WHOSE OUTCOME IS THE TEAM'S SCORE? AUDITING AGGREGATION AND SCORED-OBJECT BINDING IN MULTI-AGENT EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems increasingly coordinate multiple agents to solve complex tasks, yet their evaluation often compresses several agent executions into a single team score. Interpreting such a score requires two mappings that are rarely made explicit: how agent-level outcomes are aggregated into a team outcome, and which execution state or artifact is actually consumed by the grader. We formalize these as *agent-to-team aggregation* and *scored-object binding*, and introduce an *executable aggregation receipt* that makes this measurement path auditable from execution evidence. We apply the framework to a released multi-agent evaluation harness using source-code audits, controlled interventions, and model-driven runs. We then test whether the same audit transfers to an independently developed multi-agent framework. We find that nominally separate agents can become coupled through shared mutable state and that a reported team score can depend entirely on a single agent. In controlled interventions, changing the score-bound agent changes the reported result in all 24 paired comparisons, whereas changing either of the other agents changes none. We further show that unspecified aggregation creates a deterministic ambiguity whose width equals the agent-disagreement rate. On the independent framework, a source-pinned native adapter reconstructs the executed decision path while revealing a mismatch between the declared consensus semantics and the recorded accounting. Finally, we repair the harness by isolating agent states, explicitly aggregating all agent outcomes, and preserving scored-object lineage. Our results show that reliable multi-agent evaluation requires an auditable path from individual agent outcomes to the object actually scored.

1 INTRODUCTION

Multi-agent systems increasingly coordinate several language-model agents to search, debate, edit shared artifacts, or solve the same task in parallel. As these systems become more capable, evaluation increasingly relies on a single team-level number to compare architectures, coordination strategies, and deployment choices. Such a number is itself a measurement claim: it is meaningful only if the evaluation makes clear how agent-level outcomes become a team outcome and which execution state or artifact the benchmark actually scores. Recent work already shows that agent performance depends materially on harness and framework choices, not only on the underlying model (Kapoor et al., 2024; Yao et al., 2026; Emde et al., 2026). We argue that the reporting path from agents to a team score is an additional measurement layer that must be made explicit. We treat this as a measurement-validity problem: a team score is interpretable only when both the aggregation rule and the scored-object binding are recoverable from the evaluation record. Throughout the paper, we use *agent* for a conceptual participant in a multi-agent system and reserve *worker* for harness-specific execution units and source-code terminology.

Two mappings are central. First, an *agent-to-team aggregation* maps several agent outcomes to the quantity interpreted as team performance. Second, a *scored-object binding* determines which execution state or artifact is actually consumed by the grader. If either mapping is implicit, the same set of agent executions can support different team-level interpretations, and the object that is scored need not be the object that the reported team policy appears to describe. Complete logs do

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.

not by themselves solve this problem: a trace can preserve every agent execution while leaving both mappings unstated.

Prior work addresses adjacent but different questions. Harness-oriented evaluations measure how execution frameworks and configurations change performance (Yao et al., 2026; Emde et al., 2026); evidence-oriented work asks whether a claimed outcome or reported score is supported by retained records (Gao & Zhou, 2026; Zhu & Chang, 2026); protocol-validity work asks whether a benchmark still requires the intended capability (Shao et al., 2026); and multi-agent aggregation or attribution work asks which selection rule performs best or which agent contributed to an already-defined utility (Maryanskyy et al., 2026; Lu et al., 2026). The missing step is the measurement boundary between these lines of work: *given multiple agent executions, can a reader verify which rule produced the team value and whether the grader consumed the object implied by that rule?*

We operationalize this boundary with an *executable aggregation receipt*. The receipt binds agent-level outcomes, a declared aggregation rule, the concrete object consumed by the grader, and the resulting team score to the execution evidence supporting each step. Harness-specific checkers reconstruct the observed path from native records rather than trusting a success flag emitted by the system under audit. We further implement a harness repair that isolates agent execution state when isolation is the declared policy, records every agent outcome, computes the declared aggregation, and preserves the identity of the scored artifact.

Two of the implementation facts we audit are already documented by the source study's current version of record: shared-container execution and first-subagent `final_env_output` for SWE-bench Verified and Terminal-Bench (Kim et al., 2026). We claim no first discovery of those, and we instead make their measurement consequences executable. Our additional empirical contribution begins with PlanCraft. To our knowledge, the following PlanCraft grading-path mismatch has not been previously documented: the released independent-runner comment describes it as graded from `agent_output`, while the native evaluator derives success from `final_env_output`. We do not settle that mismatch by reading source. Running every three-agent outcome assignment through the native grader shows that the reported score follows one agent in 24 of 24 paired comparisons, while changing either of the other two agents moves it in 0 of 48.

The reporting step we audit is not confined to one harness. On the SWE-bench Verified leaderboard, 82 of the 175 submissions carrying metadata declare more than one attempt per instance, so interpreting each published number requires knowing how those attempts were reduced to a submitted result. The leaderboard's own checklist asks that this selection mechanism be recorded. In 56 of those 82, the README retained by the leaderboard does not name the mechanism: 30 only link a report and 26 say nothing, while 26 name it. A further 49 tick the checklist line reading "does not attempt the same task instance more than once" while their own metadata declares otherwise. We read the text the leaderboard holds rather than the linked reports, so the named count is a lower bound on what a reader could recover elsewhere (Appendix F). This census measures what the leaderboard-held submission text explicitly names; it does not establish how often an unnamed selection mechanism changes a published number.

Our main contributions are:

- **An agent-to-team measurement audit.** We formalize the two bindings required to interpret a multi-agent team score and trace them through a released harness. Beyond verifying the documented Docker behavior, we identify and experimentally isolate a distinct PlanCraft grading-path mismatch: under controlled interventions on harness workers, only changes to the score-bound worker change the reported result.
- **A diagnostic for aggregation ambiguity.** For the inspected family of binary agent-to-team rules, every rule-specific batch rate lies between the per-instance minimum and maximum agent outcomes, and the interval width equals the fraction of instances on which agents disagree. The diagnostic quantifies how much the record leaves unresolved when the agent-to-team rule is not encoded.
- **An executable repair and independent transfer.** We implement a receipt and harness repair that make agent execution state, aggregation, and scored-object lineage directly checkable without replacing the benchmark grader. Controlled interventions and model-driven runs under two models test the repaired path. We then apply the receipt to 220 episodes from MALLM, an independently developed multi-agent framework: a source-pinned native

adapter reconstructs the executed decision path while distinguishing it from the protocol semantics recorded by the framework.

The goal is not to prescribe one universally correct aggregation rule. It is to ensure that whichever team construct a study reports is explicit, reproducible, and bound to the object that was actually scored.

2 Related Work

Harness and system-level agent evaluation. Agent benchmarks increasingly treat the execution system as part of what is being measured. Kapoor et al. (2024) emphasizes cost, reproducibility, and benchmark design rather than treating accuracy as an isolated model property. Harness-Bench varies harness configurations under shared tasks and budgets and shows that context, tools, state, constraints, and recovery logic materially affect outcomes (Yao et al., 2026). MASEval similarly treats the complete multi-agent implementation—including framework, topology, orchestration, and error handling—as the unit of analysis (Emde et al., 2026). These works establish that implementation choices change measured performance. Our focus is the next internal boundary: once a harness has produced several agent executions, what rule turns them into the number reported as team performance, and what object is actually passed to the grader?

Evidence sufficiency, provenance, and protocol validity. A complementary line of work asks whether reported outcomes are supported by retained evidence. Evidence-supported bounds distinguish verified, failed, and unresolved outcomes and propagate unresolved cases into score intervals (Gao & Zhou, 2026). ClaimReceipt binds claim-relative evidence to a committed experiment manifest and tests whether a claim is recomputable from retained records (Zhu & Chang, 2026). Protocol-validity work audits whether scoring paths allow credit without demonstrating the intended capability (Shao et al., 2026). Workflow specifications, contracts, and provenance systems likewise make execution obligations or object relationships explicit (Amini et al., 2026; Kamath et al., 2025; Ye & Tan, 2026; Bhardwaj, 2026; Pimentel et al., 2017). We do not claim that receipts or provenance are new in general. We specialize these ideas to a multi-agent measurement problem that requires two linked semantic checks: agent outcomes must be mapped by a declared team rule, and the resulting interpretation must be bound to the object actually consumed by the scorer.

Aggregation, selection, and agent attribution. Multi-agent research also studies how to combine disagreement and assign credit. Selection and synthesis policies can substantially change final quality (Maryanskyy et al., 2026), while removal-based attribution asks how much each agent contributes to an already-defined system utility (Lu et al., 2026). Failure taxonomies and coordination analyses identify additional ways multi-agent systems can break (Cemri et al., 2025; Zhang et al., 2025; Cao et al., 2026). These studies typically optimize, compare, or attribute a team target after that target has been defined. Our paper audits the prior measurement step: whether the team value itself is reconstructible from an explicit agent-to-team rule and whether that value corresponds to the scored object.

3 Problem Formulation and Executable Aggregation Receipt

3.1 Agent outcomes, team aggregation, and scorer binding

For task i, let $\mathbf{y}_i = (y_{i1}, \ldots, y_{in}) \in \mathcal{Y}^n$ denote the recorded outcomes of n agents under a fixed benchmark outcome definition. Some aggregation rules also require auxiliary execution information, such as agent order or terminal/completion status; let $z_i \in \mathcal{Z}$ denote that information. A multi-agent reporting policy declares an aggregation rule

$$A : \mathcal{Y}^n \times \mathcal{Z} \to \mathcal{T}, \qquad t_i = A(\mathbf{y}_i, z_i), \tag{1}$$

where t_i is the intended team-level outcome.

Separately, let $e_i \in \mathcal{E}$ denote the retained execution evidence for task i. A declared scored-object binding B selects the concrete object consumed by the benchmark scorer:

$$o_i = B(e_i; A, z_i), \qquad r_i = S(o_i), \tag{2}$$

where o_i is the scored object, S is the benchmark scorer, and r_i is the reported value. The notation allows the object binding to depend on the declared aggregation policy and any auxiliary execution information that policy requires.

A team-level interpretation requires more than numerical agreement between r_i and one possible $A(\mathbf{y}_i, z_i)$. We call the path *auditable* when the record (i) identifies the agent outcomes relevant to the claim, (ii) declares the agent-to-team rule and any auxiliary execution information required by that rule, (iii) preserves the lineage of the object consumed by the scorer, and (iv) allows the reported value to be recomputed from those bindings. Without these conditions, several policies can be consistent with the same observed number.

This criterion exposes two distinct failure modes. **Execution coupling** occurs when agent outcomes interpreted as separate attempts depend on shared mutable state that is not represented in the machine-readable evaluation record. **Scoring-binding failure** occurs when the object consumed by the scorer is not the object implied by the declared team policy, or when the relevant binding is not declared. Shared state and positional scoring are not inherently invalid; the measurement problem arises when their role in the reported construct is implicit.

3.2 Aggregation ambiguity as a record-level diagnostic

Let $\mathcal{R}$ denote the inspected family of binary agent-to-team rules: positional selection, first-completed selection, disjunction, outcome-based selection, and unanimity. For $\mathbf{y}_i \in \{0,1\}^n$, every rule in $\mathcal{R}$ returns either an agent coordinate, the maximum agent outcome, or the minimum agent outcome. Thus, for every admissible auxiliary state z_i and every $A \in \mathcal{R}$,

$$\min_j y_{ij} \le A(\mathbf{y}_i, z_i) \le \max_j y_{ij}. \tag{3}$$

For a batch of m tasks, define the rule-induced rate envelope

$$L = \frac{1}{m}\sum_{i=1}^{m} \min_j y_{ij}, \qquad U = \frac{1}{m}\sum_{i=1}^{m} \max_j y_{ij}. \tag{4}$$

Averaging Eq. 3 over the batch shows that every inspected rule's batch rate lies in $[L, U]$. Because the outcomes are binary, $\max_j y_{ij} - \min_j y_{ij} = 1$ exactly when at least two agents disagree on task i, and is 0 otherwise. Therefore,

$$U - L = \frac{1}{m}\sum_{i=1}^{m} \mathbb{I}\{\exists j, k : y_{ij} \neq y_{ik}\}. \tag{5}$$

Hence the width of the rule-induced envelope equals the fraction of tasks on which agents disagree. This is a deterministic property of the retained agent-level record, not a sampling confidence interval. Because the inspected family contains both unanimity and disjunction, its lower and upper endpoints are attained by those two rules, respectively. Appendix A gives the corresponding proposition and proof.

3.3 Executable aggregation receipt

The receipt turns the auditable-path criterion into an executable interface. Its declaration specifies (i) the reporting target, (ii) the agent outcomes required for that target, (iii) the aggregation rule, (iv) the scored-object binding, and (v) the source and evidence files that establish those relationships. A manifest binds the declaration, checker code, and inputs. After execution, harness-specific checkers independently reconstruct agent outcomes and scorer inputs from native records and return *conforms*, *violates*, or *unresolved* with supporting evidence. The key specialization is compositional: the agent-to-team rule and the team-to-scorer object binding are verified as one measurement path. A success field produced by the system under audit is never treated as its own oracle.

The receipt is deliberately selective. If the agent-to-team rule conforms but the scored-object link is unresolved, the team interpretation is withheld rather than repaired by assumption. Conversely, a failure unrelated to a target does not invalidate narrower facts that remain directly supported. The more general estimand and dependency model used by the implementation is retained in Appendix A; the main paper uses only the agent-to-team and scored-object obligations needed for the experiments below.

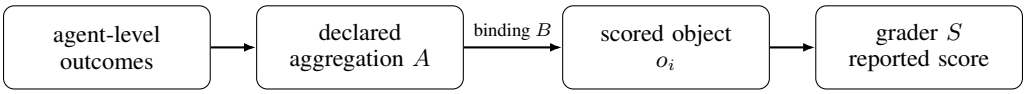

Figure 1: The measurement path audited in this paper, in which the receipt verifies the execution evidence for each binding. An agent-level record alone does not establish which aggregation produced a team outcome, and a team outcome alone does not establish which object the grader consumed.

4 Harness Repair and Verification Procedure

The receipt diagnoses whether the declared measurement path is supported; the harness repair tests whether the observed failures can be removed without replacing the underlying task grader. We implement three changes outside the vendored benchmark code. First, when agent isolation is declared, environment identity is keyed by both task and the harness worker identifier so that each agent execution receives a distinct mutable state. Second, every agent execution's terminal outcome is retained and the team value is computed from an explicit aggregation rule rather than inferred from a single environment status. Third, the code-repair path preserves an in-container identity for the full patch, allowing the artifact that changed the environment to be distinguished from a truncated textual record. The original benchmark evaluator remains unchanged.

The repair does not privilege one aggregation policy. A positional rule, disjunction, unanimity, or another supported rule can be declared over the same agent-level outcomes. The purpose is to make the policy executable and reproducible: the reported team value is accompanied by the rule that produced it and by the object lineage required to verify what the grader consumed. This separation lets us test whether the issue is merely one of documentation or whether changing the measurement path changes the reported interpretation.

Verification has two stages. The *execution check* reconstructs whether agent execution states satisfy the declared isolation or sharing policy. The *scoring check* reconstructs agent outcomes, applies the declared aggregation, follows the bound object into the benchmark evaluator, and compares the recomputed result with the reported value. These checks are deliberately separate because coordination semantics and scorer semantics can fail independently. Source hashes and manifests establish custody of the inspected bytes; the semantic check establishes whether those bytes implement the declared measurement path.

We test the implementation against changed declarations, missing observations, source substitutions, and ineffective interventions; Appendix C reports these controls. The empirical evaluation then asks four progressively broader questions: whether a binding problem exists, whether it can be causally isolated under controlled agent outcomes, whether the repair restores an auditable path, and whether the same audit transfers to an independently developed multi-agent framework.

5 Experiments

We organize the empirical evaluation around four questions that correspond directly to the proposed measurement path. **RQ1** asks what object the released harness actually scores and whether agent executions are coupled on that path. **RQ2** asks whether the harness-selected worker causally controls the reported value when the other agent outcomes are held fixed. **RQ3** asks whether the harness repair restores a directly checkable agent-to-team path under both assigned and model-driven behavior. **RQ4** asks whether the receipt transfers to an independently developed multi-agent framework. The first three questions use the released agent-scaling harness (Kim et al., 2025; Kim, 2026); RQ4 uses MALLM (Becker et al., 2025). We evaluate the method by whether the audited binding can be reproduced, isolated by intervention, made conformant after repair, and reconstructed on an independent implementation; none of these results is interpreted as a corrected historical leaderboard score.

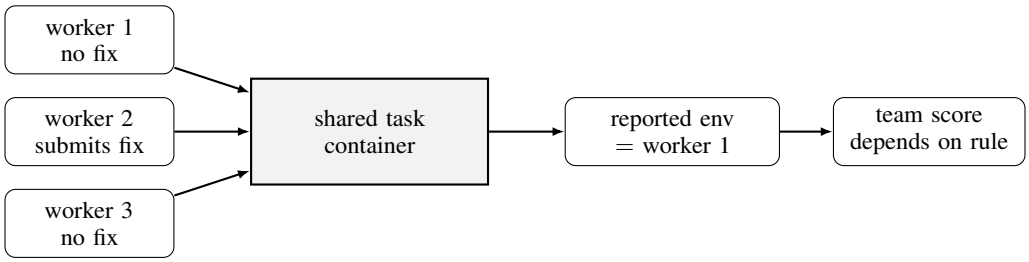

Figure 2: Observed measurement path in the released SWE-bench harness. All workers act on one mutable container, but the reported environment status is bound to worker 1. Thus, even when worker 2 successfully applies the reference fix to the shared container, the positional report remains failure when worker 1 does not submit it.

5.1 RQ1: What object does the released harness actually score?

In the released harness, each agent execution is instantiated as a worker. The independent multi-agent runner returns both a synthesized textual `agent_output` and a `final_env_output`. On the inspected environment-state path, `final_env_output` is the first worker's environment status, and the SWE-bench evaluator derives `resolved` from that status. Separately, the environment cache is keyed by task identity, so several workers on the same task can receive the same Docker container. The source paper explicitly documents this shared-state/first-subagent behavior for SWE-bench Verified and Terminal-Bench (Kim et al., 2026); RQ1 is therefore a measurement audit of its executable consequence, not a first-discovery claim.

We reproduce this path without a model on one SWE-bench Verified instance. Three workers act on the released environment; the second worker submits a reference fix. The fix evaluates successfully in the shared container, yet the positional environment status reports failure because the first worker did not submit it. Other selection rules already represented in the codebase return success on the same worker outcomes. The experiment therefore establishes object divergence under positional reporting and supplies a controlled witness for the receipt checks; it does not claim that an alternative aggregation rule is inherently preferable.

To test whether the mechanism is specific to SWE-bench, we registered four source-level predictions about Terminal-Bench, the harness's other container benchmark, before inspecting that environment. All four hold: the cache is again task-keyed rather than worker-keyed, the per-instance metric reads the selected `final_env_output`, the machine-readable execution path carries no per-run declaration of the environment-state agent-to-team scoring rule, and the agent implementation does not branch on the dataset. This is a source-level transfer check rather than an execution-rate estimate; Appendix B gives the registration and falsifiers.

5.2 RQ2: Does agent position determine the score under controlled outcomes?

PlanCraft provides the sharper scored-object test. In the released independent runner, a source comment groups PlanCraft with text-output benchmarks and states that its grader reads `agent_output`; however, the released PlanCraft dataset evaluator derives success from `final_env_output`. Because the same runner sets that object to the first worker's environment status, the grading path described by the runner comment and the one implemented by the native evaluator disagree. We test the consequence rather than relying on source inspection alone. PlanCraft allows every three-agent success/failure pattern to be attained through the native tools and grader (Dagan et al., 2025). We select six public task states and execute all 2^3 agent-outcome assignments for each, producing 48 cells. Paired comparisons change one agent's attained outcome while holding the other two fixed, which gives four pairs per agent per task and so 24 pairs per agent across the six tasks.

Changing worker 1 changes the reported score in 24/24 paired comparisons. Changing worker 2 or worker 3 changes it in 0/24 comparisons. Replacing only the synthesized text leaves all 48 grades unchanged. These results isolate the scorer dependency rather than model capability: they show that, on this released path, the reported value is a measurement of the selected worker outcome under

Changed worker	Comparisons	Score changes
Worker 1	24	24
Worker 2	24	0
Worker 3	24	0

Table 1: Controlled PlanCraft interventions. Each comparison changes one worker's attained outcome while the other two are held fixed, and every requested outcome is attained by native execution. Only changes to the first worker reach the reported score on the inspected path.

Batch	Model	m	Released = agent 1	Disagree (isolated)	Isolated envelope $[L, U]$
1	A	10	10/10	1/10	$[0/10,\ 1/10]$
2	B	10	10/10	4/10	$[4/10,\ 8/10]$
3	B	20	20/20	3/20	$[7/20,\ 10/20]$

Table 2: Model-driven batches. Here m is the number of instances in a batch and every entry is a rate over that batch, as in Eq. 4; the agent count is three throughout. The last two columns describe the isolated arm: the envelope spans the five inspected aggregation rules and, by Eq. 5, its width equals the fraction of instances on which that arm's agents disagree, so the record leaves that share unresolved when the rule is not declared. The released arm's own disagreement rates are 0/10, 2/10 and 3/20. Model identities and versions are released with the artifact and on de-anonymisation. Batches 2 and 3 share a registered stratum and may be read together; batch 1 uses a different model and selection and is never pooled with them. Appendix C gives the per-rule detail.

a team-level label. They do not establish that historical published scores are wrong, because the historical worker-level execution records needed for such a correction are unavailable.

The same observation gives the ambiguity diagnostic in Eq. 4: when agents agree, the inspected rules coincide; when they disagree, different declared team policies can produce different values from the same agent outcomes. The rule should therefore be part of the evaluation record rather than inferred from the final number.

5.3 RQ3: Does repair make the team score auditable?

We first rerun all eight fix/no-fix assignments for three workers on the SWE-bench case under the released and repaired harnesses, with no model in the loop. The released arm reports worker 1's outcome in 8/8 assignments. The repaired arm creates three distinct worker environments in every assignment, records all worker outcomes, and reproduces its declared aggregation in 8/8 assignments. Positional and disjunctive reporting diverge exactly when the declared rules should disagree—when only a later worker fixes the task. All six registered repair predictions hold. Thus the repair does more than add metadata: it converts an implicit measurement path into one whose worker state, aggregation, and scorer binding can be recomputed from the record. Appendix B provides the full accounting.

We then let models drive the workers through the same released driver under both arms. Three registered batches are reported separately and summarized in Table 2: the authors' seed-42 selection under one model, and two batches from a separately registered easier stratum under a second model. Two readings hold across every batch. The released arm reports worker 1's outcome on all 40 instances, and on every instance where workers disagree the released record alone encodes no unique agent-to-team rule while the repaired record's declared rule is directly checkable.

The envelope column is the practical cost of an undeclared rule. On batch 2, the five rules yield 4 to 8 resolved instances out of 10. This width is not sampling uncertainty about the model; it is ambiguity about which team construct the record reports. Batch 2 also shows the coupling directly, recording 24 worker successes in the shared environment against 19 under isolation, with a later worker's fix scored as zero by the positional report. Because trajectories are regenerated across arms we do not read that gap as a causal estimate.

The third batch, registered after the second and drawn from the same stratum, reproduces all four registered mechanism predictions. The two registered effect-size predictions fall outside their pre-

specified ranges: agents disagree on 3 of 20 instances rather than the registered 4 to 12, and the shared environment inflates the worker success count on 3 instances rather than the registered 5 or more. We report both misses as registered. They bound how far the batch-2 effect sizes generalize without changing the mechanism result.

These model-driven runs use a command-line model transport rather than the source study's native function-calling interface, so their rates are not compared with the paper's published benchmark numbers. They test the mechanism and the repair under one controlled transport. Appendix C reports instance-level outcomes, calls, and transport limitations.

5.4 RQ4: Does the receipt transfer to an independent multi-agent framework?

We next apply the receipt to MALLM (Becker et al., 2025), a released multi-agent debate framework we did not write, pinned at upstream revision `a57c4a77`. The evaluation contains 220 previously executed episodes: two 100-episode arms on a frozen STEM set and two task-disjoint 10-episode follow-up arms. No model is called for this audit. We first run the generic checker unchanged, then use a registered harness-specific adapter that implements the rule label from MALLM's pinned source. Appendix E gives the full registration, episode accounting, and falsifiers.

Table 3 summarizes the independent transfer. The generic receipt is unresolved on 216 of 220 episodes; two episodes admit a unique inspected rule only after the fact and two have no utterance to bind. The native adapter then reconstructs the executed answer path and separately checks the implementation named by the record and its declared semantics.

Audit check	Result	Interpretation
Generic receipt	216/220 unresolved	Configured consensus rule is not directly verifiable from the generic record.
Answer-path reconstruction	214/214	Native adapter recovers the executed answer path.
Label implementation	206/214 agree	Pinned label implementation agrees with the recorded decision.
Consensus-semantics check	97/100 violations	Recorded accounting conflicts with the source-pinned semantics on stock episodes.
Follow-up class identity	8/10 mismatch	Registry label does not name the class that ran.

Table 3: Independent MALLM transfer. The native adapter separates reconstructing the executed path from checking whether the recorded protocol label and declared semantics describe that path. Full episode accounting and registered falsifiers are in Appendix E.

No field identifies which agent utterance became the scored answer, and matching by answer string ties on 213 of 220 episodes. All four registered predictions hold. Thus the receipt transfers beyond the original harness while preserving the distinction between an executed path that can be reconstructed and a declared policy whose semantics may not match that path.

6 Discussion and Conclusion

6.1 What the Results Establish

The experiments support a measurement-level conclusion rather than a claim about one preferred coordination policy. A multi-agent evaluation can retain all agent traces while the reported team value behaves as a positional score from one harness worker, and executions without inter-agent messaging can still be coupled through mutable state. For SWE-bench and Terminal-Bench, the source paper already documents the shared-state/first-worker design; our audit makes its measurement consequence executable. PlanCraft provides the sharper intervention: a released runner comment and native evaluator describe different grading paths, and only the score-bound worker changes the reported result. The repair shows that isolation, explicit aggregation, and scored-object lineage can be introduced without replacing the benchmark grader. The independent MALLM audit then shows that the receipt is not specific to that harness: a source-pinned adapter reconstructs the executed decision path while separating it from the protocol semantics recorded by the framework. A frozen building-

compliance application in Appendix D isolates the scored-object binding with model behavior held fixed.

6.2 Implications for Multi-Agent Evaluation

The practical implication is that a team score should be treated as a structured measurement claim rather than a self-explanatory scalar. At minimum, a benchmark that reports one team number should expose the agent-level outcomes relevant to that number, the rule that combines them, the policy governing shared mutable state, and the identity or lineage of the object consumed by the grader. Without these fields, two systems can receive apparently comparable team scores while operationalizing different team constructs.

This distinction matters for more than reproducibility. Claims about coordination gains, agent scaling, architecture comparisons, and deployment readiness all rely on interpreting a reported value as evidence about a team rather than about an implicit selection rule or a particular execution object. Making the measurement path explicit separates model behavior from harness behavior and reporting policy, and makes disagreement between those layers visible before it is compressed into a leaderboard number.

6.3 Scope and Limitations

The evidence remains concentrated rather than broad. We audit one released harness in depth, transfer the source-level mechanism within it, and evaluate the receipt on 220 episodes of an independent framework using both the generic checker and a source-pinned native adapter (Appendix E). The leaderboard census measures whether the selection mechanism is named in leaderboard-held text, not whether an unnamed rule changes a published number. The controlled interventions establish scorer dependencies but do not show that historical leaderboard rankings are wrong. Model-driven evidence covers 40 SWE-bench instances under two models and a command-line transport different from the source study's native interface; it tests the mechanism rather than estimating a population rate. Two registered batch-3 effect-size predictions fall outside their pre-specified ranges and are reported as such, and three transport-limited instances remain in the denominator; Appendix C gives the accounting.

The receipt verifies a declared rule against retained evidence; it does not decide which aggregation rule is scientifically appropriate, authenticate an entire experimental history, or replace missing execution evidence. Generic contracts and provenance systems can reproduce many component decisions when supplied with the same semantic checks. Our contribution is therefore the MAS-specific composition, operationalization, and empirical evaluation of the agent-to-team and scored-object bindings, not executable verification in the abstract.

Four boundaries remain. The shared outcome space $\mathcal{Y}$ fits parallel attempts; heterogeneous roles would require $\mathcal{Y}_j$ per role. Equation 5 is binary, with timeout or crash mapped to failure, so it does not cover graded outcomes. The scoring check assumes a deterministic scorer; an LLM judge would require uncertainty-aware recomputation rather than equality. Finally, one isolated environment per agent scales resource use linearly with agent count.

6.4 Conclusion

Reliable multi-agent evaluation requires an auditable path, not only an aggregate number. The path begins with agent executions, passes through an explicit team aggregation, and ends at the exact object scored by the benchmark. The executable aggregation receipt makes this path directly checkable, while the harness repair demonstrates that the observed measurement failures can be removed under the tested conditions without replacing the underlying task grader. As multi-agent systems become more complex and their scores increasingly guide scientific and deployment decisions, making this boundary explicit is necessary for interpreting reported performance as evidence about the team.

Reproducibility Statement

The artifact pins source revisions and preserves task selections, interventions, model-driven records, checker inputs, and recomputation scripts. Appendices B–C document the released harness and model-driven accounting; Appendix E gives the registered independent-harness audit and native adapter; Appendices A and D give the ambiguity proof and cross-domain application.

Ethics Statement

The new studies use public benchmark tasks, released source code, and authored interventions. They report implementation behavior at identified revisions without estimating prevalence in deployed systems. No upstream endorsement is implied, and no historical leaderboard entry is relabeled.

ICLR 2027 AI Use Statement

Large language models assisted literature search, experiment design, code, analysis, conceptual development, and writing. The proposition and proof in Appendix A were written and checked by the human authors. Model-generated research assistance is separate from the live task calls and from automated paper reviews. No AI-generated text or analysis is treated as an independent correctness certificate. Human authors remain responsible for every claim, source, proof, implementation decision, result interpretation, and the final text. The ICLR 2027 policy requires disclosure of these uses.

References

Soufiane Amini et al. Open agent specification: Enabling cross-framework comparison of AI agents. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pp. 404–418, 2026. doi: 10.1145/3786335.3813130. URL https://doi.org/10.1145/3786335.3813130.

Jonas Becker, Lars Benedikt Kaesberg, Niklas Bauer, Jan Philip Wahle, Terry Ruas, and Bela Gipp. MALLM: Multi-agent large language models framework, 2025. URL https://arxiv.org/abs/2509.11656.

Varun Pratap Bhardwaj. AgentAssay: Token-efficient regression testing for non-deterministic AI agent workflows. *arXiv preprint arXiv:2603.02601*, 2026. URL https://arxiv.org/abs/2603.02601v1.

Hongliu Cao, Ilias Driouich, and Eoin Thomas. Beyond Task Completion: Revealing Corrupt Success in LLM Agents through Procedure-Aware Evaluation. *arXiv preprint arXiv:2603.03116*, 2026. URL https://arxiv.org/abs/2603.03116.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why Do Multi-Agent LLM Systems Fail? In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2025. URL https://arxiv.org/abs/2503.13657.

Gautier Dagan, Frank Keller, and Alex Lascarides. Plancraft: An evaluation dataset for planning with LLM agents. In *Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=nSV8Depcpx. arXiv:2412.21033v2.

Cornelius Emde, Alexander Rubinstein, Anmol Goel, Ahmed Heakl, Sangdoo Yun, Seong Joon Oh, and Martin Gubri. MASEval: Extending multi-agent evaluation from models to systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 345–356. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.acl-demo.34. URL https://aclanthology.org/2026.acl-demo.34/.

Shanshan Gao and Liyi Zhou. Can agent benchmarks support their scores? evidence-supported bounds for interactive-agent evaluation. *arXiv preprint arXiv:2605.10448*, 2026. URL https://arxiv.org/abs/2605.10448.

Adharsh Kamath, Sishen Zhang, Calvin Xu, Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. Enforcing temporal constraints for LLM agents, 2025. URL https://arxiv.org/abs/2512.23738.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI Agents That Matter. *arXiv preprint arXiv:2407.01502*, 2024. URL https://arxiv.org/abs/2407.01502.

Yubin Kim. Agent Scaling: Released multi-agent evaluation framework. Source commit 6f3bfb78a6481c1098d182680f39b0f904b292a2; implementation matches archived v2.1.3, 2026. URL https://github.com/ybkim95/agent-scaling/tree/6f3bfb78a6481c1098d182680f39b0f904b292a2.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a Science of Scaling Agent Systems. *arXiv preprint arXiv:2512.08296*, 2025. URL https://arxiv.org/abs/2512.08296.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, et al. Capable language models can outgrow the benefits of collaboration. *Nature Machine Intelligence*, 8:1157–1172, 2026. doi: 10.1038/s42256-026-01268-y. URL https://doi.org/10.1038/s42256-026-01268-y.

Mingyu Lu, Yushan Huang, Chris Lin, and Su-In Lee. Agents that matter: Optimizing multi-agent LLMs via removal-based attribution. *arXiv preprint arXiv:2605.27621*, 2026. URL https://arxiv.org/abs/2605.27621.

Artem Maryanskyy, Dmitry Budnikov, and Alibek T. Kaliyev. When agents disagree: The selection bottleneck in multi-agent llm pipelines. *Applied Sciences*, 16(10):4914, 2026. doi: 10.3390/app16104914. URL https://doi.org/10.3390/app16104914.

Joao Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. noWorkflow: a tool for collecting, analyzing, and managing provenance from python scripts. *Proceedings of the VLDB Endowment*, 10(12):1841–1844, 2017. URL https://www.vldb.org/pvldb/vol10/p1841-pimentel.pdf.

Jiaqi Shao, Hanck Chen, Wei Zhang, Maxm Pan, and Bing Luo. Do agent benchmarks measure capability? protocol validity in the age of agentic ai. *arXiv preprint arXiv:2607.22368*, 2026. URL https://arxiv.org/abs/2607.22368.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. HarnessBench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026. URL https://arxiv.org/abs/2605.27922.

Qing Ye and Jing Tan. Agent contracts: A formal framework for resource-bounded autonomous AI systems, 2026. URL https://arxiv.org/abs/2601.08815v3. Version 3.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which Agent Causes Task Failures and When? On Automated Failure Attribution of LLM Multi-Agent Systems. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2505.00212.

Peiying Zhu and Sidi Chang. Claimreceipt: Verifying evidence sufficiency and coverage in agent evaluations. *arXiv preprint arXiv:2609.01992*, 2026. URL https://arxiv.org/abs/2609.01992v1.

A Formal Receipt Model and Aggregation Ambiguity

Receipt targets. The implementation separates the outcome recorded under the realized execution from stronger interpretations that additionally require complete requested execution or policy provenance. Let q denote a reporting target and $R(q)$ the semantic checks required by that target. For evidence bundle x, check j returns $V_j(x) \in \{\texttt{conforms}, \texttt{violates}, \texttt{unresolved}, \texttt{not_applicable}\}$. The receipt permits a target only when every required check conforms:

$$\text{permit}(q, x) \iff \forall j \in R(q),\ V_j(x) = \texttt{conforms}. \tag{6}$$

A failed or unexercised check therefore withholds only the targets that require it. This rule does not establish randomization, exchangeability, population identification, or the scientific appropriateness of the declared aggregation policy.

Aggregation ambiguity. For one instance, let $\mathbf{y} \in \{0,1\}^n$ be the agent outcomes and let $z \in \mathcal{Z}$ contain any auxiliary execution information needed by the declared rule. For every $A \in \mathcal{R}$ in the inspected rule family,

$$\min_j y_j \le A(\mathbf{y}, z) \le \max_j y_j. \tag{7}$$

For a batch of m instances,

$$L = \frac{1}{m}\sum_i \min_j y_{ij}, \qquad U = \frac{1}{m}\sum_i \max_j y_{ij}, \tag{8}$$

and every inspected rule's rate lies in $[L, U]$. Because the outcomes are binary,

$$U - L = \frac{1}{m}\sum_i \mathbb{I}\{\exists j, k : y_{ij} \neq y_{ik}\}. \tag{9}$$

Thus the envelope width equals the agent-disagreement rate.

Proof. Each inspected rule returns either an agent coordinate, $\max_j y_j$, or $\min_j y_j$, so Eq. 7 holds pointwise. Averaging the pointwise inequality over the m instances yields Eq. 8. For binary outcomes, $\max_j y_{ij} - \min_j y_{ij}$ equals one exactly when at least two agents disagree and zero otherwise; averaging this identity gives Eq. 9. Because unanimity returns the minimum and disjunction returns the maximum, both endpoints are attained within the inspected rule family. □

This is a deterministic record-level statement, not a sampling confidence interval or a population-identification result. If all agents agree on an instance, all inspected rules coincide on that instance, so the agent-level outcome record alone cannot identify which rule was used.

B Released Harness, Controlled Interventions, and Repair

Released SWE-bench path. For the inspected source revision, the independent runner returns a synthesized textual answer together with one `final_env_output`; the latter is the first worker's environment status. The SWE-bench evaluator derives `resolved` from that status. The Docker environment cache is keyed by task identity rather than by both task and worker, so workers on one task can share the same mutable filesystem. In the no-model witness used in the main paper, worker 2 applies a reference fix successfully while worker 1 does not submit it; the shared container contains the fix, but positional reporting returns worker 1's status.

Controlled PlanCraft cube. On the inspected source revision, the independent-runner comment describes PlanCraft as graded from `agent_output`, while the native PlanCraft dataset evaluator derives success from `final_env_output`. Six public task states are paired with all eight three-worker success/failure assignments, yielding 48 native cells. Desired outcomes are attained through the released task tools rather than by directly setting success flags. Holding the other workers fixed, changing worker 1 changes the grade in 24/24 comparisons; changing worker 2 or worker 3 changes it in 0/24. Replacing only the synthesized text also leaves all 48 grades unchanged. These cells test scorer reachability and do not estimate a population failure rate.

Repair. The harness repair (i) keys mutable task state by task and worker, (ii) records every worker outcome and an explicit aggregation rule, and (iii) stores an in-container patch identity. On the eight fix/no-fix assignments for the SWE-bench witness, the released arm reports worker 1's outcome in all eight. The repaired arm creates three distinct worker environments in all eight, reproduces its declared rule in all eight, and separates positional from disjunctive reporting exactly when only a later worker fixes the task. Six pre-specified repair predictions hold.

Transfer to the other container benchmark. Before reading the second container environment in the harness, we recorded four source-level predictions: task-keyed shared state, scoring from the selected `final_env_output`, no per-run machine-readable declaration of the environment-state agent-to-team scoring rule, and no dataset-specific branch in the agent selector. All four hold on inspection. No container was started for this transfer, so it supports mechanism portability within the harness but not an execution-rate claim.

C Model-Driven Runs and Transport Accounting

Each instance is run under both the released arm and the isolated repair with three workers. The released arm uses the original shared environment and positional reporting; the repair uses worker-isolated environments and records every outcome under a declared rule. The model transport is command-line based rather than the source study's native function-calling interface, so the resulting rates are compared only between arms under the same transport.

Batch / arm	positional	disjunction	rule-rate envelope $[L, U]$	disagreement rate
Batch 1, released	1/10	1/10	[1/10, 1/10]	0/10
Batch 1, isolated	0/10	1/10	[0/10, 1/10]	1/10
Batch 2, released	8/10	9/10	[7/10, 9/10]	2/10
Batch 2, isolated	7/10	8/10	[4/10, 8/10]	4/10
Batch 3, released	9/20	11/20	[8/20, 11/20]	3/20
Batch 3, isolated	9/20	10/20	[7/20, 10/20]	3/20

Table 4: Model-driven-run reporting under two representative aggregation rules. Entries are rates over each batch's instance count. The full inspected rule family lies in the same $[L, U]$ envelope, whose width equals the disagreement rate. Batches 2 and 3 are drawn from the same registered stratum and may be read together; batch 1 uses a different model and selection and is never pooled with them.

Across batches 1 and 2 (20 released-arm instances), the reported value equals worker 1's outcome. In the second batch, the shared environment records 24 worker successes versus 19 under isolation, with the difference occurring on five instances. Because model trajectories are regenerated across arms, this difference is consistent with the state-coupling mechanism established by the assigned intervention but is not itself a causal estimate or an accuracy advantage. One later-worker fix is reported as zero by the positional path. Across the five disagreement cases observed in batches 1 and 2, the machine-readable released record does not encode a unique environment-state agent-to-team rule, whereas the repaired record exposes and checks its declaration. No claim is made that these instances estimate the source paper's benchmark rate.

Third batch: mechanism replication and effect-size predictions. The third batch was registered before execution as positions 11 to 30 of the same stratum used for batch 2, under the same model and transport, with six predictions. All four registered mechanism predictions (P1–P4) hold, while the two registered effect-size predictions (P5–P6) fall outside their pre-specified ranges. The released arm's reported value equals worker 1's outcome on 20 of 20 instances (P1). Positional and disjunctive reporting differ on one instance (P2). On each of the three instances where workers disagree, the undeclared record identifies no rule and the declared check conforms (P3). The end-of-run auto-submit credits a worker that never submitted on 8 instances, 3 of which are reported as resolved (P4). P5 registered an isolated-arm envelope width of 4 to 12 of 20, extrapolating batch 2's rate; the observed width is 3. P6 registered per-instance inflation on at least 5 of 20; the observed count is 3, although the arm totals still differ, 28 worker successes released against 26 isolated. We report both misses as registered. They bound how far the batch-2 effect sizes carry and do not bear on P1–P4, which concern the mechanism rather than its magnitude. Three instances (`psf__requests-1921`,

pylint-dev__pylint-4970, pytest-dev__pytest-5809) are transport-limited: the command-line transport rejected the full harness prompt deterministically, exiting nonzero without output on every attempt across four reruns. In each the repaired arm recorded only three calls. In two of the three the released arm failed the same way; in the third the released arm completed 48 calls without a worker success. All three are retained in the denominators as zero-success instances and flagged as transport-limited in the released records, so they narrow no interval and create no disagreement. Across the batch the transport issued 2,001 calls and produced no unparsed reply.

D Building-Compliance Cross-Domain Application

The building-compliance study is retained because it exercises the second boundary—scored-object binding—outside code repair. The same stored model outputs are judged twice by the same gate. Only the object selected for scoring changes, from the originally recorded object to the design-declared object. No model call is regenerated and no alternative judge is introduced.

The workflow's diagnostic threshold is 0.95, so redirecting the scorer binding changes the pass/fail reporting verdict in frozen execution B and in the first live execution. It does not move the verdict in frozen execution A, whose rate stays below the threshold on both bindings. Nor does it move the second live execution, which is above the threshold on both and whose rate the redirection leaves unchanged. These are worked applications rather than independent reliability estimates. The frozen executions reuse 30 cases and contain 450 completed runs each, the live executions contain 60 runs each, and the mismatches are concentrated rather than independently sampled. The result therefore supports only the claim made in the main paper. Choosing a different recorded object for the same fixed outputs can change a domain evaluation, so the scored-object binding belongs in the auditable record.

E The Receipt on an Independent Executed Harness

The aggregation checker used throughout the paper was built against, and until now run only on, records from the harness of Section 5. To test whether it reads anything else, we applied it unchanged to records of MALLM (Becker et al., 2025), a released multi-agent debate framework we did not write, pinned at upstream revision a57c4a77. The records are 220 episodes executed earlier under a command-line model transport: two arms of 100 on a frozen 100-task STEM set and two arms of 10 on a task-disjoint follow-up. The selection rule, the three questions, and four predictions were registered and frozen in the artifact before any verdict was computed. Both possible outcomes were declared reportable in advance. No model was called.

Each episode becomes one record. A worker row per agent carries the framework's own pinned scorer applied to that agent's final solution string; the reported value is the same scorer applied to the returned answer. The declaration is the label the record itself carries in its configuration. The recomputed scorer verdict matches the stored verdict on every one of the 214 episodes that stored one. The remaining six returned no answer and are reported as unscored rather than as the study's policy value of zero.

All four registered predictions hold. **P1**: a unique inspected rule is attributable after the fact on 0 of 200 episodes in the main set and 2 of 20 in the follow-up. The reason is the shape of the record. An agent that agrees copies the current solution, so on 213 of 220 episodes every agent's final solution scores the same. The five inspected rules then return one value, the envelope of Section 3 has zero width, and the number is consistent with every rule and attributable to none. **P2**: the label in every record, majority_consensus or a registry alias, is not one of the inspected rules, so the generic checker has no rule to check the number against on 220 of 220. We therefore wrote the harness-specific adapter that Section 3 anticipates, registered before it ran, which implements the label's pinned source. It reconstructs the returned answer on 214 of 214 answered episodes and the recorded success flag on 106 of 106 majority-labelled episodes with a final window. On the repair arm, whose label is our registered class, it conforms on 98 of 98 such episodes. The label's own documentation says consensus is reached when more than half of the agents agree. As a direct diagnostic, the framework's agreement flags satisfy that majority condition on 96 of 100 stock episodes, yet the framework records consensus on 0 of 100, as its authors' released exports do on 0 of 200 rows. The executed accounting counts a reverse-scan index, not agreeing agents. The

Record source	Rule declared, checkable	Scored-object lineage checkable	Verdict
Harness A, released	no	no	positional, 24/24
Harness A, repaired	yes	yes	conforms, 48/48
MALLM, generic checker	label only; not an inspected rule (220/220)	no	unresolved 216/220; 2 unique-fit; 2 no utterance
MALLM, native adapter	pinned label implementation reconstructed	no	answer path 214/214; label implementation 206/214; semantics violation 97/100

Table 5: The generic checker on three record sources, and MALLM again under its native adapter. The repaired arm's 48 comprises the eight assigned SWE-bench witness runs and the forty model-driven instances of Table 2. On two MALLM episodes the agents' outcomes differ enough that exactly one inspected rule fits after the fact, which is attribution by a reader rather than a declaration in the record. Two further episodes had no utterance to bind. The native adapter, registered after the generic run, implements the label's pinned source; its verdicts are read against the recorded answer and success flag.

source-pinned adapter performs the fuller semantic check against the recorded decision state and classifies 97 of 100 stock episodes as violations; this 97/100 is therefore not the same statistic as the 96/100 agreement-flag count. It likewise classifies all 8 follow-up stock episodes with a final window as violations; the registration expected at least 9 of 10, and the two others returned no utterance. The recorded answer is never the wrong solution; the success accounting is. In the other follow-up arm the label names a registry key while a sibling identity file records that a different class ran; the label's implementation disagrees with the recorded decision on 8 of 10 episodes. **P3**: no field states which agent's utterance became the scored answer, on 220 of 220; matching the answer string back to agents ties on 213. **P4**: the repaired harness of Section 4 conforms on all 48 of its records under the same checker.

On an independently developed framework, then, the executed selection is recoverable and the receipt reproduces it, while the declared protocol and the executed accounting disagree on nearly every episode, and the only rule-naming field of one live arm does not name the class that ran. The generic checker reaches *unresolved* without an adapter, which is the intended verdict when no inspected rule is declared and is also the cost of running it on a system it was not written for. The checker's specificity still rests on records built to be checkable: the repaired harness, and the repair arm here. It shows no published MALLM number to be wrong and estimates no prevalence.

F Census of a Public Leaderboard's Submissions

The leaderboard's submission checklist distinguishes a single attempt from Best@k, in which "a distinct module of your system decides which attempt to submit". It asks Best@k submitters to set `system/attempts` to `2+` in their metadata, and asks that trajectories "reflect all rollouts + the mechanism for selecting which solution was used". The checklist therefore already treats the selection rule as part of what a submission owes a reader, which is the same step this paper audits inside a harness.

We fetched `metadata.yaml` for every submission under `evaluation/verified` of the leaderboard repository on 2026-09-10: 182 folders, 175 with metadata. For every submission declaring attempts other than 1 we read the `README.md` the leaderboard itself holds and coded whether it names the selection mechanism, only links a report, or says nothing.

Three limits apply. The attempts field is self-reported. We did not read the linked reports, so "only links a report" bounds what a reader could recover elsewhere rather than showing the report silent, and the named count of 26 is a lower bound. The checklist line "Is a pass@1 submission (does not attempt the same task instance more than once)" sits in the same template that treats Best@k as pass@1, so an entry ticking it while declaring `2+` inherits a contradiction of the template; the record does not resolve it either way, and we count it rather than interpret it. The census therefore measures how often the selection mechanism is not named in the leaderboard-held submission text. It

Submissions with metadata	175
Declare more than one attempt per instance	82
names the selection mechanism	26
only links a report	30
says nothing	26
Tick the “does not attempt the same task instance more than once” line while declaring `2+`	49

Table 6: Coding of the leaderboard’s own submission text. These are population counts over the fetched set, not a sample.

does not measure how often an unnamed rule changes a published number, which would require the worker-level records these submissions do not release.

G Reproducibility Notes

The released artifact records source revisions, frozen task selections, authored assignments, model-driven-run outputs, scorer inputs, and scripts that recompute the quantities reported in the main paper. The controlled SWE-bench repair and PlanCraft cube do not require model calls. Model-driven runs require the stated model transport and preserve per-instance records. Source hashes establish which bytes were inspected but do not establish semantic correctness by themselves; the semantic checks described in the paper remain necessary.