SAME OUTPUTS, DIFFERENT WINNERS: EVALUATOR-DEPENDENT COMPARISONS IN CODE BENCHMARKS

Anonymous authors
Paper under double-blind review

ABSTRACT

AI coding research relies on benchmarks to compare code models and coding agents, assess progress, and select research baselines. Yet changing the evaluator can reverse statistically supported leads even when code outputs remain fixed: 60 of the 66 supported reversals we observed had point-estimate advantages of at least 2 percentage points in both opposing directions. To characterize this dependence, we hold tasks, system configurations, and generated outputs fixed and investigate evidence reinterpretation, criterion strengthening, and specification revision through evaluator contrasts in SWE-bench Verified, EvalPlus, and LiveCodeBench, respectively. Within each benchmark, joint inference over all system pairs and evaluator conditions provides a common evidence standard, separating support lost through an expanded inference family from support lost after evaluator changes. We find that only 38.5%–55.0% of originally supported leads retain support for their original direction across evaluators; among cross-evaluator support losses, roughly two-thirds gain support for the opposite direction. The consequences also depend on where the affected relations lie: a leading group can retain supported leads over all remaining systems while its internal winner remains unresolved, whereas a single winner-critical conflict can block a unique-winner claim. We therefore propose a shift from Rank-first to **Claim-first interpretation**: specify the target claim and its evaluation scope, identify the pairwise relations required to establish it, and determine which conclusion the available evidence supports. This framework guides progress assessment, baseline and shortlist selection, result reporting, and reassessment after evaluator updates. It connects benchmark scores and ranks to scientific conclusions that can be reexamined as evaluation conditions change.

1 INTRODUCTION

Code models and coding agents are increasingly compared on code benchmarks, whose scores and rankings guide judgments of progress and comparisons among systems (Raji et al., 2021; Jimenez et al., 2024; Jain et al., 2025). These scores, however, are not properties of the outputs themselves: they are produced by an evaluator that judges each output against its tests and rules, placing the evaluator between system outputs and comparative evidence (Figure 1). A leaderboard alone does not reveal how strongly its conclusions depend on the evaluator specification. But do conclusions about which system leads remain valid when the same outputs are judged under a different evaluator? Our results reveal that changing the evaluator alone can reverse statistically supported leads, even when tasks, system configurations, and generated outputs remain fixed.

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.

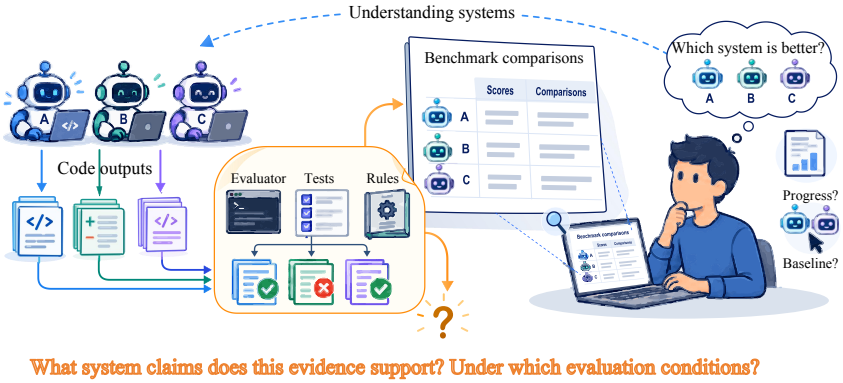

Figure 1: From code outputs to system claims: the evaluator's role in benchmark comparisons.

Benchmark comparisons often treat the evaluator as a fixed measurement layer. In practice, parsers and annotations are revised, as in UTBoost for SWE-bench (Yu et al., 2025); correctness requirements are strengthened through additional or stronger tests (Liu et al., 2023; Yu et al., 2026); and evaluation specifications evolve across benchmark versions, as in the LiveCodeBench snapshots examined here (Jain et al., 2025). The same frozen outputs can therefore yield different supported comparisons, including reversals in which different evaluators favor different systems.

To isolate evaluator effects, we fix the tasks, configured systems, and generated outputs, and vary only the post-generation evaluator. We investigate three evaluator operations—evidence reinterpretation, criterion strengthening, and specification revision—on SWE-bench Verified, EvalPlus, and LiveCodeBench, respectively. Joint inference over all system pairs and evaluators places every comparison under a common evidence standard and separates support lost through expansion of the inference family from support changes attributable to the evaluator.

Under this design, originally supported comparisons largely survive expansion of the inference family, yet only 38.5%–55.0% retain support across evaluators. About two-thirds of the cross-evaluator support losses gain support for the opposite direction. Moreover, 60 of the 66 reversals have point-estimate advantages of at least 2 percentage points (pp) in both opposing directions, so the phenomenon is not confined to near ties. The consequence further depends on where a relation sits in the comparison structure: a leading group can remain stable while its internal winner is unresolved, whereas a single conflict on a winner-critical relation is sufficient to block a unique-winner claim.

Evaluator dependence therefore matters not only through whether comparisons change, but also through which claims the affected relations can support. We consequently propose moving from Rank-first to **Claim-first interpretation**. Instead of reading system-level conclusions directly from leaderboard order, Claim-first interpretation specifies the target claim and its evaluation scope, identifies the pairwise relations required to support that claim, and asks how far the evidence on those relations supports it. Unique winners, leading sets, internal order, and shortlist membership therefore have distinct evidence requirements rather than being inferred from a single rank. This perspective applies to progress assessment, baseline and shortlist selection, result reporting, and reassessment after evaluator updates.

Our contributions are threefold:

- We reveal that, across three evaluator contrasts in code benchmarks, supported system comparisons can reverse, with about two-thirds of cross-evaluator support losses gaining support for the opposite direction.
- We introduce a fixed-output, joint-evidence framework spanning evidence reinterpretation, criterion strengthening, and specification revision, which isolates evaluator effects and judges all comparisons under a common evidence standard.
- We propose **Claim-first interpretation**, which maps system-level claims to the pairwise evidence they require, and derive guidance for progress assessment, baseline and shortlist selection, result reporting, and benchmark updates.

2 Evaluator Contrast Framework

2.1 Same Outputs, Different Evaluator Contrasts

We investigate how the evaluator itself changes system comparisons. A benchmark result, however, is jointly determined by the tasks, the configured systems, their generated outputs, and the evaluator. If these factors change together, a changed score or winner cannot be attributed specifically to the evaluator. Regenerating outputs is especially problematic, because generation variability then mixes changes in evaluation with changes in the outputs being evaluated. Identifying an evaluator-side effect therefore first requires holding the evaluated evidence fixed. To this end, we fix the tasks, configured systems, and complete generated patches or programs within each benchmark and apply every evaluator condition to the same outputs, so that each output forms a natural paired comparison across evaluators (Figure 2(a)). The result is a controlled evaluator contrast: the same evidence, judged under different evaluation conditions.

An evaluator, however, acts first on individual outputs, changing the verdicts of some of them. Only when these verdict changes are aggregated do they become changes in system-level scores. Such score changes need not alter a comparison: the pairwise gap between two systems moves only when the evaluator affects them to different degrees or in different directions. Our analysis therefore targets the evaluator-induced change in system comparisons under fixed evidence.

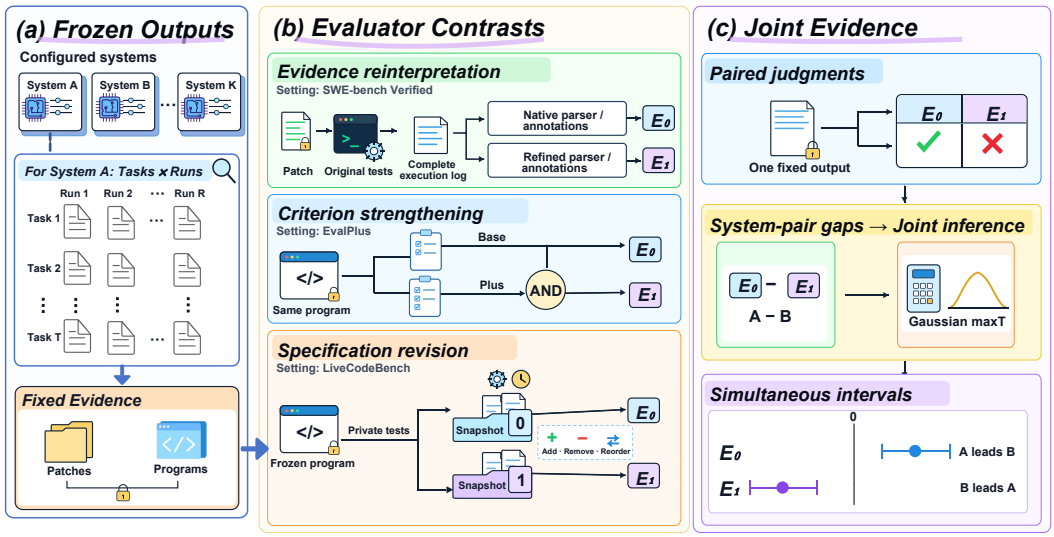

Figure 2: Frozen code outputs, evaluator contrasts, and joint evidence for system comparisons.

2.2 Evaluator Operations and Experimental Design

The evaluator is a key component through which a benchmark forms scores and comparisons, and consists of several interrelated evaluation steps. Changing the evaluator is therefore not necessarily a single operation; the change can occur at different positions along the evaluation path, and changes at different positions carry different meanings for an output's verdict. Reviewing code-benchmark evaluator designs, we focus on the following changeable components: execution-log parsing (Yu et al., 2025); annotation and verdict mapping (Yu et al., 2025); success criteria and required checks (Qi et al., 2015; Smith et al., 2015; Liu et al., 2023); test-suite composition (Liu et al., 2023; Jain et al., 2025); and checker and execution rules (Jimenez et al., 2024; Jain et al., 2025). We group these fine-grained changes into the three evaluator operations investigated here.

Evidence reinterpretation asks whether a system comparison changes when the rules for reading the same evidence change. The evidence itself is held fixed, and only the interpretation that maps evidence to a verdict changes. *Criterion strengthening* asks how a system comparison changes when the original success requirement is retained but an output must additionally satisfy new correctness

conditions. It adds necessary checks to the original requirement. *Specification revision* asks how a system comparison changes when an evaluator revises its tests or evaluation specification across versions, so that evaluation content may be added, removed, or adjusted. Such a change does not simply make evaluation stricter; rather, the evaluation specification as a whole is revised.

We instantiate each operation in an executable-code setting that provides a controlled and reproducible instance of the required evaluation change. Each benchmark instantiates an E0 $\rightarrow$ E1 contrast under the fixed-output principle of Section 2.1. *Evidence reinterpretation* requires fixed execution evidence and alternative rules that map the same evidence to verdicts. SWE-bench Verified provides both: complete execution logs can be frozen and evaluated using the original or revised parser and annotation rules (Jimenez et al., 2024; Yu et al., 2025). *Criterion strengthening* requires a base success condition to which additional correctness checks can be added. EvalPlus provides this structure through its Base and Plus tests (Liu et al., 2023). *Specification revision* requires identifiable evaluator versions with traceable changes in evaluation content. We use LiveCodeBench (Jain et al., 2025) and align two private-test snapshots. In the snapshots analyzed here, tests were added, removed, or reordered; we reevaluate the same programs with the checker and resource policy held fixed. Together, these benchmarks span repository-level software engineering, code generation, and algorithmic programming.

All three benchmarks use ten attempts per configured system–task cell. SWE-bench Verified uses 500 tasks and 12 systems with frozen *On Randomness* patches (Bjarnason et al., 2026); E0 and E1 apply the native and revised parser/annotation rules to the same replayed logs, without augmented tests (Yu et al., 2025). EvalPlus uses 164 HumanEval+ and 378 MBPP+ tasks with eight systems; E0 uses Base tests and E1 requires both Base and Plus tests (Liu et al., 2023). LiveCodeBench uses 16 systems over 400 aligned tasks (387 primary tasks and 13 controls) with historical frozen programs; E0 and E1 are two published private-test snapshots (Jain et al., 2025). Exact versions, runner settings, generation protocols, provenance, and execution/retry rules are given in Appendix B.

2.3 Joint Evidence for System Comparisons

To determine whether an evaluator change alters a system-level comparison, realized scores or pairwise gaps on the frozen outputs are not sufficient. Fixing outputs resolves attribution, but each system–task cell contains only finitely many generation attempts. We therefore target average performance under the same fixed tasks and generation protocol. Let T denote the number of tasks, K the number of systems, $\mathcal{E} = \{\mathrm{E0}, \mathrm{E1}\}$ the evaluator family, and $\mathcal{P} = \{(i,j) : 1 \le i < j \le K\}$ the set of system pairs. Attempt r by system s on task t produces output $O_{s,t,r}$ and binary judgment $Y_{s,t,r,e} = g_{t,e}(O_{s,t,r})$. With fixed, equally weighted tasks and expectation over repeated generation under the same protocol, we define

$$\mu_{s,e} = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[Y_{s,t,r,e}], \qquad \Delta_{ij,e} = \mu_{i,e} - \mu_{j,e}. \tag{1}$$

Under sampling model B0 (Appendix A.1), the finite-sample estimate $\widehat{\Delta}_{ij,e}$ is uncertain, so its sign alone does not establish directional support.

To place directional claims across system pairs and evaluators under one evidence standard, we perform joint inference over $\mathcal{G} = \mathcal{P} \times \mathcal{E}$. The comparisons share evidence: judgments of the same output are paired across evaluators, while pairwise contrasts sharing a system share estimation error. Our joint Gaussian construction preserves both dependencies (Appendix A.2) and yields a common critical value and simultaneous interval for every $\Delta_{ij,e}$:

$$\begin{aligned} M^*_{\mathcal{G}} &= \max_{(i,j,e)\in\mathcal{G}} \left| \frac{D^*_{ij,e}}{\widehat{\mathrm{se}}_{ij,e}} \right|, \qquad c_{\mathcal{G}} = Q_{1-\alpha}(M^*_{\mathcal{G}} \mid D), \\ I^{\mathcal{G}}_{ij,e} &= [L^{\mathcal{G}}_{ij,e}, U^{\mathcal{G}}_{ij,e}] = \left[\widehat{\Delta}_{ij,e} - c_{\mathcal{G}}\widehat{\mathrm{se}}_{ij,e}, \widehat{\Delta}_{ij,e} + c_{\mathcal{G}}\widehat{\mathrm{se}}_{ij,e} \right]. \end{aligned} \tag{2}$$

Here D denotes the observed records, $D^*_{ij,e}$ a drawn contrast error, $\widehat{\mathrm{se}}_{ij,e}$ the estimated standard error, and $Q_{1-\alpha}(\cdot \mid D)$ the conditional $(1-\alpha)$-quantile at nominal level $\alpha = 0.05$. An interval entirely above zero supports i leading j, one entirely below zero supports the reverse direction, and one containing zero provides no directional support. Coverage is asymptotic under B0 (Appendix A.2); invalid analyses abstain.

To distinguish evaluator-induced support changes from losses caused only by expanding the inference family, we use the $\mathcal{B} \to \mathcal{F} \to \mathcal{H}$ pipeline. $\mathcal{B}$ is the set of directions supported under E0-only inference. Re-reading E0 under the full pair-by-evaluator family gives $\mathcal{F} \subseteq \mathcal{B}$; hence $\mathcal{C} = \mathcal{B} \setminus \mathcal{F}$ is family-expansion loss and is not attributed to evaluator change. Within $\mathcal{F}$, $\mathcal{H}$ contains comparisons whose original direction remains supported under E1, so $\mathcal{F} \setminus \mathcal{H}$ contains the evaluator-dependent support losses. We partition these losses into supported reversals $\mathcal{R}$, where E1 supports the opposite direction; supported attenuation $\mathcal{A}$, where the original gap is directly supported to have narrowed without reversal; and other loss $\mathcal{O}$. Retention is $|\mathcal{H}|/|\mathcal{B}|$, while the reversal fraction is $|\mathcal{R}|/|\mathcal{F} \setminus \mathcal{H}|$. Support under one evaluator and nonsupport under the other alone establishes neither attenuation nor reversal. Appendix A.3 gives the oriented loss definitions.

3 Results

3.1 Same Outputs, Opposing Supported Comparisons

Originally supported comparisons largely survived broader inference, but many failed to retain support across evaluator operations (Figure 3(a)). Family expansion caused almost no loss of support: only 1, 1, and 0 E0-supported comparisons lost support under evidence reinterpretation, criterion strengthening, and specification revision, respectively. In contrast, changing the evaluator caused a further 26, 15, and 59 comparisons to lose support. Thus, the substantial erosion appeared at the cross-evaluator stage rather than from expansion of the inference family. Losing that common support, however, does not tell us what replaced it.

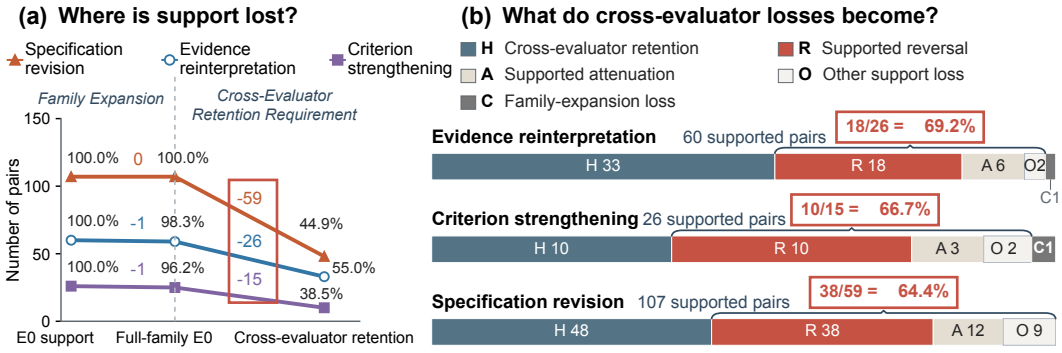

Figure 3: **(a)** Support retained after family expansion and across evaluator operations. **(b)** Composition of the original E0-supported comparisons under each evaluator operation.

Most cross-evaluator losses did not merely become unsupported: they gained statistical support for the opposing system (Figure 3(b)). Supported reversals constituted roughly two-thirds of these losses under all three evaluator operations: 69.2% for evidence reinterpretation, 66.7% for criterion strengthening, and 64.4% for specification revision. In each reversal, the full-family E0 interval and the alternative-evaluator interval supported opposite directions for the same pairwise difference. This is opposing affirmative support, rather than a significant-versus-nonsignificant mismatch. These reversals were also not confined to near ties: 60 of 66 E0-anchored reversals had point-estimate advantages of at least 2 pp in both opposing directions. Evaluator specification therefore affected not only whether the original lead remained supported, but which system the evidence favored.

3.2 The Location of Evaluator Dependence Determines Which Claims Survive

The aggregate comparison changes in Section 3.1 do not by themselves identify which system-level conclusions are affected. To locate those consequences, we classify each system pair by its directional support across E0 and E1: ROBUST if both evaluators support the same direction, CONFLICTED if they support opposite directions, CONDITIONAL if only one evaluator supports a direction, and UNRESOLVED if neither does. Under evidence reinterpretation, S01 and S02 had

ROBUST leads in all 20 comparisons against the other ten systems, while their mutual relation was UNRESOLVED. The evidence therefore supports a two-system leading set, but not a unique winner or an internal order between its members. Specification revision reveals a different pattern: 7/10 relations within the E0 top five were ROBUST, yet the sole CONFLICTED relation joined first- and second-ranked S01 and S03. Thus, even when most nearby relations remain ROBUST, a single conflict in a winner-critical position can block a unique-winner conclusion. The consequence of evaluator dependence therefore depends on where the affected relation lies, not simply on how many relations change.

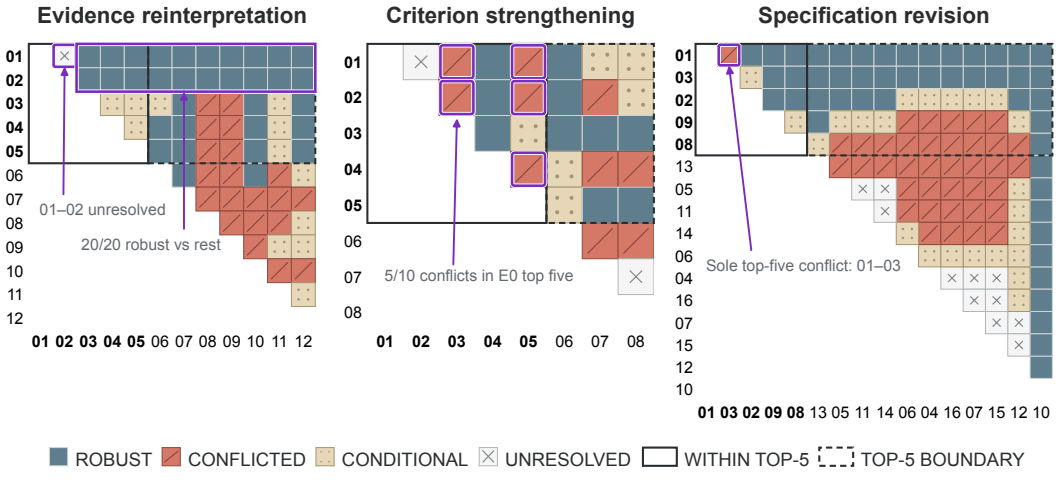

Figure 4: Pairwise evidence-state maps under evidence reinterpretation, criterion strengthening, and specification revision.

This dependence on location extends beyond the unique-winner case because different system-level claims rely on different parts of the comparison structure. A leading-set claim depends on member–outsider relations, internal order on within-group relations, top-k or shortlist membership on boundary-crossing relations, and a unique-winner claim on the comparisons that separate the candidate from its rivals. Figure 4 reveals these differences empirically. Under criterion strengthening, 5/10 relations within the E0 top five were CONFLICTED, placing much of the evaluator dependence inside the top-five set and directly affecting its internal order. Under specification revision, 14 relations were CONFLICTED and 10 CONDITIONAL among the 55 relations crossing the top-five boundary, so evaluator dependence also reached the membership boundary itself. Evidence reinterpretation reveals a third pattern: the two-system leading set remained externally ROBUST, while the broader top-five boundary contained six CONFLICTED and four CONDITIONAL relations. A smaller leading-set claim can therefore remain supported even when a broader top-k membership claim does not. Across these patterns, the system-level consequence is determined by the role of the affected relations in the comparison structure; aggregate conflict counts or a single leaderboard ordering alone do not determine which claims remain supported.

3.3 Sensitivity to Evidence and Task Scope

Reducing the available evidence weakened aggregate support, but the effects were not uniform across evaluator operations. Recomputing the full-family inference over 200 paired subsets per condition, five repeats retained a median 12/18, 8/10, and 18/38 original reversals under evidence reinterpretation, criterion strengthening, and specification revision, respectively (Figure 5(a)). At the same time, some claim-critical relations were considerably more stable: all 20 external ROBUST relations supporting the two-system leading group under evidence reinterpretation persisted in every checked condition, while the winner-critical reversal under specification revision persisted in 200/200 seven-repeat and 147/200 five-repeat subsets. These results indicate that sensitivity in aggregate reversal counts does not necessarily translate directly into the same sensitivity at the claim level.

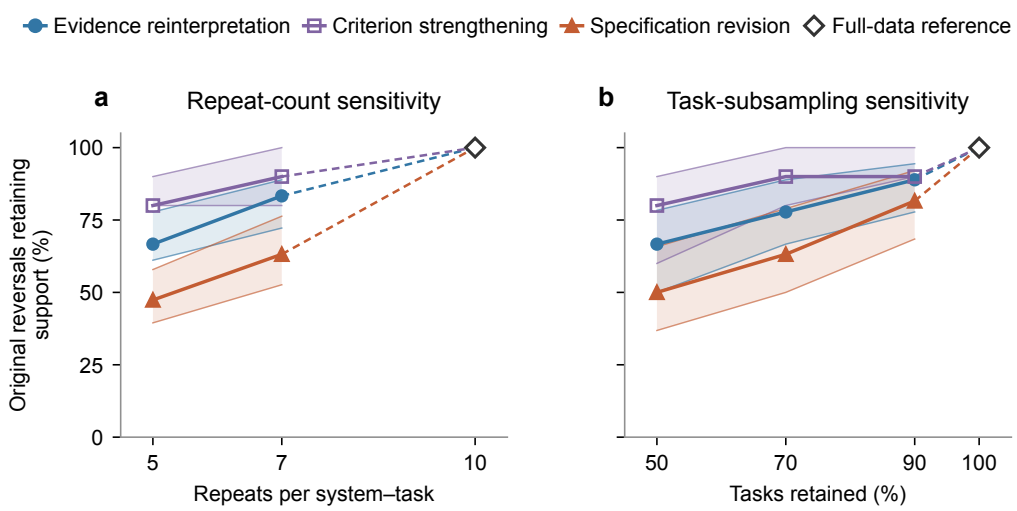

Figure 5: **(a)** Repeat-count sensitivity. **(b)** Sensitivity to the fraction of tasks retained. Points show median retention of the original 18, 10, and 38 reversals for evidence reinterpretation, criterion strengthening, and specification revision, respectively. Shading shows empirical 10th–90th percentiles across 200 paired subsets, not confidence intervals. Diamonds mark the full-data reference, excluded from subset summaries; dashed segments connect to that reference.

Repeat reduction and changes in task composition exposed different sources of variation. For the winner-critical relation under specification revision, all 200 five-repeat subsets retained the point reversal, although 53 lost reverse-side interval support; the median reverse gap changed only from 4.99 to 4.91 pp, while the interval half-width increased from 3.12 to 4.42 pp. Reducing repeats therefore mainly weakened statistical support through lower precision. In separate task-composition comparisons with repeat and task counts fixed, median reversal retention shifted from 9 to 7 across criterion-strengthening task compositions and from 28 to 17 across specification-revision task compositions. Changes in task composition therefore affect the evaluation scope itself, rather than only the precision of a fixed comparison.

4 From Rank-First to Claim-First: Interpreting and Reassessing Benchmark Evidence

4.1 Why Claim-First?

A leaderboard ordering compresses multiple pairwise evidence relations into a single rank representation, even though different system-level claims depend on different parts of that comparison structure. A unique-winner claim, a leading-set claim, an internal ordering, and shortlist membership therefore need not be supported by the same relations. Rank-first interpretation starts from the ordering and then infers what system-level conclusion it licenses, which can produce a claim stronger than the underlying evidence supports. We instead reverse this direction. Claim-first interpretation begins with the scientific claim and its evaluation scope, identifies the pairwise relations required to establish that claim, and then asks what conclusion those relations actually support. Its purpose is not to produce another ranking, but to align the strength of the system-level conclusion with the evidence that directly supports it.

The procedure is simple: start from the target claim and its scope, identify the pairwise relations required to establish it, examine the evidence for those relations, and report the strongest conclusion they support. For progress over a baseline, the required relation is the comparison between the target system and that baseline. A unique-winner claim requires the candidate to have the required directional support over every relevant rival in the stated system scope. A leading-set claim instead depends on member–outsider relations: every member must be separated from every system outside the set, without requiring a unique winner or complete order among the members themselves. Internal-order claims depend on the corresponding within-group relations, while top-k or shortlist

membership depends on the relations that cross the proposed boundary rather than on a complete ordering within the group. Once these required relations are identified, their evidence under the stated evaluation scope determines how strong a claim can be made. For a claim intended to hold across both evaluators, a CONFLICTED, CONDITIONAL, or UNRESOLVED required relation limits that claim. A conflict outside the required relation set does not by itself invalidate it. Claim-first interpretation therefore returns the strongest system-level conclusion supported by the relevant evidence rather than forcing a complete leaderboard ordering.

4.2 How Should Benchmark Evidence Be Used in Practice?

Claim-first interpretation turns benchmark use into three concrete tasks. In each application, specify the target claim and evaluation scope, inspect the required pairwise relations, and report the conclusion warranted by their evidence.

- **Research decisions: assess progress and select candidates.** For progress, examine the target system's comparison with the stated baseline under the stated evaluation conditions. For baseline or shortlist selection, inspect the candidate–outsider relations separately from the candidates' internal order. If several systems form a supported leading set while their internal order remains unresolved, retain that set rather than designate a unique winner. Report whether the evidence establishes progress, candidate membership, or internal ordering; these are distinct conclusions.
- **Result reporting: state the claim and its conditions.** Report the system-level conclusion together with its system, task, and evaluator scope. State a unique-winner conclusion only when all required comparisons support it; otherwise report the supported leading set or the remaining uncertainty. Qualify conclusions that hold under only one evaluator. This preserves the distinction between an observed leaderboard position and a conclusion established by the comparative evidence.
- **Benchmark updates: reassess the required comparisons.** Identify the pairwise relations underlying previously reported conclusions and reevaluate them after changes to tests, parsers, annotations, or other evaluator specifications. When historical outputs are available, keep them fixed to isolate the evaluator-side effect. Report which claim-critical directions retain support, lose support, or gain support in the opposite direction, then update the corresponding conclusions. This tells downstream users which conclusions remain usable and which require reinterpretation.

5 Related Work

Code evaluation and evaluator specifications. Benchmark conclusions depend on what tasks and evaluation criteria actually measure (Bowman & Dahl, 2021; Raji et al., 2021). In code evaluation, these choices are instantiated through executable tests, repository-level execution, and rules that map generated programs or patches to success judgments (Chen et al., 2021; Jimenez et al., 2024; Jain et al., 2025). Prior work indicates that incomplete tests can accept incorrect outputs, stronger test suites can change measured correctness and leaderboard order, and revised parsers or annotations can reclassify the same execution evidence (Qi et al., 2015; Smith et al., 2015; Liu et al., 2023; Yu et al., 2025; Yu et al., 2026). Related meta-evaluation research examines judge reliability and alternative correctness signals (Zheng et al., 2023; Zeng et al., 2024; Tan et al., 2025; Jiang et al., 2026; Yang et al., 2024). We focus on how such post-generation evaluation changes propagate from fixed code outputs to supported system comparisons.

Uncertainty in system comparisons. Benchmark comparisons can vary for reasons beyond evaluator specification, including search or training configuration, finite or repeated runs, generation variability, and imperfect judges (Dodge et al., 2019; Bouthillier et al., 2021; Agarwal et al., 2021; Bjarnason et al., 2026; Hariri et al., 2026; Feng et al., 2026). Research on statistical comparisons and leaderboards further indicates that conclusions depend on the inferential target, available evidence, ranking uncertainty, observation selection, and benchmark scope (Dror et al., 2018; Demšar, 2006; Neuhof & Benjamini, 2026; Huang et al., 2026; Yauney et al., 2026). Harness configurations can also change leaderboard outcomes, although such changes may affect model responses before post-generation evaluation (Parupudi, 2026). A changed score or rank therefore establishes instability,

but does not by itself identify an evaluator-side effect. We therefore separate attribution from inference: fixed outputs isolate evaluator changes, while joint evidence determines whether the resulting pairwise directions are statistically supported.

Evaluation sensitivity and comparison interpretation. Recent work has examined how leaderboard conclusions depend on evaluation choices such as harness configuration, scoring targets, observation selection, benchmark scope, and comparison specification (Parupudi, 2026; Panthi & Abdelfattah, 2026; Huang et al., 2026; Yauney et al., 2026; Huang, 2026). Together, these findings indicate that leaderboard conclusions can depend materially on how evaluation is defined and which evidence is used. Our focus is whether statistically supported system comparisons remain stable when identical code outputs are judged under different post-generation executable evaluators. By separating evaluator-induced support changes from losses caused by broader inference, we characterize which comparative conclusions remain supported across evaluation conditions.

6 Conclusion

Evaluator changes can alter not only whether a system comparison remains supported, but which system the evidence supports, even when the underlying code outputs are held fixed. Across evidence reinterpretation, criterion strengthening, and specification revision, cross-evaluator retention dropped substantially, and roughly two-thirds of the resulting support losses became supported reversals rather than simple losses of significance. To isolate these effects, we used fixed-output controlled contrasts and joint inference over system pairs and evaluator conditions, placing all directional comparisons under a common evidence standard while separating evaluator-dependent changes from losses caused by expanding the inference family. The resulting evidence also reveals that the consequence of evaluator dependence is structural: a leading group can remain supported while its internal winner is unresolved, whereas a single conflict on a winner-critical relation can block a unique-winner claim. These findings motivate a shift from Rank-first to Claim-first interpretation, in which the target claim and its scope determine the pairwise relations that must be supported, and the strongest system-level conclusion is matched to the evidence those relations provide. This perspective guides more precise progress assessment, baseline and shortlist selection, result reporting, and reassessment after evaluator updates. Our empirical evidence is drawn from executable-code evaluation; future work can examine its applicability across broader AI evaluation settings, tasks, modalities, and benchmark ecosystems, and can further investigate how previously supported conclusions should be tracked and reassessed as benchmarks and evaluator specifications evolve.

AI Use Statement

Generative AI tools were used to assist with manuscript polishing and revision, analysis-code development, and literature search and organization. All AI-assisted content, code, and references were reviewed and verified by the authors. The authors retain full responsibility for the analyses and final content of the paper.

Ethics Statement

This work investigates evaluator-dependent comparisons using existing code benchmarks and generated code outputs. It does not involve human subjects or the collection of personal or sensitive data. We are not aware of additional ethical concerns specific to this work beyond the general considerations associated with benchmark-based evaluation and interpretation.

Reproducibility Statement

To support reproducibility, Section 2.1 describes the fixed-output design and specifies which experimental factors are held fixed across evaluator conditions. Section 2.2 reports the benchmark settings, output sources, task and system coverage, repeated attempts, and evaluator contrasts for SWE-bench Verified, EvalPlus, and LiveCodeBench. Section 2.3 defines the estimands, dependence structure, Gaussian maxT procedure, and decision rules used to construct simultaneous intervals and classify comparison outcomes, while distinguishing missing judgments and analysis abstentions from unre-

solved comparisons. We will publicly release the code, frozen outputs, and analysis scripts upon acceptance of the paper.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320, 2021.

Bjarni Haukur Bjarnason, André Silva, and Martin Monperrus. On randomness in agentic evals. In *ICLR 2026 Workshop on Agents in the Wild*, 2026. doi: 10.48550/arXiv.2602.07150. URL https://arxiv.org/abs/2602.07150.

Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Naz Sepah, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Dmitriy Serdyuk, Tal Arbel, Chris Pal, Gaël Varoquaux, and Pascal Vincent. Accounting for variance in machine learning benchmarks. In *Proceedings of Machine Learning and Systems*, volume 3, 2021.

Samuel R. Bowman and George Dahl. What will it take to fix benchmarking in natural language understanding? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4843–4855, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.385.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. doi: 10.48550/arXiv.2107.03374. URL https://arxiv.org/abs/2107.03374.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1):1–30, 2006.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2185–2194, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1224.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1383–1392, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1128.

Chen Feng, Minghe Shen, Ananth Balashankar, Carsten Gerner-Beuerle, and Miguel R. D. Rodrigues. Noisy but valid: Robust statistical evaluation of LLMs with imperfect judges. In *The Fourteenth International Conference on Learning Representations*, 2026.

Mohsen Hariri, Amirhossein Samandar, Michael Hinczewski, and Vipin Chaudhary. Don't Pass@k: A Bayesian framework for large language model evaluation. In *The Fourteenth International Conference on Learning Representations*, 2026.

Jenny Huang, Yunyi Shen, Dennis Wei, and Tamara Broderick. Dropping just a handful of preferences can change top large language model rankings. In *The Fourteenth International Conference on Learning Representations*, 2026.

Wei-Jung Huang. What does an LLM-agent leaderboard rank actually compare? In *The 13th International Conference on Data Science and Advanced Analytics (IEEE DSAA 2026)*, 2026. doi: 10.48550/arXiv.2609.07785. URL https://arxiv.org/abs/2609.07785. To appear.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

Hongchao Jiang, Yiming Chen, Yushi Cao, Hung-yi Lee, and Robby T. Tan. CodeJudgeBench: Benchmarking LLM-as-a-judge for coding tasks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19416–19448, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.888.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024.

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*, volume 36, pp. 21558–21572, 2023. doi: 10.52202/075280-0943.

Bitya Neuhof and Yuval Benjamini. Quantifying ranking uncertainty in LLM benchmarks. In *ICML 2026 Hypothesis Testing Workshop*, 2026. doi: 10.48550/arXiv.2607.16259. URL https://arxiv.org/abs/2607.16259.

Sugam Panthi and Rabab Abdelfattah. Same ranking, different winner: How scoring targets shape LLM memory benchmarks. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026. doi: 10.48550/arXiv.2605.24060. URL https://arxiv.org/abs/2605.24060. To appear.

V. S. Raghu Parupudi. There is no neutral harness: Modern LLM leaderboards are manufactured by config-fragile items. *arXiv preprint arXiv:2608.21382*, 2026. doi: 10.48550/arXiv.2608.21382. URL https://arxiv.org/abs/2608.21382.

Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pp. 24–36. Association for Computing Machinery, 2015. doi: 10.1145/2771783.2771791.

Inioluwa Deborah Raji, Emily M. Bender, Amandalynne Paullada, Emily Denton, and Alex Hanna. AI and the everything in the whole wide world benchmark. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Edward K. Smith, Earl T. Barr, Claire Le Goues, and Yuriy Brun. Is the cure worse than the disease? Overfitting in automated program repair. In *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 532–543. Association for Computing Machinery, 2015. doi: 10.1145/2786805.2786825.

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 *The Thirteenth International Conference on Learning Representations*, 2025.

Guang Yang, Yu Zhou, Xiang Chen, and Xiangyu Zhang. CodeScore-R: An automated robustness metric for assessing the functional correctness of code synthesis. *Journal of Computer Research and Development*, 61(2):291–306, 2024. doi: 10.7544/issn1000-1239.202330715.

Gregory Yauney, Shahzaib Warraich, and Swabha Swayamdipta. How reliable is language model micro-benchmarking? In *The Fourteenth International Conference on Learning Representations*, 2026.

Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. UTBoost: Rigorous evaluation of coding agents on SWE-bench. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3762–3774, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.189.

Boxi Yu, Yang Cao, Yuzhong Zhang, Liting Lin, Junjielong Xu, Zhiqing Zhong, Qinghua Xu, Guancheng Wang, Jialun Cao, Shing-Chi Cheung, Pinjia He, and Lionel Briand. SWE-ABS: Adversarial benchmark strengthening exposes inflated success rates on test-based benchmark. In *International Conference on Machine Learning*, 2026.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. Evaluating large language models at evaluating instruction following. In *The Twelfth International Conference on Learning Representations*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2020. Datasets and Benchmarks Track.

A Statistical Details for Joint System Comparisons

A.1 Sampling Model and Covariance Estimation

Section 2.3 treats the benchmark tasks as fixed and equally weighted, and places the sampling expectation over repeated generation under the same system–task protocol. For each system s and task t, let $R_{s,t}$ denote the number of complete generation attempts and collect the two evaluator judgments for attempt r into

$$\mathbf{Y}_{s,t,r} = \left(Y_{s,t,r,\mathrm{E0}}, Y_{s,t,r,\mathrm{E1}}\right)^\top.$$

The basic sampling model B0 used throughout the joint analysis is as follows. Conditional on the fixed task definitions and generation protocols, complete attempts are independent and identically distributed within each system–task cell; generation errors are independent across distinct system–task cells; cell distributions may differ across systems and tasks; and the two evaluator coordinates of the same output are kept as a paired vector rather than treated as independent observations. Thus B0 permits system- and task-specific success probabilities and evaluator dependence within an output while using the complete attempt as the repeat unit.

Write

$$\mathbf{m}_{s,t} = \mathbb{E}[\mathbf{Y}_{s,t,r}], \qquad \Sigma_{s,t} = \mathrm{Cov}(\mathbf{Y}_{s,t,r}),$$

and define the cell mean and sample covariance by

$$\bar{\mathbf{Y}}_{s,t} = \frac{1}{R_{s,t}} \sum_{r=1}^{R_{s,t}} \mathbf{Y}_{s,t,r}, \qquad S_{s,t} = \frac{1}{R_{s,t}-1} \sum_{r=1}^{R_{s,t}} (\mathbf{Y}_{s,t,r} - \bar{\mathbf{Y}}_{s,t})(\mathbf{Y}_{s,t,r} - \bar{\mathbf{Y}}_{s,t})^\top.$$

The system-level estimator corresponding to Eq. (1) is

$$\widehat{\boldsymbol{\mu}}_s = \frac{1}{T} \sum_{t=1}^{T} \bar{\mathbf{Y}}_{s,t},$$

with covariance

$$V_s = \mathrm{Cov}(\widehat{\boldsymbol{\mu}}_s) = \frac{1}{T^2} \sum_{t=1}^{T} \frac{\Sigma_{s,t}}{R_{s,t}}, \qquad \widehat{V}_s = \frac{1}{T^2} \sum_{t=1}^{T} \frac{S_{s,t}}{R_{s,t}}. \tag{A.1}$$

Under B0, $\widehat{V}_s$ is unbiased for V_s. The off-diagonal entry of $\widehat{V}_s$ retains the dependence between the E0 and E1 judgments of the same output.

Let $\widehat{\boldsymbol{\Delta}}_{ij} = \widehat{\boldsymbol{\mu}}_i - \widehat{\boldsymbol{\mu}}_j$ denote the two-evaluator gap vector for systems i and j. Because generation errors are independent across systems under B0,

$$\mathrm{Cov}(\widehat{\boldsymbol{\Delta}}_{ij}, \widehat{\boldsymbol{\Delta}}_{kl}) = \mathbf{1}_{i=k} V_i - \mathbf{1}_{i=l} V_i - \mathbf{1}_{j=k} V_j + \mathbf{1}_{j=l} V_j. \tag{A.2}$$

This identity captures the second dependence structure in the main analysis: pairwise comparisons that share a system also share that system's estimation error. In particular,

$$\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\Delta}}_{ij}) = \widehat{V}_i + \widehat{V}_j,$$

and the evaluator-specific standard errors used in Eq. (2) are obtained from the corresponding diagonal entries.

A.2 Joint Gaussian Construction and Simultaneous Coverage

Conditional on the observed records D, the joint Gaussian construction draws one error vector for each system,

$$\mathbf{G}_s^* \mid D \sim N(\mathbf{0}, \widehat{V}_s), \qquad s = 1, \ldots, K,$$

independently across systems, and reuses that same draw in every pair containing system s. The contrast error for pair (i, j) is

$$\mathbf{D}_{ij}^* = \mathbf{G}_i^* - \mathbf{G}_j^*, \qquad D_{ij,e}^* = [\mathbf{D}_{ij}^*]_e. \tag{A.3}$$

The within-vector covariance of $\mathbf{G}_s^*$ preserves pairing across evaluator coordinates, while reusing $\mathbf{G}_s^*$ across all pairs containing s preserves the shared-system covariance in Eq. (A.2). Consequently,

the standardized coordinates $D^*_{ij,e}/\widehat{\mathrm{se}}_{ij,e}$ entering the maximum statistic in Eq. (2) are generated jointly under the same dependence structure as the estimated pairwise gaps.

The coverage statement in Section 2.3 is asymptotic for the prespecified finite family $\mathcal{G} = \mathcal{P} \times \mathcal{E}$. A compact justification is as follows. Consider a sequence of fixed-task designs with fixed numbers of systems and evaluators, binary outcomes, complete-attempt counts bounded below by two, and independent system–task cell estimation errors as specified by B0. Let $\widehat{\theta}_T$ stack all system-by-evaluator mean estimators and let V_T denote its covariance. If

$$TV_T \to \Omega$$

for a positive-semidefinite matrix Ω, and every contrast that is to be directionally certified has positive limiting variance, boundedness of the cell contributions gives the corresponding multivariate triangular-array central limit theorem,

$$\sqrt{T}\,(\widehat{\theta}_T - \theta_T) \Rightarrow N(\mathbf{0}, \Omega). \tag{A.4}$$

The same boundedness and independence conditions give consistency of the block covariance estimator formed from the $\widehat{V}_s$. The induced studentized pairwise-contrast vector therefore converges jointly to its Gaussian limit. The conditional Gaussian construction above uses the plug-in covariance of that same vector, so its standardized maximum converges to the limiting maximum distribution. At continuity points of that distribution, the conditional $(1-\alpha)$ critical value in Eq. (2) converges to the corresponding limiting quantile. Hence the simultaneous intervals attain asymptotic family-wise coverage $1-\alpha$ for the fixed finite comparison family specified in the main analysis. The limiting covariance may be singular; the requirement is positive limiting variance for any contrast on which directional certification is based.

A.3 Family-Expansion Bridge and Oriented Loss Definitions

The bridge from E0-only inference to the full evaluator family uses the same observed records, covariance estimates, and Gaussian draws. Let

$$\mathcal{G}_0 = \mathcal{P} \times \{\mathrm{E0}\}, \qquad \mathcal{G} = \mathcal{P} \times \{\mathrm{E0}, \mathrm{E1}\}.$$

For every shared Gaussian realization,

$$M^*_{\mathcal{G}_0} \le M^*_{\mathcal{G}},$$

because the maximum for $\mathcal{G}_0$ is taken over a subset of the coordinates in $\mathcal{G}$. Using the same conditional quantile rule therefore gives

$$c_{\mathcal{G}_0} \le c_{\mathcal{G}}. \tag{A.5}$$

The point estimate and standard error of an E0 contrast are unchanged when the family expands; only the common critical value can increase. Thus the full-family E0 interval contains the corresponding E0-only interval, and any direction supported after family expansion was also supported before expansion. This gives the monotonicity relation

$$\mathcal{F} \subseteq \mathcal{B},$$

so $\mathcal{C} = \mathcal{B} \setminus \mathcal{F}$ is attributable to family expansion rather than to the evaluator change. Reusing the same Gaussian draws makes this bridge draw-by-draw rather than introducing avoidable Monte Carlo differences between the two families.

For the remaining evaluator-dependent losses, it is useful to orient every comparison in the direction supported by E0. For $(i, j) \in \mathcal{F}$, define

$$d_{ij} = \begin{cases} +1, & \text{E0 supports } i \text{ leading } j, \\ -1, & \text{E0 supports } j \text{ leading } i, \end{cases} \qquad \widetilde{\Delta}_{ij,e} = d_{ij}\Delta_{ij,e}.$$

The oriented gap is therefore positive in the original supported direction. Its evaluator-induced narrowing is

$$\Gamma_{ij} = \widetilde{\Delta}_{ij,\mathrm{E0}} - \widetilde{\Delta}_{ij,\mathrm{E1}}. \tag{A.6}$$

Positive Γ_{ij} means that the original gap is smaller under E1.

The simultaneous intervals from Eq. (2) yield oriented endpoints without introducing a new comparison family. Define

$$(\widetilde{L}_{ij,e}, \widetilde{U}_{ij,e}) = \begin{cases} (L^{\mathcal{G}}_{ij,e}, U^{\mathcal{G}}_{ij,e}), & d_{ij} = +1, \\ (-U^{\mathcal{G}}_{ij,e}, -L^{\mathcal{G}}_{ij,e}), & d_{ij} = -1. \end{cases}$$

On the simultaneous-coverage event, $\widetilde{\Delta}_{ij,e} \in [\widetilde{L}_{ij,e}, \widetilde{U}_{ij,e}]$ for both evaluators, and therefore

$$\Gamma_{ij} \geq \widetilde{L}_{ij,\mathrm{E0}} - \widetilde{U}_{ij,\mathrm{E1}} \equiv \underline{\Gamma}_{ij}. \tag{A.7}$$

Accordingly, within $\mathcal{F} \setminus \mathcal{H}$, a comparison is classified as a supported reversal $\mathcal{R}$ when E1 supports the direction opposite to d_{ij}; it is classified as supported attenuation $\mathcal{A}$ when it is not a reversal and $\underline{\Gamma}_{ij} > 0$; and all remaining evaluator-dependent support losses form $\mathcal{O}$. This construction makes attenuation an affirmative statement that the original gap narrowed, rather than an inference from significance under one evaluator and nonsignificance under the other.

B Experimental Details and Provenance

B.1 Shared Record and Execution Rules

Records were identified by configured system, task, and generation attempt, with both evaluator judgments linked to the same frozen output. Distinct generation attempts retained separate identities even when their output text was identical. Re-executing an output did not create a new generation attempt.

Infrastructure failures triggered automatic retries; records still awaiting a valid evaluation were subsequently rerun with the same output and configuration. The first complete, valid evaluation determined the recorded verdict. Functional failures, including evaluator-defined runtime errors and test timeouts, were retained as failures rather than retried to obtain a pass. Infrastructure interruptions were not converted into failure labels. These execution rules applied before the paired judgments entered the analysis.

B.2 SWE-bench Verified: Evidence Reinterpretation

We used the `ASSERT-KTH/agentic-evals-artifacts` artifact accompanying *On Randomness*. Configured-system identities retained the source model, scaffold, and decoding setting, while attempt identities retained the source run identifiers. Task definitions used `princeton-nlp/SWE-bench_Verified` revision `c104f840cc67`. Original tests were replayed using SWE-bench commit `4c21a4831d80`, with the corresponding repository environments and a 3,600-second execution timeout.

The native parser and annotations produced E0 from the frozen replay log. For E1, we used the refined parser and revised annotations from `CUHK-Shenzhen-SE/UTBoost` commit `4f32782517f2`, specifically `update_SWE_Bench/log_parsers.py` and `updated_parser_test_instance_dict_verified.json` in the same directory. Tasks without revised annotations retained their native annotations. Both interpretations used the same saved log; published source verdicts were not substituted for the replayed E0 judgments.

B.3 EvalPlus: Criterion Strengthening

Programs were collected during July–September 2026 through the model providers' first-party APIs, using the fixed per-system request configurations in Table B.1. Each request used a fresh context and returned one candidate, with no supplied system prompt or tools. The user message was "Please provide a self-contained Python script that solves the following problem in a markdown code block:" followed by two newline characters and the task prompt. Sampling parameters and seeds were omitted except for GLM-5.2, which used temperature 1.0 and top-p 0.95. Optional output caps were omitted; Claude Opus 5 used its required `max_tokens` value of 32,768. Transport retries retained the logical generation-attempt identity; completed responses were not resampled on the basis of code quality.

Table B.1: EvalPlus systems and explicit reasoning settings. Request IDs identify the provider APIs; the DeepSeek version used was V4 Pro-0813.

System	Requested model ID	Reasoning setting
EP-S01	`deepseek-v4-pro`	Thinking enabled; high effort
EP-S02	`MiniMax-M3`	Adaptive thinking
EP-S03	`qwen3.8-max`	Thinking enabled; effort and thinking budget omitted
EP-S04	`gpt-5.6-sol`	Medium effort; standard mode
EP-S05	`claude-opus-5`	Adaptive thinking; high effort
EP-S06	`gemini-3.7-flash`	Medium thinking level
EP-S07	`glm-5.2`	Thinking enabled; maximum effort
EP-S08	`kimi-k3`	Maximum effort

Provider-specific extraction retained the final visible answer, which was processed once by the sanitizer from EvalPlus v0.3.1, commit `e5d0ed0bab962`. The resulting program was frozen before evaluation. Evaluation used that same EvalPlus revision, HumanEval+ v0.1.10, and MBPP+ v0.2.0 in the repository's `python:3.11-slim` Docker environment. Base and Plus statuses were obtained within the same evaluation workflow, retaining the native input conversion, reference checks, and special-oracle behavior. The minimum time limit was 1 second and the reference-runtime multiplier was 4. The paired verdicts were read from `base_status` and the conjunction of `base_status` with `plus_status`.

B.4 LiveCodeBench: Specification Revision

Historical programs were taken from the `code_list` field of `livecodebench/code_generation_samples`, revision `79837278b7c5`. The selected source configurations were DeepSeek-R1-Preview, DeepSeek-R1-Lite-Preview, DeepSeek-V3, GPT-4-Turbo-2024-04-09, GPT-4O-2024-05-13, GPT-4O-2024-08-06, GPT-4O-mini-2024-07-18, Claude-3.5-Sonnet-20240620, Claude-3.5-Sonnet-20241022, Claude-3-Haiku, Gemini-Pro-1.5-002, Gemini-Flash-1.5-002, Gemini-Flash-2.0-Thinking, Gemini-Flash-2.0-Exp, Mistral-Large, and Codestral-Latest. These names identify historical source configurations. Programs were aligned by `question_id` and source sample index, preserving program text, test order, and repeated test cases. Cached `pass1_list` labels were not used as current evaluator outcomes.

The E0 and E1 test snapshots were `livecodebench/code_generation_lite` revisions `d9790747f3e2` and `2c35eba5dd5f`, respectively. Both used the checker from `LiveCodeBench/LiveCodeBench` commit `28fef95ea8c9`, with a timeout parameter of 16 seconds and no additional program-level memory cap. The same checker and execution environment were used for both conditions, retaining the public tests while changing the private-test snapshot. The primary task set comprised 359 test-addition tasks, 20 test-removal tasks, and 8 tasks with order-only changes. The unchanged-task controls defined in the main text were kept separate from the primary analysis.