RUBRICCRAFT: WHERE SOLUTIONS DIVERGE, CRITERIA EMERGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating language agents on open-ended, artifact-producing tasks is difficult: model-based graders can be inconsistent, while exhaustive programmatic checks are often impractical. Textual rubrics make evaluation criteria explicit, but writing them requires domain expertise. Rubrics generated from task descriptions alone can miss requirements and failure modes that become apparent in attempted solutions. We introduce RubricCraft, which treats rubric construction as the generation of natural-language specifications guided by candidate solutions. Given a task description and independently sampled solver trajectories, RubricCraft compares how each attempt satisfies or violates the task requirements and uses these differences to construct rubric criteria. It applies the criteria to the collected solutions and revises checks that overlook errors or penalize valid alternatives. The resulting rubrics capture partial correctness without requiring any single solution to serve as a complete reference. On SpreadsheetBench v2, against a prompt-only rubric written and applied by the same model, RubricCraft rubrics pass fewer wrong outputs (29.5% against 42.6%), fail none of the outputs the benchmark's own checker accepts, and rank outputs by correctness more often, while separating models as often. Repair on a single attempt leaves grading unchanged, but repair driven by attempts from three model families acts on different tasks, and an evidence audit judges most of those repairs improvements, many on tasks whose reference solution is itself wrong. We release the induced rubrics.

1 INTRODUCTION

Evaluating an agent that produces an artifact is harder than evaluating one that produces an answer. A mathematics answer matches or does not; or a patch passes the test suite or does not (Jimenez et al., 2024). However, a populated spreadsheet or a drafted report has no such checker. Correctness is distributed across many partially-satisfiable properties: whether the model used the requested headers, avoided editing sheets it was not asked to touch, formatted the result so a human can act on it, and refrained from asserting things the data does not support. No single predicate decides the outcome.

The answer to this problem has converged on rubric-based grading: a list of natural-language criteria, written by a domain expert or generated by a model, against which a judge model scores the artifact (Arora et al., 2025; Starace et al., 2025; Zheng et al., 2023). This is a genuine improvement over holistic judging, which is position-biased, verbosity-biased, and unstable (Wang et al., 2024; Panickssery et al., 2024; Dubois et al., 2024). Conditioning the judge on explicit criteria makes the grade decomposable and auditable. However, rubrics themselves should not be treated as ground truth by construction. In practice they might fail in characteristic ways:

- **Unfair criteria.** The criterion demands something the task does not actually require, or requires an artifact property that is unachievable given the input. Every solver is penalized for a defect in the rubric.
- **Incomplete criteria.** The rubric omits a requirement on which solutions genuinely differ. Two materially different artifacts receive the same score, and the benchmark is blind to a real capability gap.

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.

- **Non-discriminative criteria.** The criterion is satisfied by every plausible attempt (or by none). It consumes grading budget and inflates apparent rubric coverage while contributing no signal.
- **Ambiguous or non-checkable criteria.** The criterion cannot be decided from the available evidence, so the judge's verdict is effectively a coin flip and flips again when the judge model is swapped.

Our Approach: Cross-variant Inductive Synthesis. A rubric cannot be simply validated by inspection, because its failure modes are relational: a criterion is unfair only relative to what competent solvers actually produce, and incomplete only relative to the axes on which solutions actually differ. Both become visible under variation. We call the resulting mechanism *cross-variant inductive synthesis*: criteria are induced from the ways independent attempts at the same task *differ*, rather than from reading the task in isolation. We therefore propose to generate N independent attempts under varied configurations, spanning model families, reasoning budgets, and scaffolds. Grading them produces a grid of (criterion, attempt, verdict, evidence) that is the substrate for defect detection: a criterion failed by attempts that a reviewer would call correct is a fairness defect; an axis on which attempts diverge with no corresponding criterion is a coverage defect; a criterion with constant verdict across the attempts is a discrimination defect.

Detected defects should not merely reported and ideally each should be converted into a targeted repair of the offending criterion. Further, the repaired rubric has to be re-validated against the same attempts. Only under this rigorous regime, the output is a rubric that has been pressure-tested against real solver behavior rather than assumed correct.

Evaluating the Evaluator. The methodological difficulty is that rubric quality cannot be measured by the scores the rubric produces. If repair raises pass rates, the natural reading is that grading became easier, not better; if it lowers them, that grading became stricter, not more accurate. We therefore organize rubric meta-evaluation around agreement with expert judgments, fidelity to independently specified checks, discrimination, and stability across judges. We call this workflow RUBRICCRAFT. Our completed study measures descriptive model separation on fixed artifacts, agreement with the benchmark's own checks, and grading stability across repeated runs. A larger score gap or a different ranking does not establish that a grader is more accurate. The control is a naive rubric written by the same model from the task and source workbook alone, graded by the same artifact-inspecting judge, prompt, and tools.

We start with spreadsheets domain because they are hard to grade, and a hard domain stresses every stage of the pipeline. Correctness is spread over many interdependent cells, e.g., in the SpreadsheetBench-v2 tasks we study (Zhu et al., 2026), the median task checks 142 target cells and must leave 6,374 others unchanged, and one wrong reference can propagate across sheets or be offset by another error so that the totals still come out right. Hard is not the same as representative, however: spreadsheets are one artifact type, RUBRICCRAFT was developed on them, and its rubric generator can use the golden workbook's difference from the input.

Contributions.

1. **RUBRICCRAFT**, a multi-agent validation-and-repair pipeline that detects defects in a given rubric by cross-variant inductive synthesis and iteratively repairs the rubric until it passes validation.
2. **A controlled rubric-comparison protocol** with fixed submissions and a matched judge, distinguishing score resolution from independently established evaluation quality.
3. **An exploratory SpreadsheetBench v2 case study** comparing three graders on three solver models: grade correctness against the seed evaluator's cell checks, four rubric-construction conditions graded three times each, an audit of the repairs that diverse attempts trigger, the effect of grading evidence on stability, and two-level semantic coverage of the seed's requirements.
4. **A defect taxonomy for evaluation rubrics**: unfairness, incompleteness, non-discrimination, and non-checkability, each defined operationally with respect to a set of solution attempts.

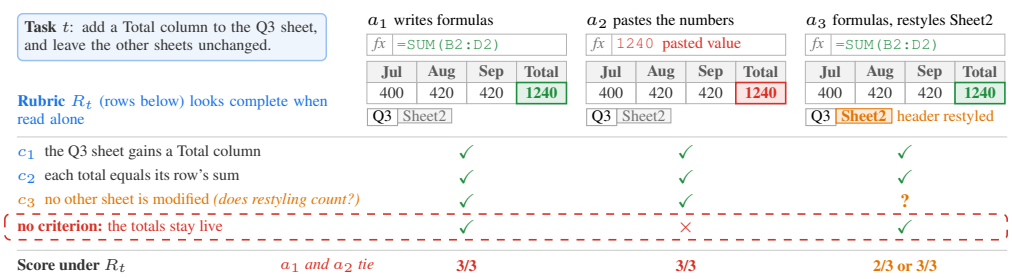


Figure 1: **The running example: a rubric that looks complete in isolation, read against three attempts.** All three attempts put the right totals in a new Total column, so c_1 and c_2 pass everywhere; green marks live formulas and passing verdicts. Red: a_1 and a_2 differ on whether the totals stay live, a formula against a pasted value, yet no criterion checks it, so they tie (incompleteness). Amber: c_3 does not say whether restyling a header modifies Sheet2, so a_3's verdict depends on the judge (non-checkability).

2 Problem Setting and Defect Taxonomy

Setup. A task t consists of a natural-language request and an initial context. A rubric $R_t = \{c_1, \ldots, c_k\}$ is a finite set of criteria, each a natural-language predicate over the produced artifact with an associated weight. A solver π maps t to an artifact a and a trajectory τ. A judge J maps (t, a, τ, c) to a verdict and supporting evidence. The benchmark score aggregates per-criterion verdicts.

We take R_t as input, whether authored by an expert or generated by a model, and ask whether it is fit to serve as the specification of correctness for t.

A running example. The following invented task is used throughout this section; Figure 1 draws it in the same colors. A user supplies a small sales workbook and asks: *"add a Total column to the Q3 sheet, and leave the other sheets unchanged."* A plausible rubric R_t has three criteria: (1) the Q3 sheet gains a column labelled Total; (2) each total equals the sum of that row's monthly values; (3) no other sheet is modified. Now consider three attempts. Attempt 1 writes `=SUM(B2:D2)` in each row. Attempt 2 computes the same numbers and pastes them as constants. Attempt 3 writes the formulas correctly but also restyles a header on an untouched sheet.

Read in isolation, R_t looks complete. Read against these three attempts it is not. Attempts 1 and 2 are scored identically, even though only one of them keeps the totals responsive to later edits, a property the requester plainly cares about and no criterion mentions. Criterion 3 is stated loosely enough that a judge may or may not count restyling as modification, so attempt 3's verdict depends on the judge rather than the artifact. Neither problem is visible from the rubric alone; both are visible the moment the rubric is read against attempts that differ.

2.1 Defects, defined by cross-variant inductive synthesis

Let $A = \{(a_i, \tau_i)\}_{i=1}^{N}$ be a set of attempts and $V \in \{0,1\}^{k \times N}$ the verdict matrix.

Unfairness. Criterion c_j is unfair if attempts that satisfy the task's stated intent nonetheless fail c_j. Operationally, we surface criteria with high failure rates and adjudicate them by rebuttal: the evidence for the failing verdict is re-examined to determine whether the fault lies with the attempt or with the criterion's phrasing or premises.

Incompleteness. R_t is incomplete if attempts differ materially on some axis that no criterion addresses. Detection compares attempts against each other and flags divergences whose verdict contribution is zero under the current rubric. The formula-versus-constant split between attempts 1 and 2 is exactly this case.

Non-discrimination. Criterion c_j is non-discriminative *on the sampled attempts* if row j of V is constant. This is a diagnostic flag, not sufficient evidence that the criterion is defective: a required

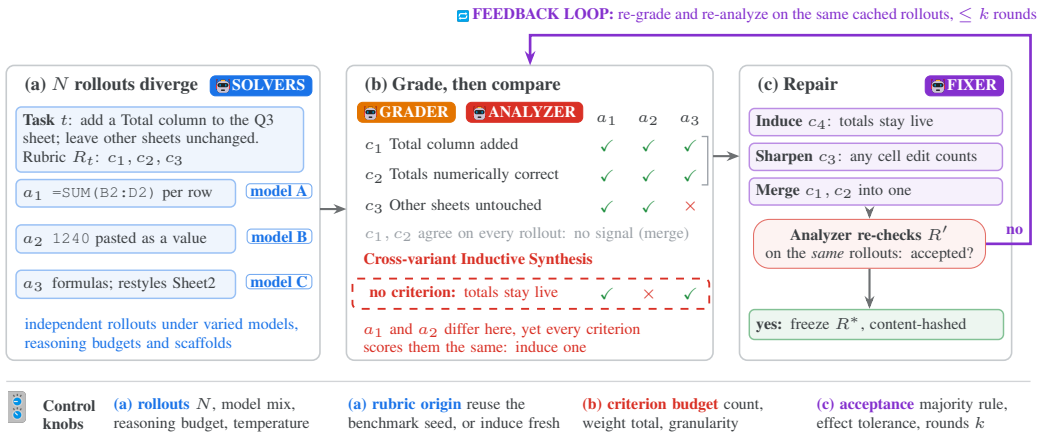


Figure 2: **RUBRICCRAFT is an agentic workflow that induces criteria from the ways rollouts differ.** Running example: a rubric that looks complete in isolation is incomplete once it is read against attempts. **(a)** Solver agents produce N independent rollouts under varied configurations. **(b)** A grader agent fills the verdict matrix with cited workbook evidence, and an analyzer agent compares the rollouts. Rows c_1 and c_2 agree on every rollout, so they separate nothing; a_1 and a_2 differ on whether the totals stay live, yet *no row covers that axis*. Finding such an axis by comparing rollouts, and turning it into a criterion, is *differential induction* (dashed box). **(c)** A fixer agent induces the missing criterion, sharpens the ambiguous one, and merges the redundant pair. The *feedback loop* then re-grades and re-analyzes the repaired rubric on the same cached rollouts, so no score change can come from a newly sampled solution; a rejected rubric returns to (b) for another round, and an accepted one is frozen. The strip lists the control knobs the user sets at each stage.

safety or preservation constraint may legitimately be satisfied by every attempt. Removing such criteria requires adjudication against the task, not variance alone.

Non-checkability. Criterion c_j is non-checkable if the judge cannot ground a verdict in evidence from (t, a, τ). Detection uses verdict instability across repeated and paraphrased judging. Criterion 3 of the running example is the archetype.

Each defect above is defined by a pattern in V or in the differences within A. None is visible from R_t alone. This is the paper's central methodological commitment: rubric validation requires executing the rubric against real solver behavior, just as specification validation requires executing against real program behavior, i.e., cross-variant inductive synthesis.

3 RUBRICCRAFT

RUBRICCRAFT is organized as four cooperating roles operating in a bounded loop (Figure 2).

Solver: constructing the attempts. RUBRICCRAFT generates N attempts under a configuration schedule chosen to *widen behavioral support* rather than to maximize quality: varied model families, reasoning budgets, scaffolds, and decoding temperature. A set of uniformly strong attempts is nearly as uninformative as a uniformly weak one, because in both cases V collapses toward constant rows and the defect signals that cross-variant inductive synthesis depends on vanish.

Grader: producing verdicts with evidence. Each attempt is graded against each criterion, and the judge must emit the evidence supporting its verdict, not only the verdict: a failure rate alone cannot distinguish an unfair criterion from a hard one.

Analyzer: defect detection. The analyzer consumes $(V, \text{evidence}, A)$ and emits a typed defect report per criterion, applying the detectors of Section 2, plus a coverage report of divergence axes with no corresponding criterion. Each defect carries an estimated effect on the score, so defects are ranked by how much they would change grading rather than merely flagged. A rubric is accepted when fairness, coverage, and conciseness each hold for a configurable majority of attempts and any

residual defect falls below a configurable effect tolerance; attempts whose solver or analyzer call failed are excluded rather than counted as passing.

Fixer: targeted repair and re-validation. Each defect becomes a scoped edit: rephrase an unfair criterion, add a criterion for an uncovered axis, merge or delete non-discriminative criteria, or replace a non-checkable criterion with a grounded one. Repairs keep a flat weighted checklist with a fixed point total, so revisions change content rather than scale. The loop re-validates the repaired rubric and halts when validation accepts it, a stage errors, the repair returns an unchanged rubric, or the round budget is exhausted; each repair prompt carries the previous rounds' change summaries, and on exhaustion the pipeline keeps the candidate with the smallest combined fairness and coverage effect rather than the last one attempted. When the attempts are cached, re-validation regrades the same artifacts, so a score change across rounds reflects the rubric rather than a newly sampled solution.

Freezing and cost. An accepted rubric is serialized and content-hashed against the task, so its cost, dominated by the N solves and gradings rather than by rubric editing, is paid once; each later evaluation run is a single grading pass.

4 META-EVALUATION PROTOCOL

A rubric's quality cannot be read off the scores it produces, so we evaluate the grader along four axes. Let $s_{tmg} \in [0, 100]$ be the reported percentage score for task t, model m, and grader g. For RC and naive, $s_{tmg} = 25u_{tmg}$, where $u_{tmg} \in [0, 4]$ is the raw judge grade; for seed, $s_{tmg} = 100y_{tmg}$, where $y_{tmg} \in \{0, 1\}$ is the benchmark's own task outcome. An RC score of 85% therefore means 85% of the maximum grade, not an 85% task-success rate; score differences are percentage points (pp).

Agreement. Do grades agree with human judgments of which outputs are right? Assuming that the benchmark's golden workbooks were constructed and checked by experts, the seed evaluator's cell comparison stands in for a human verdict on every target cell. Let $c_{tm} \in [0, 1]$ be the seed's modified-cell accuracy, the fraction of target cells whose value (or formula, or font color) matches the golden workbook. We call an output *wrong* when $c_{tm} < 0.8$ and report the false-pass rate $\Pr[u_{tmg} \geq 3.5 \mid c_{tm} < 0.8]$ and the false-fail rate $\Pr[u_{tmg} < 3 \mid y_{tm} = 1]$. The proxy inherits any errors in the golden workbooks, of which Section 6 documents several, and it sees only the compared cells, so disagreement with it bounds error rather than measuring it.

Fidelity. Does the grade move with satisfaction of the task's requirements, crediting accepted alternatives and withholding credit where a requirement is unmet? We report how often a grader fails near-complete work ($c_{tm} \geq 0.95$) and work whose only differences lie in cells the task did not ask to change, and how closely its criteria match the seed's requirements, extracted as text (Appendix D).

Discrimination. Do graders resolve differences between fixed solutions? With $\bar{s}_{mg}$ the mean of s_{tmg} over a common task set, we report the model-mean spread $\Delta_g = \max_m \bar{s}_{mg} - \min_m \bar{s}_{mg}$ and the share of tasks whose best and worst model scores differ by at least 12.5 pp; the seed outcome differs by either 0 or 100 pp, so its share is the same for any positive threshold. A common display scale does not make graded credit and binary success the same construct, and more separation is not by itself better grading.

Stability. Does the rubric produce the same verdict when the same artifact is graded again? We grade the same outputs three times under one judge configuration and report the verdict-flip rate and within-task standard deviation.

Baselines. The study compares **RC** (the finalized RUBRICCRAFT rubric), **naive** (a prompt-only rubric from the same model), and **seed**, SpreadsheetBench v2's own programmatic cell comparator. Seed is neither human grading nor the initial rubric R_0: for 297 of 321 tasks the benchmark ships no text rubric, so RUBRICCRAFT generates R_0 itself. We also grade the unrepaired R_0 and a rubric repaired on attempts from three model families (Appendix B).

5 EXPERIMENTS

Benchmark and cohort. We use SpreadsheetBench v2 (Zhu et al., 2026), the successor of SpreadsheetBench (Ma et al., 2024), pinned to dataset revision `5a2215ed` and harness commit `5c160265` and released under the MIT license. RUBRICCRAFT produced a finalized rubric for 302 tasks, 278 of them nonvisual; the 24 visualization tasks are excluded because the seed evaluator compares cells. The primary cohort is the 259 nonvisual tasks (80 debugging, 89 financial-model, and 90 template) on which all three solvers submitted an output and every grader returned a score.

Solvers and graders. GPT-5.6 Sol, GPT-6 Astra, and Opus 5 solve every task through mini-swe-agent 2.4.6 (the names identify the tested development endpoints), and their outputs are frozen before grading, so every grader scores the same workbook bytes. Naive is a rubric written by a separate GPT-5.6 Sol session from the task text and a read-only view of the source workbook, with the instruction "write a grading rubric for the following task" and no rubric-writing guidance, examples, or golden workbook. Both rubrics are applied by the same GPT-5.6 Sol artifact-inspecting judge, with the same harness, prompt, workbook-inspection tools, reasoning setting, and budget; the judge returns a report and a grade on $[0, 4]$. Seed is the benchmark's published comparator, run through an archived local wrapper: it passes an output when every target cell matches the golden workbook and at least 99.8% of preserved cells are unchanged, comparing values on 242 cohort tasks, formulas on nine, and font colors on eight.

Rubric construction. RC was built from GPT-5.6 Sol attempt. A rubric was accepted when a majority of non-errored attempts passed validation and any residual defect moved the grade by less than 0.8 of 4 points. RUBRICCRAFT's generator uses an engineered checklist prompt and receives the source-to-golden workbook difference as a sample solution, so the RC-versus-naive contrast measures the whole construction pipeline. Over the full 321-task run, 244 initial rubrics passed without repair and 58 were repaired, averaging 13.9 model requests and 575K tokens per task and 4.95 hours of wall time in total at concurrency five (Appendix A); each repair round of this run re-sampled the attempt rather than regrading the cached one. A second construction, loop + diversity, repairs the same R_0 on the three solvers' frozen outputs, which it regrades in every round.

Grading evidence and statistics. The submitted workbooks were saved without computed values, so every grader reads an Excel-recalculated copy of each submission, bound to it by the submission's hash. We call an output *wrong* when fewer than 80% of its target cells match and *near-complete* when at least 95% do; a rubric *passes* an output at a grade of at least $3.5/4$ and *fails* it below $3/4$. Intervals are 95% paired task-bootstrap intervals.

6 RESULTS

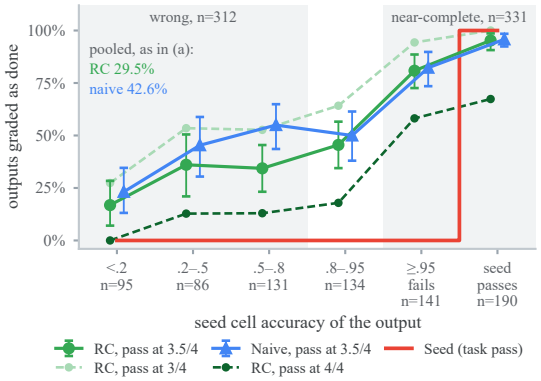
Diverse attempts expose defects that a single attempt misses. Repair on the three solvers' frozen outputs and repair on one attempt changed 29 and 30 of the 259 cohort rubrics, only 9 in common: 20 of the diversity repairs fall on tasks whose R_0 the single attempt accepted. Where diversity repaired, the rubric covers more of the seed's required fixes than R_0 (48.6% against 38.5% on 27 tasks, $[+3.6, +17.7]$ pp), whereas single-attempt repairs show no such gain (35.9% against 39.1%, $[-8.4, +0.5]$ pp). An audit of 19 of the 20 diversity-only repairs, in which reviewer agents checked each repair against the cells where solver outputs and the golden workbook disagree, judged 11 improvements, 4 regressions, and 4 neutral, and found the golden reference itself wrong on 11: when most of three solver families disagree with a rubric, the fault often lies in the rubric's reference. The same repairs contradict more of the seed's expected values (10.4% against 0.2%) and pass more of Astra's wrong outputs on these tasks (61% against 11% of 19), partly because they override wrong references and partly through regressions, such as accepting a mistake all three solvers share. Since they were validated on the very outputs graded here and audited by agents, these repairs are evidence for the mechanism rather than a held-out gain (Appendix B).

Agreement. Figure 3a compares RC with the naive rubric on the four axes of Section 4: eight of its twelve intervals favor RC, three include zero, and one favors naive. RC passes 29.5% of the 312 wrong outputs against 42.6% for naive, fails none of the 190 outputs that the seed passes (naive 2.6%), and ties or reverses 34.1% of the 258 same-task pairs whose cell accuracies differ by at least 5 points (naive 53.9%, seed bit 84.9%). Figure 3b shows where the first row comes from: both rubrics' pass rates rise with the seed's cell accuracy, RC's stays below naive's in the three wrong

a RC against naive on the four axes paired 95% task-bootstrap intervals, oriented so that right of zero favors RC

Axis	Measure		*n*	RC	Naive	Seed	RC advantage (pp)	95% CI
Agreement	Wrong outputs passed b	↓	312	**29.5**	42.6	0*		[6.7, 19.5]
with golden workbooks	Seed-passed outputs failed	↓	190	**0.0**	2.6	0*		[0.5, 5.2]
	Output pairs tied or misordered	↓	258 pairs	**34.1**	53.9	84.9		[11.1, 28.2]
	Near-complete above wrong (AUC)	↑	331 vs 312	**88.8**	81.3	–		[3.8, 11.0]
Fidelity	Near-complete outputs failed	↓	331	**2.4**	6.6	42.6		[1.6, 7.5]
to what the task asks	Failed only for untouched cells c	↓	37	2.7	8.1	100		[0.0, 13.9]
	Criteria supported by seed values	↑	238 tasks	**9.0**	4.0	–		[2.7, 7.5]
	Seed requirements contradicted	↓	238 tasks	**2.6**	14.0	–		[8.8, 14.4]
Discrimination	Tasks separating models by ≥12.5 pp	↑	259 tasks	44.0	43.6	13.1		[−4.6, 5.8]
between models	Seed gap, passed − failed (pp)	↑	242 tasks	**18.0**	12.2	–		[3.3, 8.3]
Stability	Verdict flips across three gradings	↓	242 tasks	9.1	7.4	–		[−6.6, 2.9]
across gradings	Within-task grade SD (pp)	↓	242 tasks	2.2	**1.2**	–		[−1.8, −0.4]

b Share of outputs graded as done



c A correct edit the seed fails

one of the 37 outputs failed only for untouched cells

LBO audit task: "replace hardcoded values that should be formulas" is a named objective

LBO!C33, golden vs. submitted:

`0.053 → ='Ex 12 - Debt Schedule'!H9`

H9 = F9 + G9 = 0.018 + 0.035 = 0.053 in both files

same value, now derived instead of restated

Seed	Naive	RC
FAIL	57%	100%

Seed: all 21 target cells right, but it fails the task on regression (98.96% of preserved cells, first mismatch C33); naive deducts for other, unverified gaps.

Figure 3: **RC against the naive rubric on the four axes, with the two views behind the marked rows.** (a) Values are percentages unless marked pp, and n counts outputs unless marked; the seed-gap and stability rows grade GPT-6 Astra's outputs three times. Intervals are 95% paired task-bootstrap intervals for RC's advantage over naive, signed so that right of zero favors RC (green: excludes zero in its favor; blue: in naive's; gray: includes zero); bold marks the better value where the interval excludes zero. AUC is the probability that a near-complete output is graded above a wrong one. *Zero by construction. (b) Share of the 777 outputs graded as done, by the seed's cell accuracy; the row marked b pools the three wrong bands at $3.5/4$, and the seed's task bit (red) passes nothing below its all-cells threshold. Dashed lines move RC's threshold to $3/4$ and $4/4$. (c) One of the 37 outputs in the row marked c, verified from the stored cells (Appendix E.1).

bands, and the seed bit stays at zero until every target cell is right. The gap does not hinge on the pass threshold (AUC 88.8% against 81.3%; at $4/4$, 9.0% against 12.8% of wrong outputs passed), on the cut that defines wrong, or on the nine formula-comparison tasks (Appendix C). Because the audit above found golden workbooks wrong, some counted false passes are corrections of the reference, and the rates bound disagreement rather than measure error.

Fidelity. The seed bit fails 42.6% of the 331 near-complete outputs and all 37 whose only differences lie in cells the task did not ask to change; RC fails 2.4% and 2.7% of them, naive 6.6% and 8.1%. Figure 3c is one of the 37: in a leveraged-buyout audit that asks for "hardcoded values that should be formulas" to be fixed, the submission links `LBO!C33` to a debt-schedule cell with the same value, 0.053, and the seed fails the task on preserved-cell mismatches, the first of them this cell, while RC awards full marks under a criterion that accepts equivalent live links and naive awards $2.3/4$ for other gaps we have not verified. RC criteria are also fully supported by the seed's value-level requirements twice as often as naive criteria (9.0% against 4.0%) and contradict a fifth as many

(2.6% against 14.0%), while naive rubrics, 4.9 times longer, cover more of the seed's fixes (57% against 46%).

Discrimination and stability. The seed bit gives the three models the same outcome on 225 of the 259 tasks, whereas both rubrics separate them by at least 12.5 pp on about 44% of tasks; RC ties its separations to correctness, grading Astra's seed-passed outputs 18.0 pp above its seed-failed ones against 12.2 pp for naive. Naive's larger model-mean spread (10.21 against 4.89 pp) comes almost entirely from debugging tasks, on which it grades Astra 27 and 30 pp above the other two models. In each of three gradings RC passes 29.5–35.2% of Astra's 105 wrong outputs and naive 45.7–48.6%; verdict flips do not distinguish the rubrics, and RC's grades vary more within a task, the one measure on which naive is better. Both rates depend on computed values: graded from the uncached submissions, RC flipped 16.5% of verdicts and passed 43–52% of the wrong outputs. The unrepaired R_0 matches RC on these measures (seed gap 19.0 pp, 32.7% false passes, 6.2% flips), so at $N = 1$ the gains come from RUBRICCRAFT's generator and its golden-difference evidence rather than from repair (Appendix B).

7 RELATED WORK

LLM judges and their reliability. Using a strong model as a grader became standard with MT-Bench and Chatbot Arena (Zheng et al., 2023), G-Eval's form-filling scores (Liu et al., 2023b), and evidence that model ratings track expert ratings of open-ended text (Chiang & Lee, 2023); Gu et al. (2025) survey the area. The same judges are position-biased (Wang et al., 2024), prefer their own generations (Panickssery et al., 2024), and reward length unless it is controlled (Dubois et al., 2024), and on hard, objectively checkable response pairs strong judges barely beat chance (Tan et al., 2025); RewardBench meta-evaluates reward models in the same spirit (Lambert et al., 2024b). This line of work audits the judge. We hold the judge fixed and audit the specification it is given, and the evidence result of Section 6 adds a third source of grading error: what the judge is shown.

Rubrics and checklists as evaluation specifications. Conditioning the judge on explicit criteria is the main remedy. FLASK decomposes scoring into skill-level criteria (Ye et al., 2024), Prometheus trains an open evaluator on score rubrics (Kim et al., 2024), and BiGGen Bench writes instance-specific criteria for every item (Kim et al., 2025). InFoBench scores the fraction of decomposed requirements a response meets (Qin et al., 2024), and TICK has a model generate the checklist from the instruction (Cook et al., 2024). Artifact-level benchmarks now ship rubrics as their ground truth: physician-written criteria in HealthBench (Arora et al., 2025), author-co-developed hierarchical rubrics for research replication in PaperBench (Starace et al., 2025), expert-graded occupational deliverables in GDPval (Patwardhan et al., 2025), and hierarchical requirements checked by an agentic judge in DevAI (Zhuge et al., 2024), whose artifact-inspecting judge resembles ours. In all of these the rubric is written once from the task, by an expert or a model, and then treated as correct; RUBRICCRAFT instead tests and repairs it against attempts. The closest precedent is EvalGen (Shankar et al., 2024), which reports *criteria drift*: users need criteria to grade outputs, grading outputs changes the criteria, and some criteria exist only relative to the outputs observed. Cross-variant inductive synthesis automates that observation, using divergence among attempts in place of a user's grades.

Benchmark validity and grader error. Measurement quality is increasingly treated as a property of the benchmark itself. Test sets carry label errors that can reorder models (Northcutt et al., 2021), most benchmarks omit significance testing and replication support (Reuel et al., 2024), and agentic benchmarks misestimate performance through flawed task setups and reward designs, such as insufficient tests or empty responses counted as success (Zhu et al., 2025). The code-generation analogue of our false passes is EvalPlus, where 80 times more tests caught previously undetected wrong programs and changed model rankings (Liu et al., 2023a). Item response theory treats evaluation items as more or less informative and discriminating (Rodriguez et al., 2021; Polo et al., 2024); our discrimination axis asks the same of graders, and the non-discrimination defect is its criterion-level form. Spreadsheet benchmarks grade by comparing output cells with a reference workbook (Li et al., 2023; Ma et al., 2024; Zhu et al., 2026), the kind of binary comparator we use as the seed; the golden workbooks it compares against are themselves sometimes wrong (Section 6).

Specification inference and testing the tests. The idea that a specification can be recovered or checked by observing a population of implementations rather than by inspecting the specification

alone originates in program analysis. Dynamic invariant detection infers likely invariants from many executions (Ernst et al., 2007); specification mining recovers API contracts from traces (Ammons et al., 2002); and differential testing treats disagreement among independent implementations of the same specification as evidence of a fault or an under-determined specification (McKeeman, 1998; Yang et al., 2011), the premise of N-version programming (Avižienis, 1985). RUBRICCRAFT transposes this to a setting where the "executions" are solver attempts, the "invariants" are natural-language criteria, and agreement must be judged semantically rather than by equality. The transposition removes the sound checker that makes the program-analysis version tractable, which is precisely why our meta-evaluation (Section 4) relies on external gold signals rather than internal consistency. Software testing also evaluates its own evaluators: mutation analysis scores a test suite by the seeded faults it detects (DeMillo et al., 1978), mutant detection correlates with real-fault detection (Just et al., 2014), and the oracle problem names the difficulty of deciding correct behavior at all (Barr et al., 2015). A rubric is a test oracle written in natural language, and our false-pass and false-fail rates against the seed's cells play the role of a mutation score.

Evaluation signals in post-training. Post-training on checkable tasks rewards programmatic verifiers (Lambert et al., 2024a; Guo et al., 2025), and group-based methods derive advantages from comparisons among rollouts (Shao et al., 2024). Rubric-derived scores extend such rewards to open-ended tasks: Rubrics as Rewards optimizes against instance-specific rubrics in medicine and science (Gunjal et al., 2025), and checklist feedback outperforms reward models for instruction following (Viswanathan et al., 2025). Optimizing an imperfect proxy degrades the true objective (Gao et al., 2023), so once a rubric becomes a reward its defects become the policy's target. Our work is upstream of this line: rather than proposing a reward, it asks whether the criteria such rewards are computed from are fair, complete, and discriminative in the first place. The observation that distilling per-attempt comparisons into a persistent symbolic rubric makes an otherwise transient signal reusable is a motivation for this work rather than a claim we evaluate here.

8 CONCLUSION

Rubrics are evaluation instruments, and their quality can be measured. On SpreadsheetBench v2, RUBRICCRAFT rubrics improve on a naive rubric from the same model and judge in agreement, fidelity, and discrimination: they pass fewer wrong outputs (29.5% against 42.6%), fail less near-complete work (2.4% against 6.6%), contradict a fifth as many of the benchmark's expected values, and tie model separation to correctness (a seed gap of 18.0 against 12.2 pp), with verdicts as reproducible as naive's when the judge reads computed values. With one attempt the gains come from construction; with three model families, repair acts on different tasks and an audit judges most of those repairs improvements. Testing cross-variant inductive synthesis on fresh rollouts with a human audit is the next step.

REPRODUCIBILITY STATEMENT

Section 5 and Appendices A–D specify the cohort, rubric construction, grading protocol, and statistics.

The graded outputs come from mini-swe-agent 2.4.6, the minimal bash-only agent of the SWE-agent project (Yang et al., 2024), run with its default agent loop in a local shell. Every request asks for medium reasoning effort and leaves sampling at the provider's defaults. Each task starts in a fresh working directory that holds only the input workbook. A run is capped at 40 model calls, which no run reached, and 180 s per command.

AI USE STATEMENT

The authors used generative AI exclusively for language polishing, grammar correction, and improving readability of the manuscript text. No generative AI was used for developing core concepts, synthesizing data, or mathematical formulations. The authors reviewed and edited all AI-assisted phrasing.

REFERENCES

Glenn Ammons, Rastislav Bodík, and James R. Larus. Mining specifications. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2002.

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, et al. HealthBench: Evaluating large language models towards improved human health. *arXiv preprint arXiv:2505.08775*, 2025.

Algirdas Avižienis. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-11(12):1491–1501, 1985.

Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2015.

Cheng-Han Chiang and Hung-yi Lee. Can large language models be an alternative to human evaluations? In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

Jonathan Cook, Tim Rocktäschel, Jakob Foerster, Dennis Aumiller, and Alex Wang. TICKing all the boxes: Generated checklists improve LLM evaluation and generation. *arXiv preprint arXiv:2410.03608*, 2024.

Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, 1978.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-controlled AlpacaEval: A simple way to debias automatic evaluators. In *Conference on Language Modeling (COLM)*, 2024.

Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, et al. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1–3):35–45, 2007.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning (ICML)*, 2023.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, et al. A survey on LLM-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2025.

Anisha Gunjal, Anthony Wang, Elaine Lau, et al. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *arXiv preprint arXiv:2507.17746*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, et al. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations (ICLR)*, 2024.

René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, pp. 654–665, 2014.

Seungone Kim, Jamin Shin, Yejin Cho, et al. Prometheus: Inducing fine-grained evaluation capability in language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Seungone Kim, Juyoung Suk, Ji Yong Cho, et al. The BiGGen bench: A principled benchmark for fine-grained evaluation of language models with language models. In *Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, 2025.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, et al. Tülu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024a.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, et al. RewardBench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*, 2024b.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. SheetCopilot: Bringing software productivity to the next level through large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023b.

Zeyao Ma, Bohan Zhang, Jing Zhang, et al. SpreadsheetBench: Towards challenging real world spreadsheet manipulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

William M. McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1):100–107, 1998.

Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. In *NeurIPS Datasets and Benchmarks Track*, 2021.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations. *arXiv preprint arXiv:2404.13076*, 2024.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, et al. GDPval: Evaluating AI model performance on real-world economically valuable tasks. *arXiv preprint arXiv:2510.04374*, 2025.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinyBenchmarks: Evaluating LLMs with fewer examples. In *International Conference on Machine Learning (ICML)*, 2024.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, et al. InFoBench: Evaluating instruction following ability in large language models. *arXiv preprint arXiv:2401.03601*, 2024.

Anka Reuel, Amelia Hardy, Chandler Smith, Max Lamparth, Malcolm Hardy, and Mykel J. Kochenderfer. BetterBench: Assessing AI benchmarks, uncovering issues, and establishing best practices. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Pedro Rodriguez, Joe Barrow, Alexander Miserlis Hoyle, John P. Lalor, Robin Jia, and Jordan Boyd-Graber. Evaluation examples are not equally informative: How should that change NLP leaderboards? In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4486–4503, 2021.

Shreya Shankar, J.D. Zamfirescu-Pereira, Björn Hartmann, Aditya G. Parameswaran, and Ian Arawjo. Who validates the validators? aligning LLM-assisted evaluation of LLM outputs with human preferences. *arXiv preprint arXiv:2404.12272*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Giulio Starace, Oliver Jaffe, Dane Sherburn, et al. PaperBench: Evaluating AI's ability to replicate AI research. *arXiv preprint arXiv:2504.01848*, 2025.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y. Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. JudgeBench: A benchmark for evaluating LLM-based judges. In *International Conference on Learning Representations (ICLR)*, 2025.

Vijay Viswanathan, Yanchao Sun, Shuang Ma, et al. Checklists are better than reward models for aligning language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Peiyi Wang, Lei Li, Liang Chen, et al. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*, 2024.

John Yang, Carlos E. Jimenez, Alexander Wettig, et al. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in C compilers. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 283–294, 2011.

Seonghyeon Ye, Doyoung Kim, Sungdong Kim, et al. FLASK: Fine-grained language model evaluation based on alignment skill sets. In *International Conference on Learning Representations (ICLR)*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, et al. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Jian Zhu, Yuzheng Zhang, Zeyao Ma, Bohan Zhang, Armin Schoepf, Daniel Woloch, Peter Yiliu Wang, Guangyu Robert Yang, et al. SpreadsheetBench 2: Evaluating agents on end-to-end business spreadsheet workflows. *arXiv preprint arXiv:2606.29955*, 2026.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, et al. Establishing best practices for building rigorous agentic benchmarks. *arXiv preprint arXiv:2507.02825*, 2025.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, et al. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.

A Repair-Loop Dynamics

The repair loop converges quickly and does not inflate scores (Figure 4): acceptance saturates within two repair rounds, and on the repairs with a genuine analyzer-flagged defect the grader's score moves in both directions between the first and the final round, with a median change of zero. In this run every repair round re-sampled the solver attempt rather than regrading the cached one, so the change mixes the rubric's revision with a new attempt, and each repaired rubric was accepted on a fresh attempt. Within the 259-task cohort, 30 rubrics were revised; Appendix B shows that at $N = 1$ these revisions do not measurably change grades on a held-fixed solver's outputs.

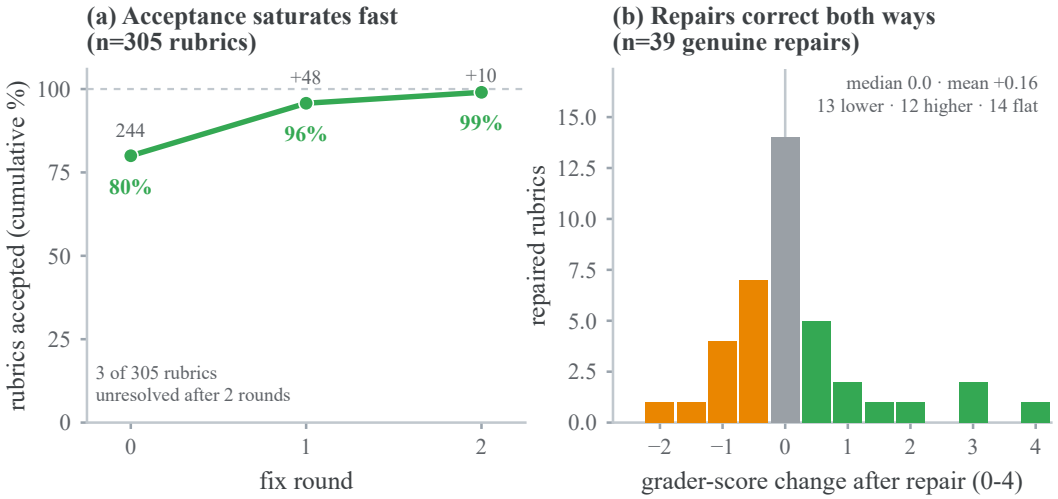


Figure 4: **The repair loop converges fast and does not inflate scores.** (a) Cumulative share of the 305 gradable rubrics accepted by fix round on the full 321-task run (16 infrastructure-skipped tasks excluded): 80% pass unrepaired, 96% within one repair, and 99% within two, with three left unresolved; the third permitted repair round was never needed. (b) For the 39 repairs carrying a clean analyzer verdict at both ends and a genuinely flagged initial defect, the change in the grader's score from the first to the final round; because each round re-sampled the attempt, the change mixes rubric and attempt differences. The distribution centers on zero (median 0; 13 lower, 12 higher, 14 unchanged; mean $+0.16$ of four points), so repair does not systematically raise scores. A further 16 of the 60 multi-round cases were retries after a transient non-JSON analyzer response rather than substantive repairs, and are excluded from (b).

B Rubric Conditions and Grading Evidence

We grade GPT-6 Astra's frozen outputs on 242 tasks three times under each of four rubric conditions, from the recalculated copies and, for comparison, from the uncached submissions (Figure 5, Table 1).

Reproducibility. The three run means of every condition lie within 0.9 pp of one another, and no condition fails any of the 61 seed-passed outputs in any run. Averaged over runs, RC's false-pass rate is 15.2 pp lower than naive's ($[-24.1, -6.9]$), its seed gap 5.8 pp larger ($[+3.3, +8.3]$), and its verdicts match the seed bit 9.8 pp more often (62.4% against 52.6%, $[+4.5, +15.0]$).

Stable grading requires computed values. Graded from the uncached submissions, the RubricCraft conditions flipped verdict on 14.5–16.5% of outputs and passed 43–62% of the wrong outputs across runs. The judge noticed the missing values in high- and low-scoring runs alike; what varied was whether it then evaluated the formulas or read stale values from the workbook difference. In `template:13_06`, one run took the input's cached gross profit as the submitted result and scored the corrected model 1.6/4, while the other two evaluated the formulas and gave 4/4. In `template:01_08`, GPT-5.6's bond model books the whole $15,900 final payment as coupon, so its extinguishment gain is $15,825 where the golden workbook has $825; from the uncached file the

RC judge read the formulas as isolating the principal and gave $4/4$, and from the recalculated copy the same rubric gives $2.7/4$ and names both errors. Recalculation cut the RUBRICCRAFT conditions' flips by 7.0–8.3 pp and their false passes by 15.9–17.8 pp, with every interval excluding zero; naive's flips did not change (+1.2 pp, $[-2.9, +5.0]$).

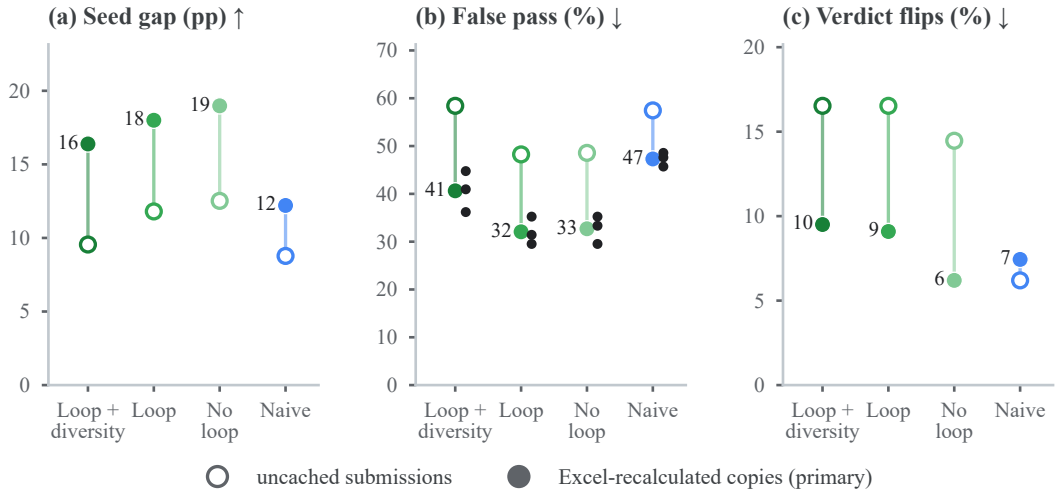


Figure 5: **Four rubric conditions, each grading the same 242 GPT-6 Astra outputs three times, from the uncached submissions (hollow) and from their Excel-recalculated copies (filled).** (a) Seed gap: mean grade on seed-passed minus seed-failed outputs. (b) False pass: share of the 105 outputs with fewer than 80% of target cells right that are graded as done; dots are the three recalculated runs. (c) Verdict flips: share of outputs whose pass/fail verdict changes across the three gradings. Recalculated evidence roughly halves the RUBRICCRAFT conditions' flips and removes a third of their false passes.

Table 1: Rubric conditions graded three times from recalculated copies of the same 242 GPT-6 Astra outputs (percent of maximum grade; ranges over the three runs). The seed gap is the mean grade on seed-passed minus seed-failed outputs, false pass uses the 105 outputs with fewer than 80% of target cells right, and flips are outputs whose pass/fail verdict changes across the three runs. No condition fails any of the 61 seed-passed outputs in any run.

Condition	Construction	Mean (runs)	Seed gap	False pass	Flips
Loop + diversity	repaired, 3-model attempts	86.0 (85.5–86.3)	16.4	40.6%	9.5%
Loop (RC)	repaired, Sol attempt	85.0 (84.9–85.2)	18.0	32.1%	9.1%
No loop	first-round R_0	84.2 (84.0–84.4)	19.0	32.7%	6.2%
Naive	prompt-only	89.6 (89.5–89.7)	12.2	47.3%	7.4%

C ADDITIONAL AGREEMENT RESULTS

Table 2 gives the full agreement and fidelity counts behind Figure 3a, including the strictest pass threshold. When wrong instead means fewer than half of the target cells right, RC passes 26.0% of those outputs and naive 33.7% ($[-14.7, -1.1]$ pp). Table 3 repeats the four main comparisons without the nine formula-comparison tasks, whose prepared copies are not recalculated; every interval still excludes zero. Figure 6 splits Figure 3b by task family.

D SEMANTIC COVERAGE PROTOCOL FOR SPREADSHEETBENCH V2

The protocol behind the coverage rows of Figure 3a reuses a published criterion-overlap implementation unchanged for the matching prompt, response validation, and metrics, and adapts only what

Table 2: Grading errors against the golden workbooks on the 259-task cohort (all three models' outputs). A rubric passes an output at a grade of at least $3.5/4$ and fails it below $3/4$; the seed passes and fails by its own rule. Intervals are 95% paired task-bootstrap intervals for RC minus naive, in pp. *Zero by construction: the reference is the seed's own cells.

Error	n	Seed	RC	Naive	RC−naive
Passes an output with $< 80\%$ of target cells right	312	0%*	29.5%	42.6%	$[-19.5, -6.7]$
Fails an output the seed passes	190	0%*	0.0%	2.6%	$[-5.2, -0.5]$
Fails an output with $\geq 95\%$ of target cells right	331	42.6%	2.4%	6.6%	$[-7.5, -1.6]$
Fails an output whose only errors are in preserved cells	37	100%	2.7%	8.1%	$[-13.9, 0.0]$
Ties or reverses a pair ≥ 5 pts apart in cell accuracy	258	84.9%	34.1%	53.9%	$[-28.2, -11.1]$
Strictest rubric setting, pass only at $4/4$ (seed row: its task bit)					
Passes an output with $< 80\%$ of target cells right (↓)	312	0%*	9.0%	12.8%	–
Passes an output with $\geq 95\%$ of target cells right (↑)	331	57.4%	63.4%	49.8%	–

Table 3: The main agreement and fidelity comparisons without the nine formula-comparison tasks (250 tasks), against the full cohort (259 tasks). Intervals are 95% paired task-bootstrap intervals for RC minus naive, in pp.

	All 259 tasks			Without formula mode (250)		
Measure	RC	Naive	Interval	RC	Naive	Interval
Wrong outputs passed	29.5%	42.6%	$[-19.5, -6.7]$	29.5%	44.4%	$[-21.7, -8.2]$
Seed-passed outputs failed	0.0%	2.6%	$[-5.2, -0.5]$	0.0%	2.1%	$[-4.7, -0.5]$
Near-complete outputs failed	2.4%	6.6%	$[-7.5, -1.6]$	2.1%	5.5%	$[-6.8, -0.6]$
Pairs tied or misordered	34.1%	53.9%	$[-28.2, -11.1]$	33.3%	55.4%	$[-30.8, -13.8]$

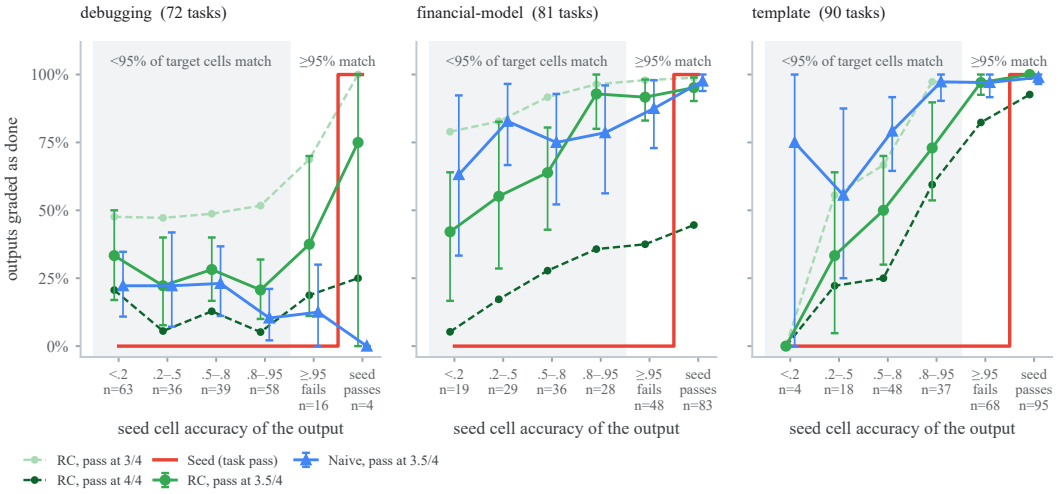


Figure 6: **Figure 3b split by task family.** Bins with few outputs (for example the four seed-passed debugging outputs) have wide intervals and should not be read individually. On financial-model and template tasks, RC at $3.5/4$ passes fewer low-accuracy outputs than naive; it matches naive on near-complete template outputs but passes fewer near-complete financial-model outputs that the seed fails (75% against 91% of 53). On debugging, where almost no output fixes every seeded error, both rubrics pass few low-accuracy outputs, and RC passes more of the seed-failed outputs with at least 95% of target cells right (40% against 10% of 20). Moving RC's threshold to $4/4$ lowers its curve most on financial-model tasks, where many outputs earn high but not perfect grades.

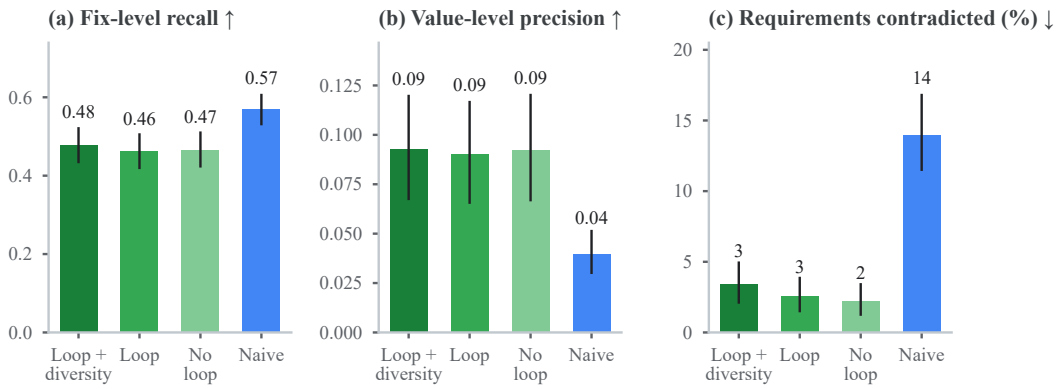


Figure 7: **RUBRICCRAFT rubrics agree with the seed's requirements; naive rubrics enumerate more of them.** Semantic coverage on the 238 tasks where every rubric set completed (task means with 95% task-bootstrap intervals; only full matches count). (a) Share of fix-level requirements (what must be fixed, and where) that a rubric fully covers. (b) Share of rubric criteria fully supported by value-level requirements (each fix with up to two key expected values). (c) Share of value-level requirements that a rubric explicitly contradicts.

SpreadsheetBench v2 requires. It was run on 243 tasks, all inside the primary cohort, and 238 of them completed for every rubric set (Figure 7).

- **Reference documents.** Per task: the instruction (context only), the published evaluator source, and a check sheet generated with the evaluator's own cell classification, listing each compared cell's input and expected value by sheet row, the mode flags, and the count of preserved cells. Adjacent cells of a row share one line; numbers are shown to seven significant digits, far finer than the evaluator's 1% tolerance. All 243 tasks fit the 240,000-character input bound without truncation.
- **Extraction.** The implementation's extraction instructions are used verbatim, followed by one SpreadsheetBench paragraph that sets the granularity. Both levels state one requirement per fix (a corrected error or restored structure in repair tasks, one built section in build tasks) and the preservation check once. Fix-level requirements carry no expected values, tolerances, or cell lists and cite only the check sheet's row headers, so the matcher never sees values; value-level requirements name at most two key expected values, and the comparison rule (tolerance and mode) is stated once as its own requirement. Every requirement must quote a verifier document exactly. The fix level extracted all 243 tasks (median 8 requirements, maximum 52); the value level extracted 241, with two failing quote validation after three attempts.
- **Generated criteria.** Checklist rubrics split at checklist items; naive rubrics, which are free-form, split at scored headings (one scored section per criterion) without discarding text; three naive rubrics with no headings are excluded.
- **Matching.** Both directions in batches of at most 24 targets against all candidates, with labels full, partial, none, contradicted, and uncertain; identical requests are answered once, so rubric conditions with identical text share results. The matching prompt's provenance sentence is replaced by a SpreadsheetBench one.
- **Choice of granularity.** Four extraction styles were compared on the same 12-task pilot before the full runs. References that list every compared cell, its expected value, and the tolerance gave recall 0.15–0.17 for RUBRICCRAFT and 0.085 for naive, because a rubric of four to six criteria spot-checks values rather than enumerating them. The implementation's instructions alone produced evaluator-mechanics requirements (for example, that all 1,598 modified cells match) and recall 0.10–0.14 for RUBRICCRAFT and 0.17 for naive. The two levels reported were the remaining styles. Value-free references cannot fully support criteria that state values, so fix-level precision is near zero by construction (0.005–0.016 on the full run) and is not reported.

The extracted reference is model-generated, not human-validated, and inherits the seed's golden-workbook errors.

E WORKED EXAMPLE

E.1 A REPRESENTATION MISMATCH THAT FAILS A CORRECT EDIT

This is the worked form of the case summarized in Figure 3c. It is included because every step is verifiable from the stored artifacts by direct cell inspection, without relying on any judge's opinion.

Task. Task `debugging:03_02` supplies a leveraged-buyout workbook and the instruction to "audit and fix this file thoroughly," listing the admissible error classes explicitly: formula logic errors, *hardcoded values that should be formulas*, broken references, data-validation issues, calculation errors, inconsistent conventions, and best-practice violations. Replacing a hardcoded value with an equivalent formula is therefore not an incidental liberty; it is one of the named objectives of the task.

What the model did. Among other edits, the submission replaced the constant in `LBO!C33` with a reference to the debt schedule, as shown in Table 4. The referenced cell is itself live, so the submitted workbook computes the same rate from its two components instead of restating it.

Table 4: Cell-level comparison for `debugging:03_02`, read directly from the stored golden and submitted workbooks. Only the audited cell differs, and it differs in representation rather than value: the two component cells are identical in both files, so both workbooks evaluate the rate to 0.053.

Cell	Golden	Submitted
`LBO!C33`	`0.053` (constant)	`='Ex 12 - Debt Schedule'!H9`
`Ex 12 - Debt Schedule!H9`	`=F9+G9`	`=F9+G9`
`Ex 12 - Debt Schedule!F9`	`0.018`	`0.018`
`Ex 12 - Debt Schedule!G9`	`0.035`	`0.035`

Since $0.018 + 0.035 = 0.053$, the submitted workbook and the golden workbook agree on the value of the audited cell. They differ only in whether that value is stated or derived, which is precisely the distinction the task asked the model to act on.

What each evaluator concluded. The seed evaluator shipped with the benchmark runs this task in formula-comparison mode. It marked all 21 modification cells correct, giving 100% modification accuracy, and failed the task on regression: 4,390 of 4,436 preserved cells matched, for 98.96% regression accuracy. The first reported regression error is `LBO!C33`, with expected `0.053` against submitted `='Ex 12 - Debt Schedule'!H9`. Because our wrapper promotes regression accuracy to a pass only at 0.998 or above, 98.96% is recorded as a task failure. The failure is therefore produced entirely by cells the model was *not* asked to change, under a comparator that treats a live reference to an equal value as a mismatch.

The RUBRICCRAFT rubric awarded full marks. Its highest-weighted criterion (3.0 of 10 points) asks the model to "replace duplicated hardcodes with formulas linked to authoritative cells" and states that "equivalent live-link formulas are acceptable," and the grader scored the submission $4/4$, or 100%. The naive rubric scored it $2.3/4$, or 57.5%: it credited the debt-schedule repairs but deducted chiefly for a missing interest-deduction cap schedule and for forecast roll-forwards left untouched, deductions we have not verified. Because the seed runs this task in formula-comparison mode, its prepared copy is not recalculated; the equivalence above is read from constants and needs no computed values.