 0.40.1) included as a fourth model and harness. Gemini 3 Flash performs the best out of all the harnesses with the raw skill: its raw-condition score on SKILLSBENCH is the highest of any harness we evaluated, so the headroom for



(a) Optimized `13f-analyzer`: the flat `SKILL.md` becomes a router plus three workflow leaves; Q-tags show the question routing.

```
[Read] 13f-analyzer/SKILL.md

[Read] select-analysis-mode/optional/
  analyze-fund-quarter/scripts/one_fund_analysis.py

[Bash] python3 scripts/one_fund_analysis.py
  --accession_number 0001037389-25-000064 --quarter
2025-q3

[stdout]
Total number of holdings:   3457
Total AUM:                  75753182516.00
Number of stock holdings:   2451
Total stock AUM:            59538855690.00

[answer] q2_answer = 2451
```

(b) Agent trajectory (Claude Sonnet 4.6, optimized skill; Q2): the agent routes to `analyze-fund-quarter`, runs the bundled script, reads both counts, and returns `Number of stock holdings`.

Figure 5: **Case study:** `sec-financial-report`**.** With the raw skill, Claude Sonnet 4.6 answered incorrectly in 4 of 5 trials (reporting 3,457 instead of 2,451); with the optimized skill it answered correctly in 5 of 5 trials.

Algorithm 1 SKILLOPTIMIZER(S, K, T, W)

Require: Raw skill S with frontmatter F and body B; cycle count K; prompt count T; web flag W
Ensure: Optimized skill tree S^*

Stage 1 — Initialization
$(G, Z) \leftarrow$ DESIGNSTRUCTURE(F, S) ▷ DAG nodes + top-level skill zones; preserve frontmatter
concurrently
$N \leftarrow$ GENERATENEGATIVESAMPLE(F, G) ▷ source-blind model-prior skill
$S^{(0)} \leftarrow$ INITIALIZEFROMRAWSKILL(S, G, Z) ▷ route raw content into a shippable initialized skill
end concurrently
if $K = 0$ **then return** FINALIZE$(S^{(0)}, N, S)$
end if

Stage 2 — Prompt Generation *(fixed across all cycles)*
$P \leftarrow$ GENERATEPROMPTS(G, T, S, W) ▷ min-path-cover + diversity axes
$\{D_\emptyset,\ D_N,\ D_{\text{raw}}\} \leftarrow$ CACHEBASELINEDRAFTS(P, N, S) ▷ cycle-1 only

Stage 3 — Optimize
for $k = 1, \ldots, K$ **do**
  $\widetilde{S}^{(k-1)} \leftarrow$ FREEZEFORCYCLE$(S^{(k-1)})$ ▷ fixed skill version for all paths in this cycle
  $D^{(k)} \leftarrow$ DRAFTANSWERS$(P, \widetilde{S}^{(k-1)})$ ▷ one current-skill draft per path
  $\Pi^{(k)} \leftarrow$ BUILDCOMPARISONPANEL$(D^{(k)}, D_\emptyset, D_N, D_{\text{raw}}, D^{(k-1)})$ ▷ include previous cycle when $k > 1$
  $\mathcal{F}^{(k)} \leftarrow$ GRADEPANEL$(\Pi^{(k)}, G, P)$ ▷ reverse-DAG rubric grading + per-node findings
  for each node $v \in G$ with findings or open regressions **do**
    $S^{(k)}[v] \leftarrow$ REVISESUBSKILL$(S^{(k-1)}[v], \mathcal{F}^{(k)}[v], N[v], S[v], W)$
  end for
  $S^{(k)}[\texttt{SKILL.md}] \leftarrow$ UPDATETOPLEVELSKILL$(S^{(k-1)}, \mathcal{F}^{(k)}, N, S, W)$
  VALIDATECYCLE$(S^{(k)}, N, S, \mathcal{F}^{(k)}, P)$ ▷ negative-sample, ranking, rubric, leak, provenance audits
end for
return FINALIZE$(S^{(K)}, N, S)$ ▷ provenance gate + source coverage + generalization sweep

SKILLOPTIMIZER appears smaller and the structured subskill DAG that the other harnesses navigate appears to be a poorer fit for it. The aggregate paired delta is -5.5pp (Table 7), but the change is not uniformly negative: SKILLOPTIMIZER improves Gemini 3 Flash on 18 tasks and regresses on 27 (with 38 ties), giving a wins/decisive fraction of $18/45 = 40\%$. The largest per-task gains and regressions are somewhat mixed; office and some software engineering tasks gain, Natural Science and a software engineering task regress. Here, the regression is much larger in absolute value than the gains, unlike in the main results, where it is more even, suggesting more extreme collapses for Gemini 3 Flash (Table 8).

Table 7: **Main results on SKILLSBENCH (83 tasks, n=5 trials).** Mean task score (%) for RAW vs. SKILLOPTIMIZER per harness. Δ is the *paired* (per-task) delta. **Wins / decisive** is the count of tasks where SKILLOPTIMIZER beats RAW out of tasks with a non-zero delta (sign-test convention; ties typically reflect 0/5=0/5 or 5/5=5/5 — uninformative).

Harness	RAW (%)	SKILLOPTIMIZER (%)	Δ paired (pp)	Wins / decisive
Claude Sonnet 4.6	45.9	50.7	+4.8	27/42 (64%)
GPT-5.4 mini	45.8	51.3	+5.5	27/47 (57%)
Gemini 3 Flash	54.1	48.6	−5.5	18/45 (40%)
MiniMax-M2.7	20.0	22.3	+2.3	18/28 (64%)

Table 8: Top-3 and bottom-3 tasks by per-task Δ (SKILLOPTIMIZER − RAW, percentage points). Top tasks are where SKILLOPTIMIZER helps most for ; bottom tasks are where it regresses most.

Task	Domain	Difficulty	Avg Δ (pp)
Top 3 (SKILLOPTIMIZER helps most):			
`trend-anomaly-causal-inference`	Software Engineering	hard	+42.4
`fix-build-agentops`	Software Engineering	easy	+40.0
`organize-messy-files`	Office & White Collar	medium	+40.0
Bottom 3 (SKILLOPTIMIZER hurts most):			
`flood-risk-analysis`	Natural Science	medium	−100.0
`gravitational-wave-detection`	Natural Science	medium	−80.0
`fix-build-google-auto`	Software Engineering	easy	−80.0

Why `flood-risk-analysis` regresses (-100pp). One of the largest single regressions illustrates the inverse of the `sec-financial-report` case. The raw `nws-flood-thresholds` `SKILL.md` inlines a short `urllib` recipe that Gemini 3 Flash pastes verbatim and passes $5/5$. SKILLOPTIMIZER restructured the skill to route through a bundled retrieval script and removed the inline code; in our runs the script's network call failed, and with no inline fallback in the `SKILL.md`, the agent drifted into hand-coding flood thresholds and got every trial wrong. Optimization replaced a recipe the agent was leaning on with an abstraction it couldn't recover from when the script path broke.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction enumerate three concrete contributions (subskill DAG format, synthetic optimization pipeline, and SkillsBench evaluation), each delivered in the corresponding paper section.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated Limitations section discusses the limitations.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.

- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper presents no theoretical results; all contributions are empirical.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: §3 specifies the benchmark, evaluation metric, pinned harness and model versions, optimization parameters (K=1, T=4), number of trials per condition, and excluded tasks; the optimization pipeline will be released as a runnable skill.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.

- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The SKILLOPTIMIZER pipeline code and optimized skill artifacts are released publicly, see https://anonymous.4open.science/r/SkillTrainer-2F32/README.md.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: §3 reports benchmark details, conditions, agent harnesses with pinned versions, optimization model roles, the evaluation metric, and n=5 trials per condition—all information necessary to understand the results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Main result figures include 95% percentile bootstrap confidence intervals (n=1000 resamples, per-task units) computed from n=5 independent trials per (condition, task, model) triple.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Appendix D, E reports the compute used for optimization (one-time per skill) and inference evaluation (API calls to hosted models), including estimated token counts and wall-clock time.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The research involves only automated skill optimization and evaluation on existing benchmarks, this is discussed in an Ethics Statement in the appendix.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: SKILLOPTIMIZER makes agent capabilities more accessible by improving skill quality without expensive downstream data collection. Negatively, as a general optimization technique it inherits the harmful-use potential of the agent systems it improves: a more capable agent skill could equally improve performance on harmful workflows, and this risk is not mitigated by the pipeline itself.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The optimized skills will be released, and are structured text/code skill files; no models or scraped datasets are released, and the artifacts pose no largely elevated misuse risk beyond the agent systems they are designed for and their existing raw reference skills in SkillsBench.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All external assets are cited: the benchmark SKILLSBENCH [Li et al., 2026] and the models used are cited where introduced.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The SKILLOPTIMIZER pipeline (runnable skill) is released with a README and usage instructions in the supplemental material. The optimized skills will be released upon acceptance.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human participants; all experiments are fully automated.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve human subjects; no IRB approval is required.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are a core, original component of SKILLOPTIMIZER: they are responsible for executing the entire optimization pipeline (decomposition, drafting, grading, rewriting) and were also used to manage experiments and help develop and iterate on the methodology itself. This is also discussed in the Ethics Statement.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.