VERIHARNESS: SCALING AGENTIC VERIFICATION FOR LONG-HORIZON TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

As LLM agents undertake increasingly complex, long-horizon tasks, verifying their outputs becomes increasingly challenging. We study how verification capability can be strengthened with a fixed base model, without access to reference answers or grading rubrics at test time. Repeated sampling yields multiple rollouts that can contain complementary correct claims, but we need a reliable verification mechanism to determine which claims to trust. We first find that disagreement often exposes correct alternatives, while consensus can conceal errors. These observations motivate VERIHARNESS, which turns the underlying LLM a generator uses into an agentic verifier by giving it a workspace, evidence tools, and reusable verification skills. A disagreement resolver checks competing claims against environmental evidence, while a consensus challenger tests shared claims and searches for omitted requirements. Their findings guide the selection and revision of the final artifact. Across five long-horizon workspace benchmarks and two frontier models, VERIHARNESS achieves the highest selection scores among the evaluated baselines. Evidence-backed revision further improves average performance, bringing gains over a single rollout to 6.2 points with Gemini 3.5 Flash and 6.4 points with Claude Opus 4.8. We further show that verification skills can self-improve from failure feedback, demonstrating VERIHARNESS as a novel and critical approach for scaling long-horizon agentic verification.

1 INTRODUCTION

Long-horizon agents increasingly produce reports, spreadsheets, and other artifacts whose correctness requires substantial work to assess (Vidgen et al., 2026; Tang et al., 2026; Tencent WorkBuddy Bench Team et al., 2026). Recent workspace benchmarks show that even frontier models struggle to produce reliable artifacts (Zhu et al., 2026; Li et al., 2026b). Verifying these artifacts is harder still. In coding and mathematics, unit tests, executable checks, or formal proof checkers give direct feedback on correctness even when no reference answer exists (Ehrlich et al., 2025; Li et al., 2025). General complex tasks rarely offer an equally straightforward way to assess the entire output. For example, checking a report may require tracing claims to sources, recomputing quantities, and interpreting requirements across several files. As agents take on more of this work, verification becomes a central challenge and must develop alongside generation (You et al., 2026).

We study test-time verification without a stronger judge: the generator and verifier share the same model and tools, and the verifier checks the output artifacts without access to reference answers or grading rubrics. This setting reflects practice at the frontier, where the generator is already the strongest available model and no stronger judge exists to oversee it (Bowman et al., 2022; Burns et al., 2024). Holding the base model fixed also ensures that no knowledge from a stronger model enters the verifier: any gain comes from the model's own rollouts and environment (Huang et al., 2023). In this setting, the verifier draws on two unique sources of information: 1) the candidate rollouts and 2) the task environment. Repeated sampling yields several rollouts of the same task (Brown et al., 2024; Zhu et al., 2025), and comparing them reveals competing claims and shared conclusions. The task environment provides source files, data, and constraints against which those claims can be checked.

Existing test-time verification methods mostly use only the rollouts. Majority voting and self-consistency (Wang et al., 2023; Chen et al., 2023) treat the most common answer as correct, and LLM-as-a-judge approaches (Zheng et al., 2023; Kwok et al., 2026) score rollouts by reading

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

them. Both rely on what the model already believes rather than on evidence the environment could provide. To address this shortcoming, we first propose to bring verification into the environment: individual claims are tested against the workspace's source data and constraints. Second, we find that **consensus does not guarantee correctness, while disagreement between rollouts can expose correct alternatives.** Figure 1 shows that roughly a third of the agreed values are judged incorrect, so errors cannot be corrected by voting or selection alone. Also, disputed claims often include a correct candidate, although the most frequent answer may be wrong (Section 2). Checking these alternatives against evidence can help identify the correct value. Therefore, a verifier has to test competing answers and challenge the consensus that all candidates share.

We introduce VERIHARNESS, a general-purpose, plug-and-play harness for the verifier, which uses the same underlying model as the generator. The harness encompasses a workspace, evidence tools, and reusable verification skills to support these two checking tasks. A *disagreement resolver* tests competing claims against source files, data, and task constraints. A *consensus challenger* searches for evidence that could refute consensus values or reveal omitted requirements. Their findings guide the selection and revision of the final artifact, accompanied by a record of the supporting evidence. The verifier decides what to check, how to interpret the evidence, and which changes to make. We further ask whether the harness can grow its verification skills with experience. Building on reusable skill and experience libraries (Wang et al., 2024; Ouyang et al., 2026), we let the harness accumulate skills from failure feedback on development tasks, starting from either an empty library or a human-authored one.

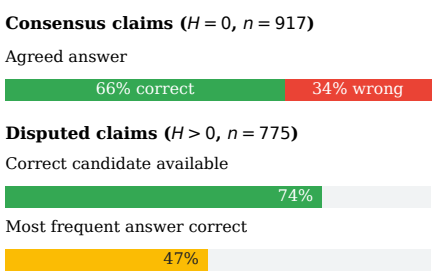

Figure 1: **Consensus can preserve errors; disagreement can contain correct alternatives that voting discards.** APEX-Agents claims from ten-rollout pools of Claude Opus 4.8; percentages are within each group, correctness uses the benchmark's rubric grades, and H is the claim entropy of Section 2.

We evaluate on five demanding workspace benchmarks with two frontier models, Gemini 3.5 Flash and Claude Opus 4.8, each serving as both generator and verifier. VERIHARNESS achieves the highest selection score among the evaluated baselines on every benchmark with both models, including LLM-as-a-Verifier (Kwok et al., 2026). Evidence-backed revision further improves average performance, yielding gains over a single rollout of 6.2 points with Flash and 6.4 with Opus (Table 1). On held-out tasks, skills evolved from an empty library outperform the human-authored library, and evolving from the human-authored library adds 6.8 points on APEX-Agents and 3.7 on SpreadsheetBench 2 over that library (Section 5).

Our contributions are threefold: (1) **Verification principle.** We identify two complementary checking tasks in same-model verification: resolving disputed claims and challenging consensus claims against evidence from the environment. (2) **General-purpose harness.** We build VERIHARNESS, the first agentic verification harness for long-horizon tasks, which is training-free and plug-and-play across benchmarks and models, and evaluate its ability to select and revise agent outputs. (3) **Evolving verification skills.** We show that a fixed model can accumulate useful verification skills from failure feedback, with human expertise providing an effective starting point for evolution.

2 PROBLEM FORMULATION

Task and rollouts. A task $t = (s, \mathcal{E})$ consists of a description s and an environment $\mathcal{E}$ containing source files, data, and application state. The *generator* is an agent built on model M that attempts the task. Each attempt produces a *rollout* $r_i = (d_i, w_i, \tau_i)$, comprising a delivered artifact d_i, final workspace state w_i, and action trace τ_i, and repeated sampling yields a pool $R = \{r_1, \ldots, r_N\}$. An artifact makes *claims*: assertions about task-relevant values, interpretations, or the satisfaction of requirements, such as a reported figure, the reading of a clause, or a required section.

Verification objective. The *verifier* uses the same model and tools as the generator. Given the task and the pool, it delivers a rollout $r^\star$, selected from R or revised from one of its members, together with a record C of checks, findings, and unresolved issues. Its objective is to maximize the quality of

$r^\star$ as judged by an external grader. The verifier has no access to this grader, to reference answers, or to grading rubrics, and all compared methods receive the same rollout pool.

Checks and evidence. We examine artifacts at the level of claims. A *check* χ is an operation that tests a claim, such as reading a source passage, recomputing a total, or inspecting file metadata, and its result $e = \mathcal{E}(\chi)$ is the evidence obtained by running the check in the environment. For example, "FY2025 revenue is EUR120m" is a claim, and inspecting the final report for the amount and currency is a check. Evidence acquisition is *active*: the verifier chooses each check from the task, the candidate pool, and the evidence gathered so far. An informative check helps distinguish candidate answers or test a constraint that the current answer may violate. Evidence can support a candidate, contradict it, or establish a correction absent from the pool, and a claim remains unresolved when the evidence is insufficient for a verdict. Task requirements also identify claims to examine when the corresponding content is absent from every artifact.

Consensus and disputed claims. Let p identify a common question or requirement, and let $v_{i,p}$ denote the answer or content supplied by artifact d_i, treating equivalent expressions as the same value and omission as a distinct value. The candidate set $\mathcal{V}_p = \{v_{i,p}\}_{i=1}^{N}$ has empirical distribution and entropy

$$\pi_p(v) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[v_{i,p} = v], \qquad H(\pi_p) = -\sum_{v \in \mathcal{V}_p} \pi_p(v) \log \pi_p(v). \tag{1}$$

A claim is a *consensus* claim when $H(\pi_p) = 0$ and a *disputed* claim when $H(\pi_p) > 0$. The entropy measures variation across candidates only: unanimity can hold even when every candidate is incorrect.

Empirical observations. On ten-rollout pools of Claude Opus 4.8 on APEX-Agents, whose rubrics permit claim-level grading (details in Appendix A), 34% of consensus values are judged incorrect, while 74% of disputed claims contain a correct candidate and the most frequent value is correct in only 47% (Figure 1). These statistics motivate two complementary checking tasks. For disputed claims, the verifier seeks evidence that distinguishes the exposed alternatives, while allowing that every candidate may be incorrect. For consensus claims, it must first propose how the consensus value could be wrong and then test those proposals against the environment. Proposing failures requires knowledge of how artifacts fail, which the pool does not supply. Section 3 develops a harness for both tasks.

3 VERIHARNESS

VERIHARNESS implements the two checking tasks of Section 2 with the same model as the generator. As Figure 2 illustrates, a disagreement resolver and a consensus challenger use shared tools and verification skills to produce evidence records. Adjudication in a fresh context turns both records into a selection and revision plan for the delivered artifact. Appendix B discusses why the same model can detect errors made during generation.

3.1 A GENERAL-PURPOSE VERIFICATION HARNESS

An agent harness $\mathcal{H}_G$ supplies the tools, memory, and operating loop that turn a frozen model M into an agent, $\text{Agent} = (M, \mathcal{H}_G)$. VERIHARNESS applies the same construction to verification:

$$\text{Verifier} = (M, \mathcal{H}_V), \qquad \mathcal{H}_V = (\mathcal{W}, \mathcal{T}, \Pi, \mathcal{S}), \tag{2}$$

where $\mathcal{W}$ is the workspace, $\mathcal{T}$ the evidence tools, Π the verification protocol, and $\mathcal{S}$ the skill library. Table 5 summarizes how they provide access, organize checking, and supply reusable knowledge.

Workspace and tools. The workspace exposes each rollout's artifact d_i, final state w_i, and trace τ_i as files alongside the task description s and environment $\mathcal{E}$. The verifier can trace a claim to its source or computation, recompute quantities, and execute code to obtain evidence $\mathcal{E}(\chi)$ for a check χ.

Verification protocol. The protocol Π assigns the two checking tasks to separate model contexts, followed by adjudication and delivery. The program provides the contexts and information access; the model chooses which claims to examine, which checks to run, and when to stop. Because the workspace is file-based and the protocol does not depend on the artifact type, the same harness applies to reports, spreadsheets, and code; only the skills hold artifact-specific knowledge.

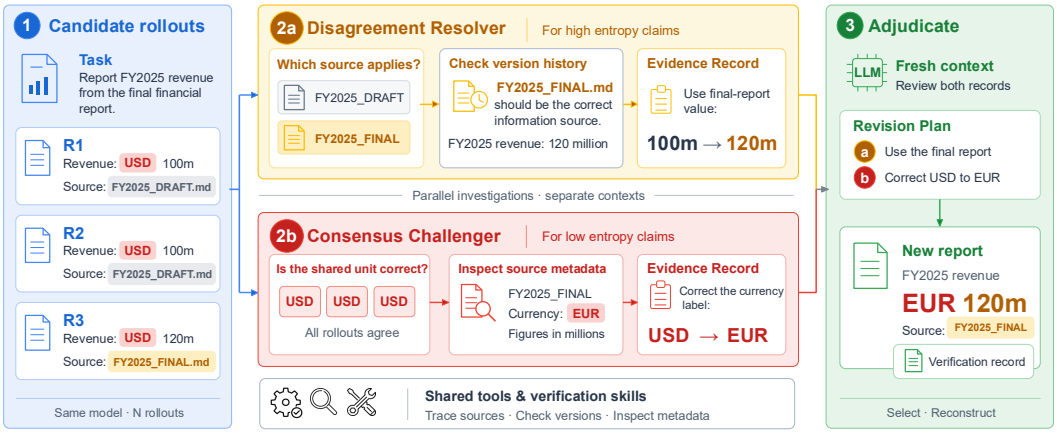

Figure 2: **VERIHARNESS on a worked example.** The task asks for FY2025 revenue from the final financial report. (1) Two rollouts cite the draft and report 100m; a third cites the final report and gives 120m, and all three label the figure USD. (2a) The resolver checks version history and establishes that the final report supersedes the draft, supporting 120m. (2b) In a separate context, the challenger checks source metadata and finds that the currency every rollout uses should be EUR. (3) Adjudication in a fresh context combines the evidence into a revision plan, and delivery produces the revised report with its verification record.

Verification skills. A skill describes a reusable failure mode and its checking procedure in a short text with an optional script. Tools perform a check, while skills identify what needs checking: a tool reads a financial table, whereas a skill directs the verifier to test whether a growth rate spans the years in its label. Skills exclude task-specific answers. Section 4 uses a human-authored library $\mathcal{S}$, listed in full with the tools in Appendix G; Section 5 studies how failure feedback can extend this library without changing the model or protocol.

3.2 RESOLVING DISAGREEMENT THROUGH EVIDENCE

The disagreement resolver compares artifacts to locate a disputed claim p and candidate values $\mathcal{V}_p$, which it traces to their sources and computations. It chooses the check that best distinguishes them. Verification skills guide these choices, including the source-version check in Figure 2 and checks of competing interpretations of a workbook convention. Let b_p be its current belief over the correct value given the evidence $e_{<k}$ gathered before step k, distinct from the empirical frequencies π_p in Section 2. The resolver prefers checks with large expected information gain (Lindley, 1956),

$$\mathrm{Gain}(\chi) = \mathbb{E}\big[\, H(b_p) - H(b_p(\cdot \mid \mathcal{E}(\chi))) \;\big|\; e_{<k}\big], \tag{3}$$

which it judges qualitatively rather than computes. After running a check and obtaining evidence e, the resolver eliminates contradicted candidates:

$$\mathcal{V}_p \leftarrow \{v \in \mathcal{V}_p : \; v \text{ is consistent with } e\}. \tag{4}$$

The resolver closes a claim when the evidence settles it and stops when it judges that further checks are unlikely to change the outcome. If the evidence contradicts every candidate, it records a new value only when the evidence establishes one; otherwise the claim remains open. The resolver records all outcomes in the evidence record $L_{\neq}$ for joint adjudication with the challenger's findings.

3.3 CHALLENGING CONSENSUS

Consensus supplies no competing value to test, so the challenger identifies consensus claims, proposes how each could fail, and tests those possibilities against the environment. It prioritizes checks by how likely they are to expose a task-relevant error given the task, the pool, and the evidence so far,

$$\mathrm{Priority}(\chi) = P\big(\chi \text{ exposes a failure of the consensus claim} \;\big|\; t, R, e_{<k}\big), \tag{5}$$

which it judges qualitatively rather than estimates numerically. The challenger examines three kinds of consensus claim::

- **Consensus values.** The challenger recomputes quantities from raw inputs and compares labels with source metadata, including the currency check in Figure 2.
- **Consensus readings.** The challenger tests a common interpretation against the file the task names, the period the source states, or the definition the workbook carries.
- **Consensus omissions.** The challenger checks the delivered artifacts against the task description s for requirements that every rollout overlooked.

An omission must first be made visible as a verification target because it appears in no artifact. More generally, challenging consensus requires knowledge of how an apparently settled artifact can fail. Skills supply that knowledge: a sign-convention skill, for example, gives the challenger a concrete property to test despite unanimous values.

The challenger records its findings in the evidence record $L_{=}$.

3.4 Adjudication and Evidence-Backed Revision

Adjudication. The investigations run in separate contexts so that disputed claims do not displace attention from consensus claims; Appendix E evaluates this separation. A fresh context of the same model then reviews both records with the task and rollouts, without inheriting either investigator's conversational history. It can uphold, overturn, or leave findings unresolved. It outputs a base rollout, an evidence-supported revision plan, and a list of unresolved claims:

$$(\text{base},\ \text{revision_plan},\ \text{unresolved}) = A(L_{\neq}, L_{=}, t, R), \tag{6}$$

where $\text{base} \in R \cup \{\varnothing\}$ is the rollout to build on, revision_plan specifies the changes to make and their supporting evidence, and unresolved lists claims that the evidence has not settled. Adjudication selects the base with the fewest evidence-supported defects on the task's substantive requirements, weighing findings by their importance to the requested deliverable. When no candidate provides a usable starting point, it sets $\text{base} = \varnothing$.

Delivery and verification record. In selection-only evaluation, the adjudicated $\text{base} \in R$ is returned unchanged. The full harness applies the revision plan and returns the resulting rollout with its verification record:

$$\begin{aligned} r^{\star} &= \text{Apply}(\text{base},\ \text{revision_plan},\ \text{unresolved}), \\ C &= (L_{\neq},\ L_{=},\ \text{base},\ \text{revision_plan},\ \text{unresolved}). \end{aligned} \tag{7}$$

An empty revision plan leaves the base unchanged; a nonempty plan revises it, or produces a new artifact from the plan when $\text{base} = \varnothing$. Claims in unresolved stay marked as unsettled in C, and delivery presents both readings where the artifact format allows.

Each evidence record links a claim, check, evidence, and verdict, making changes traceable and preserving open issues. Together with grader feedback on development tasks, these records support skill development in Section 5. Appendix C gives the complete procedure and delivery rules.

4 Experiments

4.1 Setup

Evaluation scope. We evaluate on five recent long-horizon workspace benchmarks spanning professional documents, spreadsheets, code, and multi-file tasks: APEX-Agents (version 1.0) (Vidgen et al., 2026), Workspace-Bench Lite (Tang et al., 2026), WorkBuddy Bench (Tencent WorkBuddy Bench Team et al., 2026), SpreadsheetBench 2 (Zhu et al., 2026), and JobBench (Li et al., 2026b). Table 3 in Appendix D summarizes their scale, artifacts, and graders.

Same-model comparison. We use Gemini 3.5 Flash and Claude Opus 4.8 as both generator and verifier, with thinking level *high*. For every task and model, all methods receive the same frozen pool of $N = 10$ rollouts and the same model. Outputs are scored by the benchmark graders, which are never available to the verifier. Appendix D gives the task, runtime, and grading details.

Baselines. We compare VeriHarness with majority voting, best-of-N judging, a pairwise tournament, LLM-as-a-Verifier (Kwok et al., 2026), and an *agentic verifier (env. access)* given the same

Table 1: **Main results.** Three-seed mean score on each benchmark's own scale, with the cross-seed standard deviation in gray; Avg is the unweighted mean over the five benchmarks, and gains are over the single rollout. Within each model, bold marks the best score in the primary selection comparison and, separately, among the methods with revision; CLI variants are reported separately, followed by methods that produce new artifacts. The single rollout is a reference baseline. The selection oracle takes the highest-scoring rollout per task under the benchmark grader and bounds selection from the fixed pool; it does not bound revision. SB-2: SpreadsheetBench 2; WSB: Workspace-Bench Lite.

Method	Avg	APEX	WSB	WorkBuddy	SB-2	JobBench
Gemini 3.5 Flash						
Single rollout	47.2	48.1	56.5	73.2	27.3	30.9
Majority voting	47.5	47.3 ±0.4	57.0 ±1.8	74.0 ±1.0	27.0 ±1.3	32.4 ±0.5
Best-of-N with judge	47.3	47.2 ±0.4	57.0 ±2.1	73.6 ±1.2	27.7 ±0.3	30.9 ±0.9
Pairwise tournament	49.1	49.2 ±1.4	59.6 ±1.0	74.8 ±0.8	30.4 ±0.7	31.4 ±1.5
LLM-as-a-Verifier	49.5	49.8 ±0.5	60.4 ±0.2	75.3 ±0.5	30.2 ±0.5	31.9 ±1.2
Agentic verifier (env. access)	49.4	49.0 ±0.8	60.1 ±1.3	75.1 ±0.9	30.9 ±1.1	31.8 ±1.4
VERIHARNESS (select)	**51.6**	**53.3** ±1.2	**62.0** ±0.9	**77.8** ±1.4	**31.5** ±2.4	**33.5** ±1.8
gain over single rollout	+4.4	+5.2	+5.5	+4.6	+4.2	+2.6
Deployed in existing agent CLIs						
VERIHARNESS in Gemini CLI	51.5	51.3 ±1.1	63.2 ±1.1	77.6 ±0.8	31.5 ±1.0	33.8 ±1.2
VERIHARNESS in Codex	51.4	51.5 ±1.1	63.8 ±1.1	75.8 ±1.1	30.5 ±1.1	35.6 ±1.3
Selection oracle	62.7	67.3	70.3	87.7	37.4	51.0
With revision						
Aggregation over the pool	49.9	49.6 ±1.0	60.9 ±1.5	74.4 ±1.1	29.0 ±1.2	35.6 ±1.6
Agentic verifier + revision	50.7	50.5 ±1.0	62.0 ±1.5	76.0 ±1.0	31.0 ±1.2	34.0 ±1.5
VERIHARNESS	**53.4**	**54.8** ±1.0	**65.4** ±1.0	**78.2** ±0.2	**32.2** ±0.6	**36.2** ±1.1
gain over single rollout	+6.2	+6.7	+8.9	+5.0	+4.9	+5.3
Claude Opus 4.8						
Single rollout	49.6	35.8	60.5	79.0	30.1	42.8
Majority voting	50.4	38.0 ±0.2	62.0 ±0.7	79.3 ±0.7	29.5 ±0.5	43.3 ±1.5
Best-of-N with judge	50.6	38.4 ±0.4	62.8 ±0.3	79.2 ±1.1	30.5 ±0.0	42.3 ±1.9
Pairwise tournament	51.3	38.4 ±1.1	63.1 ±0.7	80.5 ±1.4	31.4 ±0.7	42.9 ±0.9
LLM-as-a-Verifier	51.2	36.7 ±0.8	62.4 ±0.2	81.7 ±0.3	30.2 ±0.3	45.0 ±1.8
Agentic verifier (env. access)	51.6	38.1 ±0.9	63.4 ±0.6	80.9 ±0.7	31.8 ±0.8	43.9 ±1.3
VERIHARNESS (select)	**53.7**	**41.2** ±0.7	**65.0** ±0.5	**83.0** ±0.4	**33.4** ±1.2	**46.1** ±0.8
gain over single rollout	+4.1	+5.4	+4.5	+4.0	+3.3	+3.3
Deployed in existing agent CLIs						
VERIHARNESS in Claude Code	51.7	39.0 ±1.1	64.9 ±1.3	81.5 ±0.9	30.9 ±1.0	42.4 ±1.4
VERIHARNESS in Codex	52.2	38.4 ±1.1	63.9 ±1.2	81.8 ±0.9	33.1 ±1.0	43.6 ±1.5
Selection oracle	63.1	53.8	72.4	89.4	39.6	60.4
With revision						
Aggregation over the pool	52.3	40.2 ±1.1	63.5 ±0.8	80.8 ±0.9	30.9 ±0.7	45.9 ±1.4
Agentic verifier + revision	53.1	42.0 ±1.0	64.5 ±0.8	82.0 ±0.7	32.0 ±1.0	45.0 ±1.3
VERIHARNESS	**56.1**	**47.5** ±0.2	**67.3** ±0.2	**83.7** ±0.7	**34.3** ±0.2	**47.6** ±0.5
gain over single rollout	+6.4	+11.7	+6.8	+4.7	+4.2	+4.8

workspace and evidence tools as our verifier but no VeriHarness protocol or skills. VERIHARNESS (select) returns the base chosen by adjudication, and VERIHARNESS also applies the evidence-backed revision plan. For the latter, we add aggregation of the pool into a new artifact (Wang et al., 2025a; Lee et al., 2026b), which also produces an artifact absent from the pool. We also include an *agentic verifier + revision* baseline: the same model checks the pool against the workspace, selects a base, and revises it using the same evidence tools. The single-rollout mean is the reference point for gains, and the grader-informed selection oracle bounds selection from the fixed pool. We report each benchmark's native score over three independent verifier runs, which we refer to as seeds. Appendix D defines the baselines, metrics, and revision protocol.

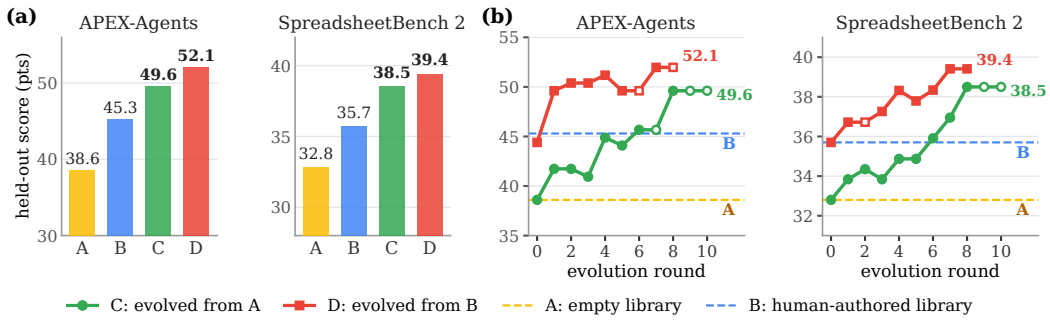

Figure 3: **Evolved libraries improve held-out scores.** (a) Held-out scores of the four libraries. A: empty; B: human-authored; C: evolved from A; D: evolved from B. (b) Held-out scores recorded each round for C and D, with A and B as dashed lines; end labels give the final-library scores of (a). Filled markers after round 0 indicate a new library retained using development scores; hollow markers indicate that the current library is unchanged.

4.2 Results

VeriHarness outperforms all selection baselines across both models. Table 1 shows that it improves the single-rollout average by 4.4 points with Flash and 4.1 points with Opus, exceeds every baseline in all ten model–benchmark settings, and gains more than the cross-seed spread in every cell. Majority voting and best-of-N judging yield only modest average gains on these long-horizon workspace tasks. The consistent advantage over methods that judge only the candidate pool supports the value of checking claims against environmental evidence, and the agentic verifier with the same environment access but no protocol or skills recovers only about half of the harness's gain, so access alone is not sufficient.

The gain transfers to existing agent harnesses. Running the protocol inside Gemini CLI, Claude Code, or Codex instead of our own implementation preserves most of the selection improvement: with Flash, both CLIs match our implementation within two points on every benchmark, and with Opus the CLIs recover most of the gain on Workspace-Bench Lite and WorkBuddy Bench and about half of it on APEX-Agents, but fall back to the single-rollout level on JobBench, as does Claude Code on SpreadsheetBench 2. This result shows that the verification protocol and skills are not tied to one agent implementation; Appendix F discusses why we still provide a dedicated one.

Evidence-backed revision adds value beyond selection. Applying the revision plan improves all ten model–benchmark settings at the cost of one editing pass per task (Appendix J), and raises the average gain over a single rollout to 6.2 points with Flash and 6.4 points with Opus, with the largest added gain on APEX-Agents with Opus and on Workspace-Bench Lite with Flash. Aggregating the pool into a new artifact without evidence checks stays below the harness on every benchmark, so the revision gain comes from the evidence records rather than from rewriting itself.

5 VeriHarness Can Self-Evolve

In this section, we study whether the verification skills can accumulate from failure feedback. The skill library $\mathcal{S}$ supplies reusable knowledge of what to check and how to revise an artifact. Section 4 uses a human-authored library. Here we let the same verifier model improve the library from development-task failures, starting either from an empty library or from the human-authored library. The model, verification protocol Π, and general tools remain fixed.

The learning loop. We split the eligible tasks of each benchmark into a development set and a held-out test set at a ratio of about 3:1. In each round, the verifier model reads development failures together with per-item grader feedback and proposes candidate libraries. The candidate with the highest development score replaces the current library if it matches or exceeds that library's score. Held-out scores are recorded each round but never used for selection, and verification on held-out tasks sees no reference answers or grading rubrics. Appendix H details the split, the failure packages, the filtering of task-specific content, and the selection rule.

Four conditions. We compare **A**, an empty library; **B**, the frozen human-authored library; **C**, the final library evolved from A; and **D**, the final library evolved from B. All four use the full harness, including evidence-backed revision, with Claude Opus 4.8 on APEX-Agents and SpreadsheetBench 2. These experiments use held-out subsets, so their scores are distinct from the full-benchmark results in Table 1. A versus C measures learning from failures, B versus C compares learned and human-authored skills, and B versus D measures improvement beyond human initialization.

Skills learned from empty outperform the human-authored library. Relative to A, C improves the held-out score by 11.0 points on APEX-Agents and 5.7 points on SpreadsheetBench 2 (Figure 3a). C also exceeds B on both benchmarks. With the model and protocol fixed, these comparisons show that failure feedback can improve the reusable skills that guide checking and revision.

Human initialization improves final performance and accelerates progress. Evolving from B yields the highest final score on both benchmarks, adding 6.8 points on APEX-Agents and 3.7 points on SpreadsheetBench 2 over B. Figure 3b shows that D reaches C's final trajectory score earlier on both benchmarks, with a larger reduction in rounds on APEX-Agents. Appendix H gives the round-by-round comparison and distinguishes progress in rounds from compute efficiency.

6 Analysis

We ask four questions about the harness: which parts of the harness carry the gain, on which tasks selection gains, what happens when the verifier faces a unanimous pool, and how the evolved libraries differ from the human-authored library.

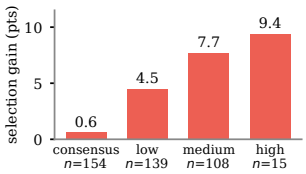

Figure 4: **Selection gains where the pool disagrees.** Selection score minus pool mean on APEX-Agents, Claude Opus 4.8, by the claim entropy of the pool.

The disagreement resolver and the consensus challenger recover different errors. Table 4 in Appendix E runs the full harness with one part removed. With Claude Opus 4.8 the resolver alone raises the five-benchmark average from 49.6 to 54.3 and the challenger alone to 52.6, and the two together reach 56.1, so their gains are complementary. The resolver's share is largest on APEX-Agents, where disputed claims are dense, and the two are closest on SpreadsheetBench 2 and JobBench, where shared errors in units, sign conventions, and omitted requirements make up more of the gap. Merging the two investigations into one context costs 0.8 points, which supports running them in separate contexts. The same ordering holds with Gemini 3.5 Flash.

Selection gains come from the pools that disagree. Following Figure 1, we group the APEX-Agents tasks of Claude Opus 4.8 by the claim entropy of their pools. On the 154 consensus pools, where the ten rollouts give the same answer, the harness's selection scores only 0.6 points above the pool mean, while on the 262 disputed pools it scores 6.1 points above. Within the disputed pools the gain grows with entropy, from 4.5 points on low-entropy pools to 7.7 on medium-entropy pools (Figure 4). Section 2 observed that disagreement exposes correct alternatives, and the harness turns these alternatives into score.

Challenging consensus does no harm, and the shared errors that survive show where skills are needed. On APEX-Agents we match the challenger's verdicts on unanimous claims to the rubric items they concern (Table 7 in Appendix I) and find two things. (i) The challenger is safe: it never refuted a claim that every rollout had right. (ii) The shared errors it left standing are mostly a matter of where it looked. In about 70% of them it tested an input or an intermediate quantity that was correct, while the graded quantity lay downstream, for example confirming the gain on a property sale when the rubric graded the tax owed. Another 14% are differences of definition, period, or source version, and in 12% it confirmed a wrong value. The verifier tends to stop where the rollouts stopped writing, and telling it where else to look is the role of the skill library.

The evolved libraries learn the verifier's own blind spot. We manually compare the 95 checks in the four evolved libraries with the human-authored library: 5 restate a human check, 48 make a human check concrete with a script, a constant, or a document type, and 42 are new. The most notable new checks concern the verifier itself. The library learned on APEX-Agents warns that "agreement between your recompute and the majority is not confirmation," and the library learned on SpreadsheetBench 2, in a separate run, that "reproducing the pool's shared value from the same

inputs is the null result, not a confirmation." Both name the failure of the previous paragraph. Neither appears in the human-authored library, because the pattern shows only across many graded pools.

7 Related Work

Verifying agent outputs. Verification of model outputs began as judgment, with a prompted judge (Zheng et al., 2023) or a verifier or process reward model trained on labeled data (Cobbe et al., 2021; Lightman et al., 2024; Zhang et al., 2025). Recent work scales verification at test time through finer scores, repeated evaluation, and checkable criteria (Kwok et al., 2026; Lifshitz et al., 2025; Zhao et al., 2026; Zeng et al., 2026a; Wan et al., 2026), and lets the verifier act by inspecting the workspace, gathering repository evidence, or calling tools under reinforcement learning (Zhuge et al., 2025; Zeng et al., 2026b; Zhang et al., 2026e; Yuan et al., 2026). Same-model verification faces three known limits: a model cannot correct its own reasoning without external feedback, its self-checks are mostly confirmatory, and its errors correlate with those of a same-family judge (Huang et al., 2024; Long et al., 2026; Goel et al., 2025). VeriHarness is training-free and uses the generator's own model; it takes its targets from the disagreement and consensus among rollouts, settles them with environmental evidence, and returns a better rollout instead of a score.

Test-time scaling for agents. Repeated sampling raises the chance that some rollout is correct, but voting and reward models often fail to identify it (Brown et al., 2024; Snell et al., 2025), and agent studies find that parallel scaling stalls at a verification gap (Zhu et al., 2025; Li et al., 2026a). Proposed remedies vote over the rollouts, compare them in tournaments, or fuse them with an aggregating agent (Kim et al., 2026; Wang et al., 2023; Chen et al., 2023; Wang et al., 2025a; Lee et al., 2026b; Zhao et al., 2025). External evidence has mainly been used for code, through generated tests or distinguishing inputs (Ehrlich et al., 2025; Li et al., 2025), and debate addresses a judge no stronger than the debaters through argument rather than environmental evidence (Irving et al., 2018; Khan et al., 2024). VeriHarness brings evidence acquisition to tasks without executable tests and uses the rollouts to decide what to check.

Harnesses and self-evolution. The agent harness is a recognized engineering layer around a frozen model (Anthropic, 2024; OpenAI, 2026; Wei, 2026), and Huang et al. (2026) extend it to the environment side. Recent work evolves the harness from execution traces (Nie et al., 2026; Zhang et al., 2026b; Lee et al., 2026a; Zhang et al., 2026a; Lin et al., 2026) or grows skill and experience libraries without touching the weights (Wang et al., 2024; 2025b; Ouyang et al., 2026; Zhang et al., 2026d; Gao et al., 2026), and Zhang et al. (2026c) make verification itself an evolving skill. Wang et al. (2026) find that evolved harnesses often fail to beat budget-matched test-time scaling and overfit when development and evaluation sets coincide. VeriHarness evolves the verifier's skills on a development set, evaluates them on held-out tasks, and needs no ground truth at deployment.

8 Conclusion

We presented VeriHarness, a verification harness that uses the generator's own model to resolve disagreement and challenge consensus through environmental evidence. Across five workspace benchmarks and two models, it achieves the highest selection scores among the evaluated baselines, while evidence-backed revision improves average performance over a single rollout by more than six points with each model. We further show that skills evolved from development feedback outperform the human-authored library on held-out tasks, and human initialization further improves final performance. Verification capability therefore does not have to wait for a stronger model. It can be built around the model that exists, from the structure of its own rollouts, the evidence in its environment, and its accumulated experience. Beyond test-time use, the verification record links every claim to its check, evidence, and verdict, and thus provides a claim-level signal that could supervise the generator; turning these records into training data for long-horizon agents is a natural next step. This study has limits: the verifier is the same model as the generator, so the gain from a stronger verifier remains untested; the harness assigns no score to individual rollouts; and we evaluate two frontier models on ten-rollout pools, leaving smaller and open-weight models to future work (Appendix K).

Ethics Statement

This work evaluates on five public benchmarks (APEX-Agents, Workspace-Bench Lite, WorkBuddy Bench, SpreadsheetBench 2, and JobBench) with their released tasks and graders, and involves no human subjects or personal data. Verification runs in a sandbox, with task inputs mounted read-only (Appendix D). We exclude the security domain of WorkBuddy Bench, whose tasks trigger safety alerts from the model APIs. The verification record is meant to make agent outputs easier to audit; it does not replace human review of professional deliverables.

Reproducibility Statement

Section 3 and Algorithm 1 in Appendix C specify the complete procedure. Appendix D gives the benchmark versions, models and thinking level, rollout pools, sandbox, CLI versions, graders, baselines, and metrics; Appendix G lists the tools and the full human-authored skill library; Appendix H describes the split, candidate generation, and selection rules of the self-evolution experiments. All methods share frozen rollout pools, and every reported score is a three-seed mean with its standard deviation. We will release the harness implementation, skill libraries, prompts, and rollout pools upon publication.

AI Use Statement

Large language models are the object of study in this paper: Gemini 3.5 Flash and Claude Opus 4.8 serve as generator, verifier, skill proposer, and the claim reader of Appendices A and I. Beyond this role, we used Claude Code as a coding assistant to implement the harness and baselines, run and monitor experiments. The authors designed the method and experiments, reviewed all AI-assisted code, figures, and text, and take full responsibility for the content of this paper.

References

Anthropic. Building effective AI agents. Anthropic engineering blog, published 19 December 2024, 2024. URL https://www.anthropic.com/research/building-effective-agents.

Samuel R. Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamilė Lukošiūtė, Amanda Askell, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Christopher Olah, Daniela Amodei, Dario Amodei, Dawn Drain, Dustin Li, Eli Tran-Johnson, Jackson Kernion, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Liane Lovitt, Nelson Elhage, Nicholas Schiefer, Nicholas Joseph, Noemí Mercado, Nova DasSarma, Robin Larson, Sam McCandlish, Sandipan Kundu, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Timothy Telleen-Lawton, Tom Brown, Tom Henighan, Tristan Hume, Yuntao Bai, Zac Hatfield-Dodds, Ben Mann, and Jared Kaplan. Measuring progress on scalable oversight for large language models, 2022. URL https://arxiv.org/abs/2211.03540.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever, and Jeffrey Wu. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 4971–5012. PMLR, 2024. URL https://proceedings.mlr.press/v235/burns24b.html.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. Universal self-consistency for large language model generation, 2023. URL https://arxiv.org/abs/2311.17311.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. CodeMonkeys: Scaling test-time compute for software engineering, 2025. URL https://arxiv.org/abs/2501.14723.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru WANG, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=CTr3bovS5F. Survey Certification.

Shashwat Goel, Joschka Strüber, Ilze Amanda Auzina, Karuna K Chandra, Ponnurangam Kumaraguru, Douwe Kiela, Ameya Prabhu, Matthias Bethge, and Jonas Geiping. Great models think alike and this undermines AI oversight. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 19621–19678. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/goel25b.html.

Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, Yifei Ming, Jiaqi Pan, Bhavana Dalvi Mishra, Jiaxin Huang, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. EnvHarness: Awakening static worlds for agent learning, 2026. URL https://arxiv.org/abs/2608.19880.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1051–1068. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.67. URL https://aclanthology.org/2023.emnlp-main.67/.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Geoffrey Irving, Paul Christiano, and Dario Amodei. AI safety via debate, 2018. URL https://arxiv.org/abs/1805.00899.

Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R. Bowman, Tim Rocktäschel, and Ethan Perez. Debating with more persuasive LLMs leads to more truthful answers. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 23662–23733. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/khan24a.html.

Joongwon Kim, Wannan Yang, Kelvin Niu, Hongming Zhang, Yun Zhu, Eryk Helenowski, Ruan Silva, Zhengxing Chen, Srini Iyer, Manzil Zaheer, Daniel Fried, Hannaneh Hajishirzi, Sanjeev Arora, Ruslan Salakhutdinov, Gabriel Synnaeve, and Anirudh Goyal. Scaling test-time compute for agentic coding. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=J9kWWkRzUM.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. LLM-as-a-verifier: A general-purpose verification framework, 2026. URL https://arxiv.org/abs/2607.05391.

Yoonho Lee, Roshen Sanjay Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. In *Third Conference on Language Modeling*, 2026a. URL https://openreview.net/forum?id=tmbOUyFx3R.

Yoonsang Lee, Howard Yen, Xi Ye, and Danqi Chen. Agentic aggregation for parallel scaling of long-horizon agentic tasks. In *Third Conference on Language Modeling*, 2026b. URL https://openreview.net/forum?id=YF0F93vRnj.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15964–15978, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.865. URL https://aclanthology.org/2025.findings-emnlp.865/.

Xiaochuan Li, Ryan Ming, Pranav Setlur, Abhijay Paladugu, Andy Tang, Hao Kang, Shuai Shao, Rong Jin, and Chenyan Xiong. Benchmark test-time scaling of general LLM agents, 2026a. URL https://arxiv.org/abs/2602.18998.

Yuetai Li, Yichen Feng, Zhangchen Xu, Zixian Ma, Kaiyuan Zheng, Fengqing Jiang, Xinghua Sun, Rulin Shao, Zichen Chen, Yue Huang, Xinyang Han, Brian Lee, Kayla Xu, Shenglai Zeng, Hang Hua, Xiangliang Zhang, Basel Alomair, Ranjay Krishna, Luke Zettlemoyer, Pang Wei Koh, Bhaskar Ramasubramanian, Luyao Niu, Xiang Yue, and Radha Poovendran. Jobbench: Aligning agent work with human will, 2026b. URL https://arxiv.org/abs/2605.26329.

Shalev Lifshitz, Sheila A. McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=LriQ3NY9uL.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026. URL https://arxiv.org/abs/2604.25850.

D. V. Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27(4):986–1005, 1956. doi: 10.1214/aoms/1177728069. URL https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-27/issue-4/On-a-Measure-of-the-Information-Provided-by-an-Experiment/10.1214/aoms/1177728069.full.

Quanyu Long, Kai Jie Jiang, Jianda Chen, Xu Guo, Leilei Gan, and Wenya Wang. Self-verification dilemma: Experience-driven suppression of overused checking in LLM reasoning, 2026. URL https://arxiv.org/abs/2602.03485.

Jun Nie, Yonggang Zhang, Jun Song, Qianshu Cai, Dahai Yu, Yike Guo, Xinmei Tian, and Bo Han. TTHE: Test-time harness evolution, 2026. URL https://arxiv.org/abs/2607.08124.

OpenAI. Codex as a platform: Build on the open agent harness. OpenAI Developers blog, published 19 August 2026, 2026. URL https://developers.openai.com/blog/codex-as-a-platform.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, Shaoli Yu, Jinqi Wang, Zihang Zhou, Hongyi Zhou, Yuting Lv, Jinyang Li, Jiashuo Liu, Ruoyu Chen, Chunwei Liu, GuoLiang Li, Jihua Kang, and Fan Wu. Workspace-bench 1.0: Benchmarking ai agents on workspace tasks with large-scale file dependencies, 2026. URL https://arxiv.org/abs/2605.03596.

Tencent WorkBuddy Bench Team, Siqi Cai, Shaopeng Chen, Xiang Fei, Yong Mao, Zihan Xu, Zhiheng Lyu, Zhijian Shao, Yuchen Shi, Shuwen Zhang, Chaofan Qiu, Linjie Che, Xiaoxi Zhao, Feng Wu, Kai Zhang, Chaofan Zhu, Yubin Qi, Xiaoyun Liang, Peijie Dong, Yunhao Zhang, Yuanjie Zhu, Ling Jiang, Xianjun Zhang, Zhehang Chu, Anyuan Sang, Zhen Feng, Sen Nie, Shi Wu, Yuanzhen Xu, Xin Li, Ning Yang, Zhiqiang Dong, Hande Dong, Qiang Lin, Yi Liu, Yunsheng Wu, Ke Li, and Xing Sun. Tencent workbuddy bench: A multi-domain coding-agent benchmark with contamination-resistant task construction, 2026. URL https://arxiv.org/abs/2607.20911.

Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, Neel Venugopal, Alannah Hsia, Isaac Robinson, Calix Huang, Olivia Varones, Daniyal Khan, Michael Haines, Austin Bridges, Jesse Boyle, Koby Twist, Zach Richards, Chirag Mahapatra, Brendan Foody, and Osvald Nitski. Apex-agents, 2026. URL https://arxiv.org/abs/2601.14242.

Yuxuan Wan, Tianqing Fang, Zaitang LI, Yintong Huo, Wenxuan Wang, Haitao Mi, Dong Yu, and Michael R. Lyu. Inference-time scaling of verification: Self-evolving deep research agents via test-time rubric-guided verification. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 24822–24835, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1243. URL https://aclanthology.org/2026.findings-acl.1243/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Junlin Wang, Jue WANG, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=h0ZfDIrj7T.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents, 2026. URL https://arxiv.org/abs/2607.12227.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 13–19 Jul 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Hu Wei. Architectural design decisions in AI agent harnesses, 2026. URL https://arxiv.org/abs/2604.18071.

Runyang You, Hongru Cai, Caiqi Zhang, Qiancheng Xu, Meng Liu, Tiezheng Yu, Yongqi Li, and Wenjie Li. Agent-as-a-judge, 2026. URL https://arxiv.org/abs/2601.05111.

Aojie Yuan, Yi Nian, Haiyue Zhang, Zijian Su, and Yue Zhao. SEVA: Self-evolving verification agent with process reward for fact attribution, 2026. URL https://arxiv.org/abs/2606.29713.

Weihao Zeng, Keqing He, Chuqiao Kuang, Xiaoguang Li, and Junxian He. Pushing test-time scaling limits of deep search with asymmetric verification. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=hxL4Uf9tR3.

Wenhao Zeng, Yuling Shi, Xiaodong Gu, Chao Hu, Chaofan Wang, Yuhao Cui, Hongting Zhou, Mengnan Qi, Jianqiao Wangni, Zhaojian Yu, Shuzheng Gao, Kai Cai, and Shilin He. Dockerless: Environment-free program verifier for coding agents, 2026b. URL https://arxiv.org/abs/2606.28436.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, Yue Liao, Heng Chang, Xiaobin Hu, Qibing Ren, Wangchunshu Zhou, Chuanrui Hu, Yafeng Deng, and Shuicheng Yan. JIT-agent: Scaling harness intelligence via just-in-time harness evolution, 2026a. URL https://arxiv.org/abs/2608.25593.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves, 2026b. URL https://arxiv.org/abs/2606.09498.

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. In *Third Conference on Language Modeling*, 2026c. URL https://openreview.net/forum?id=gQjmJIichQ.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026d. URL https://openreview.net/forum?id=pUpzQZTvGY.

Jiazheng Zhang, Ziche Fu, Zhiheng Xi, Wenqing Jing, Mingxu Chai, Wei He, Guoqiang Zhang, Chenghao Fan, Chenxin An, Wenxiang Chen, Zhicheng Liu, Haojie Pan, Dingwei Zhu, Tao Gui, Qi Zhang, and Xuanjing Huang. AgentV-RL: Scaling reward modeling with agentic verifier. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23078–23100, San Diego, California, United States, July 2026e. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1156. URL https://aclanthology.org/2026.findings-acl.1156/.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Ccwp4tFEtE.

Eric Zhao, Pranjal Awasthi, and Sreenivas Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 77272–77309. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/zhao25a.html.

James Xu Zhao, Hui Chen, Bryan Hooi, and See-Kiong Ng. FineVerify: Scaling test-time compute with fine-grained self-verification for agentic search, 2026. URL https://arxiv.org/abs/2606.00660.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*,

volume 36, pp. 46595–46623. Curran Associates, Inc., 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

Jian Zhu, Yuzheng Zhang, Zeyao Ma, Bohan Zhang, Armin Schoepf, Daniel Woloch, Peter Yiliu Wang, Guangyu Robert Yang, Samuel Jacob, Siddharth Nagisetty, Abhiram Chundru, Jean Lin, Spencer Mateega, and Jing Zhang. Spreadsheetbench 2: Evaluating agents on end-to-end business spreadsheet workflows, 2026. URL https://arxiv.org/abs/2606.29955.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling test-time compute for LLM agents, 2025. URL https://arxiv.org/abs/2506.12928.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 80569–80611. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/zhuge25a.html.

APPENDIX CONTENTS

A Claim-Level Analysis Protocol 17
B Why Same-Model Verification Can Improve Outputs 17
C Verification Procedure and Delivery Details 17
D Experimental Details 18
E Ablation of Harness Components 19
F Why a Dedicated Runtime 20
G Tools and Skills 20
H Self-Evolution Details 21
I Why Shared Errors Survive 23
J Compute Cost 23
K Limitations 24

A Claim-Level Analysis Protocol

This appendix describes how the claim statistics of Figure 1 are computed. The analysis uses the ten-rollout pools of Claude Opus 4.8 on APEX-Agents and covers 434 tasks and 1,692 claims. It is an offline analysis that reads the benchmark's rubrics, which the verifier never sees.

Claims and values. Each rubric criterion names one quantity or determination that the answer should state, and we treat each criterion as one claim p. Claude Opus 4.8 reads the ten answers at temperature zero and records the value $v_{i,p}$ that each rollout asserts for the criterion. Numbers count as the same value when they agree at the criterion's precision, determinations count as the same value when they state the same conclusion, and a rollout that does not address the criterion receives the value *absent*. The grouped values give π_p and $H(\pi_p)$ in Eq. (1), which yields 917 consensus claims and 775 disputed claims.

Correctness. Correctness comes from the benchmark's own grades, which mark every rollout as passing or failing each criterion. A value is correct when the majority of the rollouts asserting it pass the criterion. For consensus claims, we report how often the agreed value is correct. For disputed claims, we report how often at least one rollout passes the criterion and how often the most frequent value is correct, averaging over tied values.

B Why Same-Model Verification Can Improve Outputs

With model parameters fixed, the verifier can still operate with different information, a narrower objective, and accumulated checking experience. Table 2 explains how these differences can help it detect errors made during generation.

Table 2: **Sources of advantage in same-model verification.**

Source	How it helps verification
Exposed alternatives	Multiple rollouts make competing values and interpretations explicit. The verifier can investigate a concrete discrepancy that an individual generation attempt never encountered.
Focused checking	Given completed artifacts, the verifier can devote its attention to testing particular claims. Separate resolver and challenger contexts reserve attention for both disputed and shared claims.
Environmental evidence	A targeted source lookup or recomputation can reveal evidence omitted from the rollouts. The generator has the same tools; verification directs their use toward a specific claim whose validity is now in question.
Accumulated skills	Human-authored checks and procedures learned from development failures suggest what to examine, including requirements that every candidate omitted. This experience can guide later verification without changing model weights.

These advantages explain how verification can improve with a fixed model. Their value depends on finding informative checks: shared blind spots can persist when a required claim goes unexamined or the available evidence cannot settle it.

C Verification Procedure and Delivery Details

Algorithm 1 summarizes the procedure described in Section 3. The program provides the workspace and model contexts; each context uses the same model to choose checks, interpret evidence, and make the decisions assigned to it.

Delivery rules. When $\mathrm{revision_plan}$ is empty, delivery returns the selected base unchanged. Otherwise, it applies the evidence-supported changes to the base; when $\mathrm{base} = \varnothing$, it rebuilds the artifact from the inputs and evidence. Claims in $\mathrm{unresolved}$ remain marked as unsettled in the verification record. For unresolved interpretations, where the artifact format admits it, delivery presents both readings with the base's reading first; where the format has a single slot, it takes the reading preferred by adjudication.

Algorithm 1 VERIHARNESS

Require: task $t = (s, \mathcal{E})$, rollouts R, model M, skills $\mathcal{S}$
expose workspace $\mathcal{W}$ containing R, tools $\mathcal{T}$, protocol Π, and skills $\mathcal{S}$ ▷ program
open two contexts of the verifier that cannot see each other ▷ program
disagreement resolver context: the verifier identifies the disputed claims
while the verifier finds a disputed claim worth a check **do**
    choose p and χ_p by Eq. (3); $e \leftarrow \mathcal{E}(\chi_p)$; eliminate by Eq. (4); record in $L_{\neq}$
end while
consensus challenger context: the verifier identifies the consensus claims
while the verifier finds a consensus claim worth a challenge **do**
    choose χ by Eq. (5); $e \leftarrow \mathcal{E}(\chi)$; record in $L_{=}$ whether the claim holds
end while
open a fresh context of the verifier on $(L_{\neq}, L_{=}, t, R)$ ▷ program
$(\text{base}, \text{revision_plan}, \text{unresolved}) \leftarrow A(L_{\neq}, L_{=}, t, R)$
$r^{\star} \leftarrow \text{Apply}(\text{base},\ \text{revision_plan},\ \text{unresolved})$
return $(r^{\star},\ C)$

D EXPERIMENTAL DETAILS

Benchmarks. All five benchmarks require an agent to produce an artifact from a workspace without exposing the grading rubric or reference answer. APEX-Agents (Vidgen et al., 2026) contains long-horizon investment-banking, legal, and consulting tasks in file-rich simulated environments. All our experiments use version 1.0 of the benchmark; version 1.1 was released after they were completed. Workspace-Bench (Tang et al., 2026) contains realistic workspaces with large file-dependency graphs; we use its 100-task Lite subset. WorkBuddy Bench (Tencent WorkBuddy Bench Team et al., 2026) contains tasks derived from real commits and business scenarios; we evaluate its code, office, and web domains, with 80, 50, and 70 tasks, and do not run its security domain, whose tasks consistently trigger safety alerts from the model APIs. SpreadsheetBench 2 (Zhu et al., 2026) evaluates financial modeling, template completion, debugging, and visualization in large workbooks. JobBench (Li et al., 2026b) covers 35 white-collar occupations with heterogeneous reference files. Table 3 summarizes the task counts, artifacts, and graders.

Table 3: **Benchmarks.** Tasks is the number of tasks in our pools; every task has $N = 10$ rollouts per model.

Benchmark	Tasks	Domain	Artifact	Grader
APEX-Agents (v1.0)	480	banking, law, consulting	memos, models, answers	expert rubric
Workspace-Bench Lite	100	file-heavy workspaces	documents and files	rubric judge
WorkBuddy Bench	200	code, office, web	patches and office files	composite verifier
SpreadsheetBench 2	321	business spreadsheets	workbooks	recompute, compare
JobBench	65	35 white-collar occupations	multi-file artifacts	rubric judge

Models and rollout pools. We use Gemini 3.5 Flash and Claude Opus 4.8 with the thinking level set to high. For each benchmark and model, an executor using the benchmark's standard tools and instructions produces $N = 10$ rollouts per task. Each rollout includes its artifact, final workspace state, and trace. We freeze these pools before verification, so every method operates on the same candidates.

Harness runtime. The verifier has file reading, search, and shell tools. Rollouts, task inputs, and the environment are mounted read-only in a sandbox with no network access; only the output directory is writable. Spreadsheets are also rendered as cell-level text for comparison, and evidence skills cover spreadsheets, documents, presentations, PDFs, document bundles, and code patches (Appendix G). The two investigations, adjudication, and delivery run in separate contexts of the same model, with only the evidence records and revision plan passed between contexts. The CLI rows of Table 1 run the same instruction and skills inside Gemini CLI 0.46.0, Claude Code 2.1.246, and Codex 0.149.1, using each CLI's own tools and sandbox in place of our runtime.

Grading protocol. Each benchmark's own grader scores the delivered artifacts; the verifier never accesses that grader. When the grader is a judge model, the base and delivered rollouts are scored in

the same round. We report the archived base score plus their same-round difference, which controls for judge drift between rounds.

Baselines. Every method receives the same $N = 10$ rollouts and model. The *single rollout* score is the pool mean. *Majority voting* selects the rollout most consistent with the others, extending self-consistency (Wang et al., 2023) to open-ended artifacts. *Best-of-N with a judge* scores all rollouts against the task description in one call. The *pairwise tournament* repeatedly compares two rollouts and advances the winner. *LLM-as-a-Verifier* (Kwok et al., 2026) decomposes the task into criteria and ranks candidates through per-criterion pairwise comparisons. The *agentic verifier (env. access)* runs the best-of-N judge as a tool-using agent in the same sandbox as our verifier, with the same file, search, and shell tools, but with a single scoring instruction and no protocol or skills. The *selection oracle* selects the rollout with the highest grader score for each task and serves only as an upper bound on selection from the fixed pool. For the full harness, *aggregation over the pool* asks the model to write a new artifact from the N rollouts and the task description without environment access, following mixture-of-agents aggregation (Wang et al., 2025a; Lee et al., 2026b). The *agentic verifier + revision* extends the agentic verifier to artifact editing: in a single context, it checks the candidates against the environment, chooses the best starting artifact, and corrects the errors it finds.

Selection and revision. The primary comparison selects an existing rollout, isolating verification judgment from generation and allowing comparison with the oracle. The full VERIHARNESS also applies the adjudicator's revision plan to the selected base. This adds one editing pass without another investigation. Methods that generate a new artifact directly from the pool, such as aggregation agents (Lee et al., 2026b), are outside the selection comparison.

Metrics. Scores use each benchmark's native scale: task success rate for APEX-Agents, mean rubric score for JobBench and Workspace-Bench Lite, the task-weighted mean across WorkBuddy Bench's three domains, and headline accuracy for SpreadsheetBench 2. We report the three-seed mean, gain over the single-rollout score, and cross-seed standard deviation. The single-rollout mean and oracle are expectations over the fixed pool and require no resampling. Figure 5 plots the expected selection-oracle score as a function of the pool size N for both models.

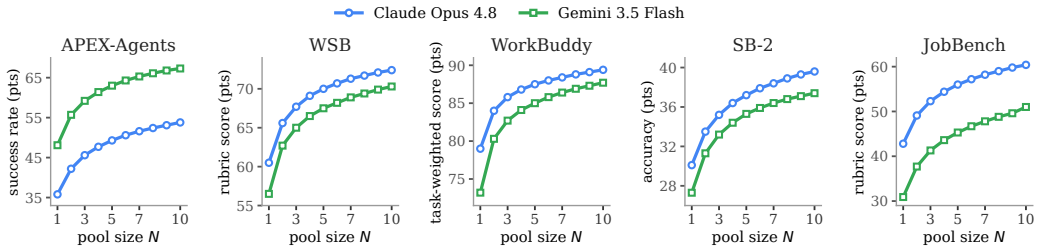

Figure 5: **Selection oracle against pool size.** Expected score of the best rollout among N drawn without replacement from the pool of ten, on each benchmark's own scale; N=1 is the single rollout and N=10 is the selection oracle of Table 1.

E ABLATION OF HARNESS COMPONENTS

Table 4 runs the full harness with only one investigation active. In the resolver-only variant, adjudication receives $L_{\neq}$ alone and consensus claims are never challenged. In the challenger-only variant, it receives $L_{=}$ alone and disputed claims are left as the base rollout states them. Both variants use the same model, tools, skills, adjudication, and delivery as VERIHARNESS. The comparison applies the revision plan, because a consensus error is shared by every candidate and a challenger finding can therefore be realized only through revision. Two further variants keep both investigations: *without skills* runs the protocol with an empty skill library, and *single context* runs the resolver and challenger in one context, so that adjudication reads one combined record. Section 6 discusses the contributions of the resolver and the challenger. Removing the skill library lowers the score on every benchmark, with the largest loss on APEX-Agents. Merging the two investigations into one context costs less but lowers the score on every benchmark, which supports the separate contexts of Section 3.4.

Table 4: **Score of the full harness with one part disabled.** Three-seed mean on each benchmark's own scale after applying the revision plan; Avg is the unweighted mean over the five benchmarks.

Method	Avg	APEX	WSB	WorkBuddy	SB-2	JobBench
Gemini 3.5 Flash						
Single rollout	47.2	48.1	56.5	73.2	27.3	30.9
Resolver only	51.5	53.1	63.2	77.0	30.0	34.0
Challenger only	50.0	50.9	60.3	75.2	29.7	33.7
VERIHARNESS without skills	51.3	50.9	63.5	77.1	30.6	34.4
VERIHARNESS in a single context	52.6	53.9	64.5	77.9	31.3	35.3
VERIHARNESS	**53.4**	**54.8**	**65.4**	**78.2**	**32.2**	**36.2**
Claude Opus 4.8						
Single rollout	49.6	35.8	60.5	79.0	30.1	42.8
Resolver only	54.3	44.6	65.5	82.3	32.9	46.3
Challenger only	52.6	39.3	64.0	80.9	32.6	46.1
VERIHARNESS without skills	53.8	42.0	65.6	82.4	32.8	46.4
VERIHARNESS in a single context	55.3	46.1	66.5	83.3	33.5	47.0
VERIHARNESS	**56.1**	**47.5**	**67.3**	**83.7**	**34.3**	**47.6**

F WHY A DEDICATED RUNTIME

VERIHARNESS's verification protocol and skills can be deployed in either our dedicated runtime or an existing agent CLI. The CLI rows of Table 1 run the protocol inside existing agent CLIs. In the terms of Eq. (2), a CLI keeps its own generation harness $\mathcal{H}_G$ and receives the protocol Π and skills $\mathcal{S}$ as text in its instructions. These rows recover most of the gain, showing that the verification design transfers across runtimes. We nevertheless keep a dedicated runtime H_V for three reasons.

- *The program executes the protocol.* Running the two investigations in separate contexts, passing the evidence records to adjudication, and delivering the artifact with its verification record are steps that $\mathcal{H}_V$ executes explicitly. In our CLI deployments, protocol execution depends on the host agent's operating loop, including its system instructions and context management. The weaker CLI results on JobBench and SpreadsheetBench 2 with Opus suggest that the host runtime can affect how effectively the verification protocol is carried out.
- *Explicit workspace and tool configuration.* $\mathcal{W}$ and $\mathcal{T}$ expose the N rollouts and the environment read-only, expose one writable output, render spreadsheets at cell level, and pass evidence records and revision plans between contexts as files. Our runtime configures these facilities explicitly and consistently across verification runs.
- *A controlled implementation.* Our runtime supports reproducibility and component ablations, as in Table 4, while the CLI results demonstrate the portability of the verification protocol and skills.

G TOOLS AND SKILLS

Table 5: **Components of the verification harness.**

Part	What it provides	Examples
Workspace and tools $(\mathcal{W}, \mathcal{T})$	Access to the objects under check	Read a file, recompute a table, trace a source
Verification protocol Π	Division of checking labor	Resolver, challenger, adjudication
Verification skills $\mathcal{S}$	Reusable knowledge of what to check	Period, unit, source version, required items

Tools. VERIHARNESS adds no tool of its own. The verifier acts through the agent's native tools: `read`, `ls`, `find`, and `grep` to look at the workspace and the rollouts, `bash` to run code, including the scripts a skill ships, and `write` and `edit`, which are restricted to the output directory, for its records and the files it delivers. This is what lets the same protocol and skills run unchanged inside Gemini CLI, Claude Code, and Codex (the CLI rows of Table 1).

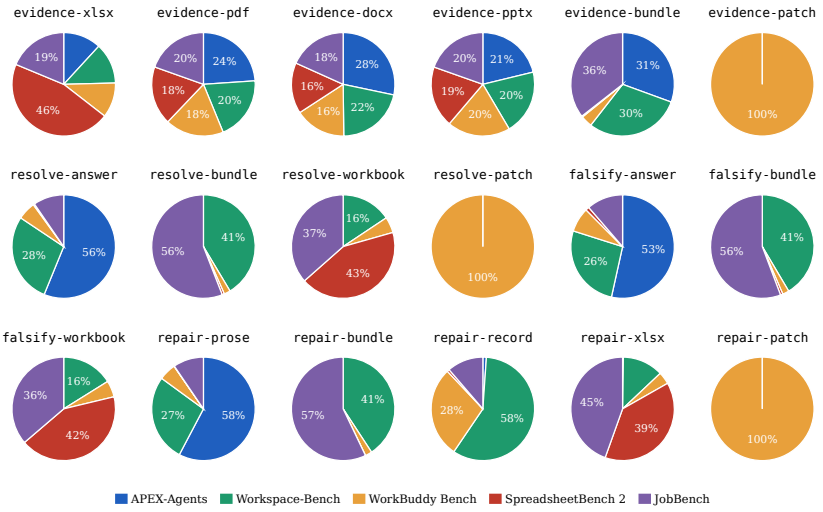

Figure 6: **Where each skill was used.** For each of the eighteen skills, the number of invocations per task in the runs of Table 1 in our implementation, split by benchmark with both models pooled. We report rates rather than counts so that a large benchmark does not weigh more. An invocation is the skill being loaded into a context, its file being read by the model, or one of its scripts being run.

Skills. What the harness adds is the skill library: eighteen skills of four kinds, listed in Table 6. A skill is a short text file, optionally with scripts, written for one kind of deliverable. The six *evidence* skills tell the verifier how to read a workbook, PDF, document, deck, file bundle, or code patch as evidence rather than as extracted text, and ship the scripts for doing so. The four *resolver* skills say what counts as deciding evidence when candidates of a type disagree and in which order of authority to apply it. The three *challenger* skills list the ways a whole pool of that type goes wrong together, one check per entry. The five *revision* skills say how the type is graded, how an unresolved claim is closed, and what must never be edited.

Where the skills are used. Figure 6 shows, for each skill, how its use per task divides among the benchmarks. The document-format evidence skills are used at similar rates everywhere, except the spreadsheet skill, which SpreadsheetBench 2 draws on several times as often; the type-specific skills follow the deliverables each benchmark asks for, from written answers on APEX-Agents to patches and pages on WorkBuddy Bench.

H Self-Evolution Details

Setup and split. The self-evolution experiments use Claude Opus 4.8 as both verifier and library proposer. All four conditions run the complete harness, including revision, with fixed rollout pools, protocol, and general tools; evolution changes only the skill library. The APEX-Agents experiment excludes file-deliverable tasks, and the SpreadsheetBench 2 experiment covers template and financial-model tasks. Each benchmark is partitioned once into about 75% development and 25% held-out tasks. All ten rollouts of a task stay in one partition, and tasks that share a source workspace, source documents, or a workbook template are grouped before splitting so that related instances cannot appear on both sides. The split uses a fixed seed, balances task categories as far as the grouping permits, and is shared by conditions A–D throughout. Figure 3 reports held-out scores under this setup, whereas Table 1 reports full-benchmark pools, so condition B need not match its full-harness row.

Information boundary. The human-authored library is written from general document-handling knowledge and development failures only, then frozen; its authors do not inspect held-out artifacts, traces, or grades. Human-authored and evolved skills contain reusable procedures rather than task identifiers or reference answers. Held-out grades are visible only to the evaluation and are never returned to the proposer or used to change prompts, skills, candidate counts, or stopping

Table 6: **The skill library.** The eighteen skills of the human-authored library, the deliverable each is written for, what it supplies, and the scripts it ships.

Skill	Deliverable	What it supplies	Scripts
`evidence-xlsx`	any	formulas beside cached values, workbook diffs, rendered sheets	`xlsx_dump`, `xlsx_diff`, `xlsx_gaps`, `xlsx_forks`, `xlsx_render`
`evidence-pdf`	any	tables with their columns, words with coordinates, rendered pages	`pdf_render`, `pdf_tables`, `pdf_words`
`evidence-docx`	any	text in reading order, tables, tracked changes and comments	`docx_changes`, `docx_text`
`evidence-pptx`	any	shapes in slide order, tables, charts, notes, rendered slides	`pptx_render`, `pptx_text`
`evidence-bundle`	file bundle	inventory of what each candidate delivered and states	`bundle_inventory`
`evidence-patch`	patch or page	build and test of every candidate patch; headless-browser probe of a delivered page	`patchlab`, `pageprobe`
`resolve-answer`	written answer	which source, provision, and question governs a split	–
`resolve-bundle`	file bundle	the output contract, the basis of a count, enumerated lists, stored values	–
`resolve-workbook`	workbook	order of authority: input note cells, labels, sibling formulas, task words	–
`resolve-patch`	patch	executed behaviour, task words, repository idiom, project tests	–
`falsify-answer`	written answer	unopened provided documents, uncited governing clauses, statistics over the wrong set	–
`falsify-bundle`	file bundle	unused stated filenames, unsurfaced breakdowns, over-broad exclusions	–
`falsify-workbook`	workbook	blank labelled cells, filled unasked cells, scale, references, ranges, check rows	–
`repair-prose`	written answer	how prose is graded, what an unresolved claim permits, what never to change	–
`repair-bundle`	file bundle	how office bundles are graded, coverage against the other candidates	–
`repair-record`	structured record	how records are graded, filling unresolved fields, completing derived lists	–
`repair-xlsx`	workbook	what in-place edits break, recalculation, proof that edits touched only their targets	`xlsx_recalc`
`repair-patch`	patch	edit the tree, never the patch text; the patch must apply, reproduce, and pass what the base passed	–

decisions. The verification run itself never sees reference answers or grading rubrics, so evolution uses development supervision while preserving the verification-time boundary of Section 4.

Learning loop. A development failure package contains the task and workspace, the fixed candidate rollouts, the harness's evidence records and revision plan, the delivered artifact, and the grader's per-item verdicts. From these packages the proposer generates K candidate libraries per round, with $K = 3$ at first and $K = 5$ later; candidates may include scripts, which run under a ten-minute limit. An automatic filter rejects candidates that contain task identifiers, benchmark names, development filenames, or reference-answer values. Each candidate and the current library are evaluated on the same development tasks and pools; the candidate with the largest nonnegative paired gain replaces the current library, and the library is kept when every candidate has a negative gain. The held-out score is recorded after this decision, so a retained update can lower it. The schedules of ten rounds for C and eight for D were fixed before any held-out trajectory was inspected, and the final libraries are those retained at the end of the schedule rather than checkpoints chosen by held-out score.

Round-by-round comparison. On APEX-Agents, C first reaches its final trajectory score of 49.6 at round 8, and D reaches that score at round 1. On SpreadsheetBench 2, C first reaches 38.5 at round 8, while D first exceeds it at round 7 with 39.4, so the reduction in rounds depends on the benchmark. Both trajectories include held-out regressions, consistent with selection on development scores, and candidate counts change across rounds, so round counts alone do not establish a compute or elapsed-time advantage. The figures report point estimates and do not establish statistical significance.

What the learned skills contain. The following procedures illustrate the learned libraries:

- **Requirement coverage.** For a written answer, enumerate required figures, list members, and elements of a governing rule. Compare the value supplied by each rollout for each item, then check the supporting evidence before accepting or revising the base artifact's value.
- **Normalization.** For a normalized financial line, identify each named nonrecurring item and check that the adjustment reverses its contribution under the workbook's conventions.
- **Period alignment.** For a growth rate, identify the start and end columns from the label and check that the formula uses that interval.
- **Sign conventions.** For a line labelled as a deduction, check how the workbook stores and applies the sign before changing a value or formula.

Because evaluation scores the delivered artifact, the gains measure the combined effect of the learned skills on checking, adjudication, and delivery. The held-out comparisons show that the learned libraries help on unseen tasks within the evaluated splits; generalization beyond these task distributions, and improvement without development grader feedback, remain to be established.

I Why Shared Errors Survive

This appendix supports the analysis of the consensus challenger in Section 6. On APEX-Agents with Claude Opus 4.8, we match each row of the challenger's evidence record that concerns a claim shared by all ten rollouts to the rubric items it concerns, using the same model to read the record and the rubric. The archived per-item grades identify the claims on which every rollout was wrong and the challenger upheld the shared value. Table 7 classifies these surviving errors by the relation between the claim the challenger tested and the item the rubric grades.

Table 7: **Why shared errors survive the challenger.** APEX-Agents, Claude Opus 4.8; share of the unanimous errors that the challenger upheld.

Relation between the tested claim and the graded item	Share
The tested claim is an input or intermediate quantity; the graded quantity lies downstream	70%
Same quantity, different definition, period, or source version	14%
The challenger confirmed the wrong value of the graded quantity	12%
Omission or mapping noise	4%

J Compute Cost

This section compares the compute cost of the verification methods under a single accounting. Figure 7 plots each method's compute per task against its gain over the single rollout for both models.

VERIHARNESS lies on the cost–gain frontier for both models, with and without revision. With Gemini 3.5 Flash, selection costs $1.44 per task, a tenth of the cost of LLM-as-a-Verifier at twice the gain, and less than the pairwise tournament. With Claude Opus 4.8, selection costs $3.92, about half the cost of LLM-as-a-Verifier, majority voting, and best-of-N, whose ten-rollout prompts are expensive at Opus prices, and a quarter of the cost of the tournament. Applying the revision plan costs about three times as much as selection and yields the largest gain in the figure. Running the same harness inside an agent CLI reaches the same gain with Flash at three times the cost of our runtime, and a lower gain with Opus at two to three times the cost. The harness is inexpensive because its calls are incremental turns over one workspace, so 86 to 91% of its input tokens are served from cache. When cached tokens are charged at the full input price, selection costs $5.40 with Flash and $13.06 with Opus, and the harness remains on the frontier, because the baselines re-read whole rollouts on every call and do not benefit from the cache.

Metering. Costs are computed from the token counts of the runs at list price, with cached input charged at the cache-read rate. Gains are the five-benchmark means of Table 1. LLM-as-a-Verifier is run with four evaluations per comparison. Majority voting, best-of-N, and the CLI runs are estimates from their call counts and context sizes rather than metered runs and are shown with hollow markers and an asterisk; the aggregation baseline calls its own client and could not be metered.

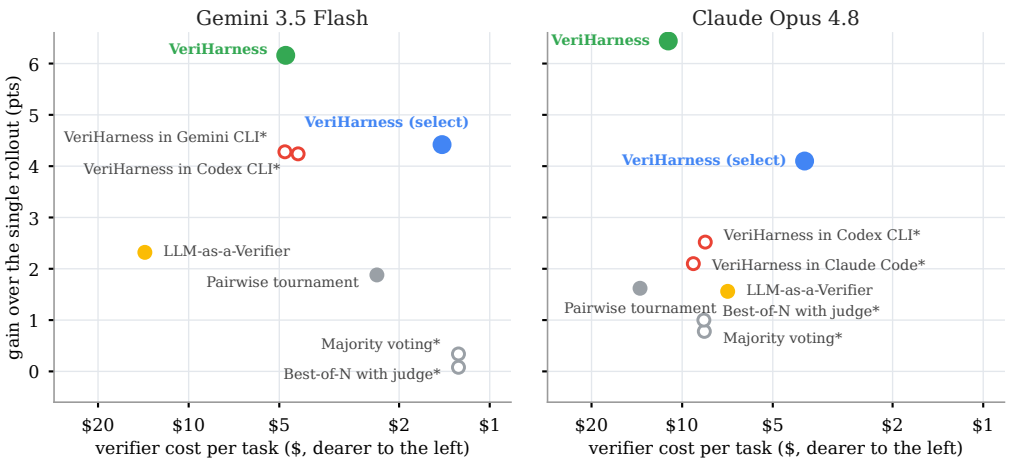

Figure 7: **Cost against gain.** Compute per task at list price with cached input at the cache-read rate, on a log scale with cost decreasing to the right, against the gain over the single rollout, the five-benchmark mean of Table 1. Hollow markers with an asterisk are estimates.

K LIMITATIONS

Cost of the rollout pool. VERIHARNESS verifies from a pool of rollouts, and the reported results use ten per task. This is the standard budget of test-time scaling: every baseline in Table 1 receives the same ten rollouts, so the comparison is made at equal generation cost, and Appendix J reports the cost of verification itself.

No score for individual rollouts. The harness delivers an artifact and a verification record, and it does not assign a scalar score to each rollout. A long-horizon artifact contains many claims that can be right or wrong independently, and the harness judges them claim by claim against evidence. Turning these judgments into a calibrated rollout-level score, for example as a reward signal, is left to future work.

Same-model setting. We restrict our study to a verifier that is the same model as the generator. A stronger verifier may raise the scores further, but its gain would mix the contribution of the harness with the capability gap between the two models, and separating them is outside the scope of this paper.

Model coverage. We evaluate two frontier models, Gemini 3.5 Flash and Claude Opus 4.8. The benchmarks are recent and hard, and each model requires ten long-horizon rollouts per task on five benchmarks before verification begins. We therefore leave open-weight and smaller models to future work.

Latency. Verification is a multi-turn investigation and adds wall-clock time to a task. We target professional deliverables such as reports, workbooks, and patches, where quality matters more than response time. Deep research systems make the same trade and run for many minutes to return a better answer.