DIAGNOSTIC RESOLUTION: WHAT AGENT DIAGNOSES PRESERVE FOR DOWNSTREAM USE

Anonymous authors
Paper under double-blind review

ABSTRACT

Failure diagnoses are often evaluated by whether they identify the right failure, but later use depends on what information the diagnostic record preserves. We show that task outcomes, failure labels, recurring patterns, and unchanged citations can each collapse distinctions that matter for subsequent decisions: a correct label may lack the evidence for the next check, similar failures may require different procedures, and previously valid judgments may require reassessment as evidence changes. We introduce *Diagnostic Resolution* (DiagRes), a consumer-conditional diagnotic protocol that evaluates whether a frozen diagnostic record supports correct later judgments under declared source access and reading budgets. DiagRes tests judgment support, check applicability, and reassessment, requiring answers to be correct, sufficiently supported, and bound to the appropriate requirement, execution, and version. Across 22 public tasks, DiagRes-S selection improves follow-up correctness over failure-label selection. Controlled studies across DiagRes-S/A/T distinguish material information loss from harmless record changes; with equal fact budgets and equal individual-decision accuracy, complete versus split evidence yields 50% versus 0% joint success. DiagRes-guided revision further raises correct coverage from 59.6% to 78.9% across 114 questions from 19 sources. These results suggest that reusable diagnosis depends not simply on retaining more information, but on preserving the complete grounds needed for later decisions.

1 INTRODUCTION

Trajectory-level diagnostic of agents supports improvement in multiple domains (Ge et al., 2025; Lu et al., 2026) by examining trajectory progress (Ma et al., 2024) and procedural compliance (Gritta et al., 2026). Beyond explaining individual executions, diagnoses can inform later checks and repairs across tasks and versions. Such reuse requires records to preserve not only conclusions, but also the evidence and conditions under which those conclusions hold.

Recent work develops complementary approaches to failure diagnosis. Localization methods identify responsible agents and execution steps (Zhang et al., 2025); structured methods ground conclusions in constraints and logs (Barke et al., 2026), reconstruct causal dependencies (Wang et al., 2026), and test hypotheses through interventions (Ma et al., 2025). Evaluation further assesses attribution reasoning (In et al., 2026) and repair outcomes, while verified diagnostic memory supports reuse across trajectories (Li et al., 2026). These advances improve how diagnoses are produced, but leave open what information a diagnostic record must retain for later use.

This creates a protocol gap between diagnosis's production and consumption. Records with the same failure label can differ in whether a test actually ran, which inputs a check covers, or which version supports a claim. Question-based summary metrics assess retained information (Deutsch et al., 2021; Scialom et al., 2021), but diagnostic use also depends on whether that information is sufficiently supported and bound to the conditions warranting a decision. We therefore ask: *what must a diagnostic record preserve for another person or agent to use it correctly?* Failure-label agreement serves as a coarse-abstraction control for testing whether label-equivalent records are interchangeable in subsequent use.

Across 193 public trajectories spanning 76 Harbor-Index 1.0 tasks (Shi et al., 2026), execution checks, and software-update studies, three findings expose distinct forms of diagnostic information loss: (i) *Outcome Blind Spots* hide unsupported reports or verified local repairs behind task scores;

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.

(ii) *Template Aliasing* shows that shared failure patterns can require different checks, while similar checks can apply without a shared retained pattern. (iii) *Diagnostic Staleness* shows that conclusions may require reassessment even when existing citations remain unchanged. Together, they motivate tests of support, applicability, and reassessment.

We introduce *Diagnostic Resolution* (DiagRes), a diagnostic protocol measures these requirements through the decisions a record supports. A downstream consumer as *reader*, receives a record frozen before follow-up questions or updates are revealed. Under fixed source access and reading budgets, paired answers must be correct, sufficiently supported, and grounded in evidence bound to the right requirement, execution, and version. This reader-conditional interface accommodates both free-form and structured records from different diagnostic methods.

We validate DiagRes by showing that its measured distinctions translate into better subsequent decisions. Across 22 public tasks, DiagRes-S selection improves follow-up correctness over failure labels by 6.0 and 5.7 percentage points for two readers. Controlled studies across DiagRes-S/A/T reliably detect material information loss while preserving harmless record changes, and temporal studies recover updates missed by citation-based tracking. DiagRes-guided revision further raises correct coverage from 59.6% to 78.9% across 114 questions from 19 sources.

Contributions. We make four contributions. **(i)** We formulate diagnostic evaluation around the information a record must preserve for correct subsequent judgments. **(ii)** We identify three empirical forms of information loss motivating tests of support, applicability, and reassessment. **(iii)** We introduce DiagRes, a paired evaluation protocol with controlled access, reading budgets, and evidence-bound scoring. **(iv)** We validate DiagRes through record selection, controlled interventions, fact restoration, and revision, and provide an extensible toolkit for new records, readers, and evidence criteria.

2 Background and Related Work

Diagnostic methods and their evaluation targets. Let $x = (q, \tau, \eta)$ comprise a task, recorded execution, and diagnostic context with versioned sources. Method g produces record $z = g(x)$. Attribution benchmarks compare its agents, error steps, or failure categories $\alpha(z)$ with reference annotations y_x^*:

$$S_{\text{attr}}(z; x) = m\big(\alpha(z), y_x^*\big), \tag{1}$$

where m is the benchmark's matching or ranking metric. Who&When measures agent- and step-level accuracy (Zhang et al., 2025); VerifyMAS measures F1 over agents, error modes, and their pairs (Qiao et al., 2026). MP-Bench compares rankings of failure steps against multiple expert perspectives and separately judges attribution reasoning (In et al., 2026). AgentRx examines support through applicability guards and validation logs (Barke et al., 2026). Intervention evaluation scores $\tau' = \text{Exec}(x, I(z))$ after proposed change $I(z)$; DoVer measures success, progress, and intervention compliance (Ma et al., 2025). Later use requires additional tasks and criteria.

The protocol gap: what a diagnostic record must preserve. For fixed x, records satisfying $\alpha(z_1) = \alpha(z_2)$ receive the same score in Equation 1. They can still preserve different evidence, applicability conditions, or revision dependencies. Let $\xi = (Q, S, a)$ specify a follow-up question, its governing requirement, and target scope. Write $J(x, \xi)$ for its source-referenced answer, including an unresolved state. For fixed ξ, a retained view $\phi(x)$ can determine the answer only if admissible contexts satisfy

$$\phi(x) = \phi(x') \implies J(x, \xi) = J(x', \xi). \tag{2}$$

Identical views cannot resolve different required answers. Task scores, pattern assignments, and citation lists can discard these distinctions; retrieval can recover them, making access and budget part of the protocol. Verified diagnostic memories preserve checks and guards for reuse (Li et al., 2026). Truth maintenance records revision dependencies (Doyle, 1979); recent agent-memory work examines stale dependencies and missed checks (Sun & He, 2026; Nakayashiki, 2026). Their usability for later judgments is the evaluation target.

Evaluating a record through its use. Question-based evaluation offers a foundation: QAEval and QuestEval test information retained in summaries through questions about references or source documents (Deutsch et al., 2021; Scialom et al., 2021). ALCE evaluates answer correctness and

citation quality (Gao et al., 2023). CheckList and contrast sets test behavioral capabilities and sensitivity to meaningful input changes (Ribeiro et al., 2020; Gardner et al., 2020). DiagRes combines these ideas to evaluate support, applicability, and reassessment (Section 3) through frozen records, controlled access and budget, and evidence-bound answers (Section 4).

3 Empirical Evidence on Diagnostic Information Loss

Practice-grounded evaluation emphasizes reading traces and revising checks (Husain & Shankar, 2026). We test whether outcomes, shared patterns, and retained citations preserve the information needed to verify reports, reuse checks, and revisit judgments (Equation 2 and Figure 1).

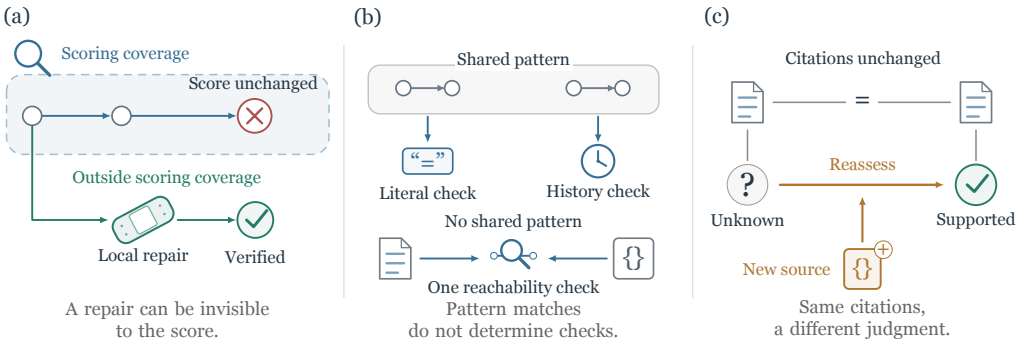


Figure 1: **Three distinctions a reusable diagnosis must preserve.** (a) *Outcome blind spots*: a local SEIRS repair changes behavior outside scoring coverage, leaving the reward unchanged. (b) *Template aliasing*: output and historical-claim cases share a pattern but need different checks; test and solver cases admit one reachability check without a shared retained pattern. (c) *Diagnostic staleness*: newly available Flask code resolves an unknown judgment while existing citations remain unchanged. These schematics summarize the case comparisons in Sections 3.1–3.3. Appendix A elaborate the source evidence.

The cases instantiate three information-loss mechanisms: outcome coverage can differ from claim support, categorical similarity from check applicability, and retained citations from current evidential sufficiency. From 193 Harbor-Index 1.0 trajectories across 76 tasks (Shi et al., 2026), we retain 103 failure instances. We use GPT-6 Astra builds evidence-linked graphs, programs check quotations and structure, and source-based model review assesses propositions and relations. We compare claims with executions and checks across cases at fixed granularity. Temporal studies track fixed propositions in six constructed workflows and 30 screened SWE-bench Verified tasks across 11 repositories (OpenAI, 2024), registering dependencies before source changes.

3.1 Outcome Blind Spots

We compare task-verifier verdicts with execution evidence for individual claims: did the agent do what its report says? In a Django regression-test task, the verifier accepts a submission whose report labels two control assertions as passing. An earlier failure prevents both from executing. The task outcome therefore leaves a reporting requirement unsatisfied (Figure 6).

The SEIRS case examines the other direction: whether a local improvement is visible in the task score. We hold the scorer fixed and change one solver branch. Let f_0 be the original solver and f_1 a repair that returns the exact solution when initial infection is absent. Write $V(f)$ for the original reward and $\mathrm{Acc}_{D_0}(f)$ for correctness on the eight zero-infection settings D_0. Paired execution against the analytic solution gives

$$V(f_0) = V(f_1) = 0, \qquad \mathrm{Acc}_{D_0}(f_0) = 50\% < \mathrm{Acc}_{D_0}(f_1) = 100\%. \tag{3}$$

The unchanged reward follows from the scorer's coverage. The repaired domain D_{repair} has zero initial exposed and infected fractions, $E(0) = I(0) = 0$, while the scoring inputs D_{score} satisfy $D_{\text{score}} \cap D_{\text{repair}} = \varnothing$ (Figure 1a). All 56 positive-infection controls are identical across solvers, and both conserve mass at all 64 settings. The first scoring failure remains on the unmodified integration path (Figure 7). Therefore, we find:

Task outcomes can conceal unsupported reports and verified local improvements. Evaluation needs claim-specific support tied to execution conditions.

3.2 Template Aliasing

Reusing a check requires determining whether its procedure applies to the new case, beyond recognizing a shared failure pattern. Our library contains two templates covering 18 discovery instances from 12 tasks. Of ten pair comparisons, six align in target and checking granularity, while four do not. Appendices A.3–A.6 give representative cases in both directions: shared patterns can require different checks, while the same check can apply across cases without a shared retained template.

The qutebrowser and Expensify reports share an error-to-outcome template. The former uses `successful` where the task requires `exited successfully`; the latter treats present code as confirmation of a past maintainer preference. Checking an output string cannot settle the historical question, which needs a record of the earlier decision (Figures 8 and 9). Conversely, a failed Django assertion prevents a later assertion, while an SEIRS input bypasses a claimed shortcut. Without a shared retained template, both admit an execution-reachability check bound to version, input, and prerequisite (Figures 10 and 11).

To express this contrast, let $M_{ij} = 1$ when a retained template covers both cases i, j, and $A_\kappa(i, j) = 1$ when a specified check κ transfers from i to j under permitted input or version rebinding. Transfer preserves the evidence type and decision predicate. The comparisons give (Figure 1b)

$$\text{Output/history: } (M, A_\kappa) = (1, 0), \qquad \text{Test/solver: } (M, A_\kappa) = (0, 1). \tag{4}$$

Through this case, we find:

In the retained library, pattern membership is neither sufficient nor necessary for check transfer. Reuse requires an applicability test for the evidence, decision rule, and scope.

3.3 Diagnostic Staleness

Across versions, reuse requires testing whether conclusions still hold and new evidence warrants review.

Controlled revision. Six constructed workflows yield 54 obligation states over three updates, including 17 transitions. References combine executions, recomputation, and model review (supplementary workflow records). Frozen judgments remain correct on 48.1% of states and 11.8% of changes. Task-level reassessment, dependency revision, and information-matched text each recover 100% of states and changes, including four required unknowns.

For fixed propositions $\mathcal{C}$, let $J_t(c) = J(x_t, \xi_c)$ denote version-t state and $\mathcal{R}_\Delta$ the review set after update Δ. For nonempty $\mathcal{C}_\Delta$, changed-state coverage is

$$\mathcal{C}_\Delta = \{c \in \mathcal{C} : J_t(c) \neq J_{t+1}(c)\}, \qquad \text{Coverage}(\mathcal{R}_\Delta) = \frac{|\mathcal{R}_\Delta \cap \mathcal{C}_\Delta|}{|\mathcal{C}_\Delta|}. \tag{5}$$

Review scope is $|\mathcal{R}_\Delta|/|\mathcal{C}|$, post-review accuracy is assessed separately.

Real source updates. Across 30 SWE-bench Verified tasks, 136 propositions are fixed before revealing one developer patch per task. An independent-context model audit identifies 32 reference changes in 17 tasks. Direct-citation caching selects 51 propositions but covers 28.1% of changes. Using citations from the same initial judgments as the dependency conditions selects 52 and covers 31.3%. Dependency-based revision and information-matched text each cover 100%, as does full re-review. In Flask, a previously unavailable constructor becomes visible and establishes rejection of empty names. The original supporting citations remain unchanged (Figures 1c and 12). We find:

Judgments can become stale while their citations remain unchanged. Evaluating their continued validity requires reassessment against changes to their supporting evidence and conditions.

The following protocol in §4 turns these information requirements into shared tests of support, applicability, and reassessment under declared access and budget.

4 DIAGNOSTIC RESOLUTION

We introduce Diagnostic Resolution (DiagRes), a diagnostic protocol measures usability conditional on reader r, source-access policy π, and reading budget B. These define the evaluation condition for support, applicability, and reassessment (see Figure 2 and Section 3).

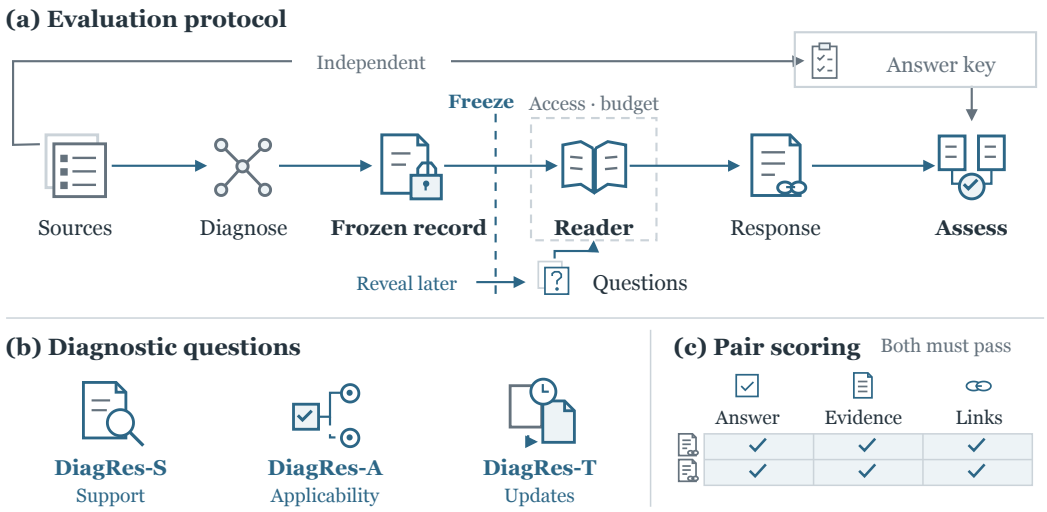


Figure 2: **Diagnostic Resolution evaluates what a diagnosis lets its reader resolve.** (a) Records are frozen before follow-up questions or updates; source access and reading budget are fixed. Independent references enter only the assessment. (b) DiagRes-S tests judgment support, DiagRes-A applicability, and DiagRes-T reassessment. (c) Both cases must satisfy all three conditions: correct answer, sufficient evidence, and correct binding. Scores are averaged within task groups, then across groups, separately for each family.

4.1 DIAGNOSTIC SETUP

Freeze the diagnostic record $z = g(x)$ before revealing follow-up questions or updates. Records can be text, logs, or structured objects. A reader r, either a person or an agent, answers using the record and permitted sources under a fixed reading budget. Define a case and the reader's response as

$$u = (\xi, F, v), \qquad (\widehat{y}, \widehat{E}, \widehat{b}) = r(z, \xi, v; F_\pi, B), \tag{6}$$

where $\xi = (Q, S, a)$ fixes the question, requirement, and target scope from Section 2; F is the source set at version v. The response contains an answer, evidence, and its binding to the target. The access rule π selects accessible sources F_π; B is the reading budget. Record-only access sets $F_\pi = \varnothing$. Record-plus-source access provides the same retrieval pool across methods; full-source reading supplies an information-access reference.

The reference $J(x, \xi)$ and acceptable evidence and bindings are fixed from complete declared sources before assessment, common across record conditions. *Unresolved* is correct when those sources leave the question open. If a source log establishes that an assertion was never reached, "unknown" from a truncated record fails: missing record content does not change the source-referenced answer.

4.2 THREE DIAGNOSTIC TASKS

Judgment support (DiagRes-S). The answer space is $\mathcal{Y}_S = \{\text{supported}, \text{refuted}, \text{unresolved}\}$. The reader assesses a specified proposition under requirement S using evidence for that property: a reported test result, for example, requires the relevant execution. Each question identifies whether S comes from the original task or an additional research profile.

Check applicability (DiagRes-A). Represent a check as $\kappa = (S, \mathcal{E}, \psi, \gamma, \Theta)$: its requirement, evidence type, decision predicate, applicability guard, and permitted rebindings. A binding $\theta \in \Theta$

substitutes declared inputs or versions while preserving $\mathcal{E}$ and ψ. The reader determines whether γ holds directly, after permitted rebinding, never within Θ, or remains unresolved from the sources. These yield four answers: direct, rebind, inapplicable, and unresolved. They operationalize $A_\kappa(i, j)$ from Section 3.2 at a fixed checking granularity. Positive and negative cases include shared-pattern and cross-pattern pairs.

Reassessment after change (DiagRes-T). For a fixed proposition c, the reference comprises both a review decision $\rho^*_\Delta(c) \in \{0, 1\}$ and the current evidential state:

$$y^*_T(c, \Delta) = \big(\rho^*_\Delta(c), J_{t+1}(c)\big), \qquad J_t(c) \neq J_{t+1}(c) \implies \rho^*_\Delta(c) = 1. \tag{7}$$

A relevant update can also warrant review while preserving the state. Records and update registrations precede future sources. The reader commits to a review decision from the update event, then uses permitted current sources to reassess selected propositions; unselected ones retain their recorded state. Historical judgments keep their original versions.

4.3 Scoring and Evaluation

Across all three tasks, a supported answer must be correct and have sufficient evidence bound to its target. Each contrast pair $p = (u_a, u_b)$ controls a diagnostic difference, such as execution reachability, a check precondition, or update relevance. References fix three binary tests for each response: A_u for answer correctness, E_u for evidence sufficiency, and B_u for material binding. Item and pair scores are

$$s(u) = A_u(\widehat{y})\, E_u(\widehat{y}, \widehat{E})\, B_u(\widehat{E}, \widehat{b}), \qquad s(p) = s(u_a)s(u_b). \tag{8}$$

Equation 8 is conjunctive by design: all three criteria must hold for a supported answer. The controlled interventions in Section 5.2 hold diagnostic answer correctness fixed while changing retained grounds, linking these information requirements to subsequent use. A pair passes only if both responses satisfy all three tests. Pairing fixes the unit of resolution: two diagnostic alternatives that must be distinguished, or two required judgments that jointly complete a check. The grouping is set from the task before observing reader answers. E_u admits alternative valid passages or derivations. B_u binds them to the right claim, requirement, execution, and version. A faulty locator is tracked separately if the retained premises still establish the judgment; a wrong run or governing rule fails material binding. Meaning-preserving pairs use the same product and are reported separately as invariance controls. Errors and timeouts score zero and remain in the denominator.

Let $\mathcal{G}_k$ be the independent task clusters for family k, and P_h the contrast pairs in cluster h. With $s(p)$ evaluated under fixed g, r, π, B, the primary score is

$$\mathrm{DiagRes}_k(g, r; \pi, B) = \frac{1}{|\mathcal{G}_k|} \sum_{h \in \mathcal{G}_k} \frac{1}{|P_h|} \sum_{p \in P_h} s(p). \tag{9}$$

Resolution rates are reported as percentages. We also report the pair-weighted mean, component errors, and denominators. Clusters keep every variant of an original task together; cross-task pairs join their tasks into a connected component. Confidence intervals and paired method differences resample these clusters. Scores remain separate across the three task families, readers, and access policies.

Procedure. Fix source inventory and prior exposure, reserve task clusters, and develop questions on a separate partition. Freeze requirements, questions, references, and comparisons. Construct records before exposing methods to follow-up questions or future updates. Then read, independently assess, and aggregate all planned pairs under declared access and budget.

5 Evaluating the Diagnostic Resolution Protocol

We evaluate whether DiagRes captures distinctions that matter for later diagnostic use. The studies cover record selection, controlled changes to retained evidence, and record revision. Failure-label agreement provides a control for label-equivalent records. Record construction, reading, and assessment use separate contexts, and follow-up readers never see probe answers or DiagRes scores. Overlapping source sets are analyzed separately.

5.1 Selecting Records for Subsequent Use

We evaluate 22 tasks from 14 project clusters, with three generated diagnoses per task, four support-probe pairs per record, and 163 distinct follow-up questions. In the mixed-model pool, Astra produces standard and handoff records, while Sol produces causal records. All builders receive the same source material and a 300–400-word target, but not the probes or follow-up questions. Separate Astra and Sol readers then answer the follow-up questions using only the selected records, without source access.

Label-only selection ranks records by agreement with the source failure label, while DiagRes-S uses the average score over four support-probe pairs. Ties are handled by averaging over all highest-scoring candidates; random selection averages over all three records. We report task-macro correctness so that each task contributes equally, regardless of its number of follow-up questions. This setup tests whether DiagRes-S identifies records that are more useful for questions unseen during record construction.

Table 1: Task-macro follow-up correctness (%) for mixed-model record selection on 22 tasks.

Selection	Astra reader	Sol reader
Label-only	39.2	41.9
DiagRes-S	45.2 (+6.0)	47.6 (+5.7)

Table 1 reports absolute correctness, while Figure 3 shows task-level changes. DiagRes-S improves task-macro correctness over label-only selection by 6.0 and 5.7 percentage points for the two readers, with gains in the same direction for both. Results are reported separately because reader identity is part of the evaluation condition. A separate exploratory expansion tests 72 natural and edited records across 12 tasks. Project-balanced score–use concordance reaches 69.0%, compared with 50% for labels, while defect ordering is weaker at 56.9%.

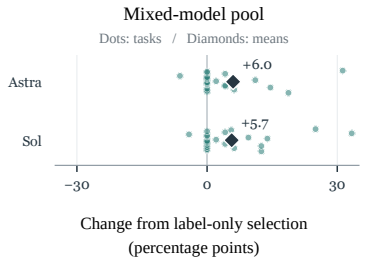


Figure 3: Record selection in the mixed-model pool. Dots show task-level changes from label-only selection. Diamonds show mean gains across the same 22 tasks for both readers.

5.2 When Correct Labels Are Not Enough

Controlled interventions on retained grounds. We construct two real-source cases in which the failure label remains fixed while the retained evidence changes (Figure 4). A builder with access only to the source material produces complete, material-mismatch, reordered, and locator-only records before seeing the diagnostic probes or follow-up questions. Separate readers then assess and use these records. This setup isolates the effect of changing the retained grounds while keeping the diagnostic conclusion and scoring rule fixed.

In Pandas, substituting performance observations from another run for the target-run evidence diverts the reader from the empty-input check that reproduces the crash. In Pillow, retaining the final reward rule resolves both follow-up judgments, whereas substituting an intermediate rule leaves them unresolved. All eight records retain the correct failure label, but material mismatches fail the evidence and binding criteria and no longer support the correct follow-up decisions; complete, reordered, and locator-only records remain usable (Table 2). The resulting scores agree with a separate source audit on all eight diagnostic answers. Here, A measures answer correctness, while AEB additionally requires sufficient evidence tied to the correct claim, execution, and rule.

Restoring facts in historical reports. Across 23 public tasks, we compare each historical report with two matched-length variants: one restores a key fact from the source, while the other adds unrelated text. Four support-probe pairs and three to six follow-up questions per task yield 103 distinct use questions, with each condition read in a separate context. Restoring key facts raises task-macro correctness from 94.9% to 98.9%, compared with 93.8% for unrelated additions (Table 3); four tasks improve and none decline. The restored facts come from retained code, logs, or trajectory artifacts and enable specific downstream judgments that the original reports leave unresolved.

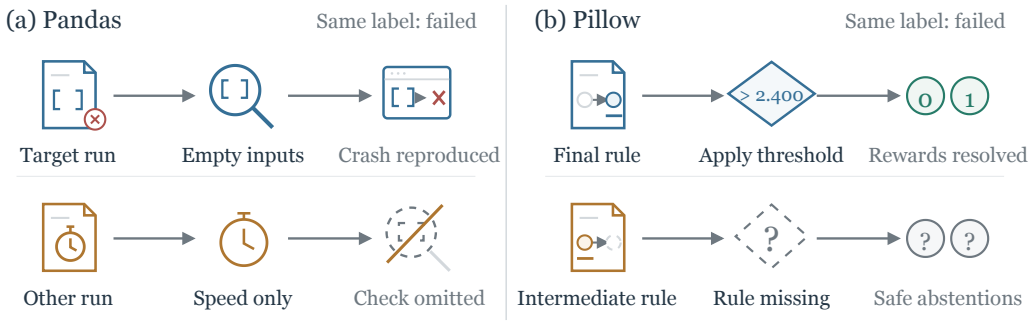


Figure 4: **The same failure label can lead to different next steps.** (a) Target-run evidence directs a check that reproduces an empty-input crash; another run's performance evidence diverts that check. (b) The final reward rule resolves two judgments; an intermediate rule leaves both unresolved.

Table 2: Controlled material interventions. Use accuracy is per record, with one Pandas and two Pillow questions.

Source	Record	A	AEB	Use accuracy
Pandas	Each control	1	1	100%
	Material mismatch	1	0	0%
Pillow	Each control	1	1	100%
	Material mismatch	1	0	0%

Table 3: Restoration correctness across 23 tasks and 103 questions. Pooled and task-macro percentages use different weights; the same questions are used in every condition.

Condition	Pooled (%)	Task macro (%)
Original report	95.1	94.9
Key fact restored	99.0	98.9
Unrelated addition	94.2	93.8

Equal fact budgets and complete decisions. Cerberus, Marshmallow, Voluptuous, and Dacite each provide two configured checks requiring two decisions. Starting from the same base description of option semantics, we add exactly two true facts in eight words under two conditions: either both facts complete one check, or one fact is assigned to each check. Local executions provide reference answers for 32 independent two-decision jobs, where joint success requires both decisions to be correct. Individual-decision accuracy is 50% in both conditions, but joint success and pair resolution reach 50% when one check is complete and 0% when the facts are split across checks, over 16 jobs per condition. For example, in Cerberus, `required=False` and `nullable=False` jointly resolve the missing-field and null-value decisions. Fact groupings are fixed before reading.

5.3 Using Feedback to Revise Diagnoses

We evaluate whether DiagRes feedback helps improve existing diagnoses. Across 19 public sources from 15 project groups, we compare original records, self-revision, and DiagRes-guided revision. Both revision methods receive the same source and original record under a 400-word output limit; DiagRes additionally receives feedback from four paired probes, capped at three issues and 225 words. A fresh reader then answers six follow-up questions per source that were unseen during revision. All stages use Astra / high in separate contexts.

Across 114 follow-up questions, correct coverage rises from 59.6% for the original records to 78.9% after DiagRes-guided revision. All resolved answers are correct, so the gain comes from answering previously unresolved questions rather than changing already correct ones. DiagRes also improves over self-revision on five sources (Figure 5). These gains reflect two effects: adding missing premises and preserving useful information that ordinary rewriting can lose.

In WideSearch, retaining the project-name/completion-time matching key and deduplication rule raises correctness from 66.7% to 100% over self-revision: one question gains a missing premise, while another preserves a premise lost during rewriting. In SLD, correctness increases from 33.3% for the original record to 50% after self-revision and 66.7% after DiagRes-guided revision. The additional answer combines the conversion $R^2 = 1 - \mathrm{NMSE}$ with the strict threshold $R^2 > 0.995$ for a new validation result. Together, these cases show how revision can both restore missing premises and preserve rules needed for later use.

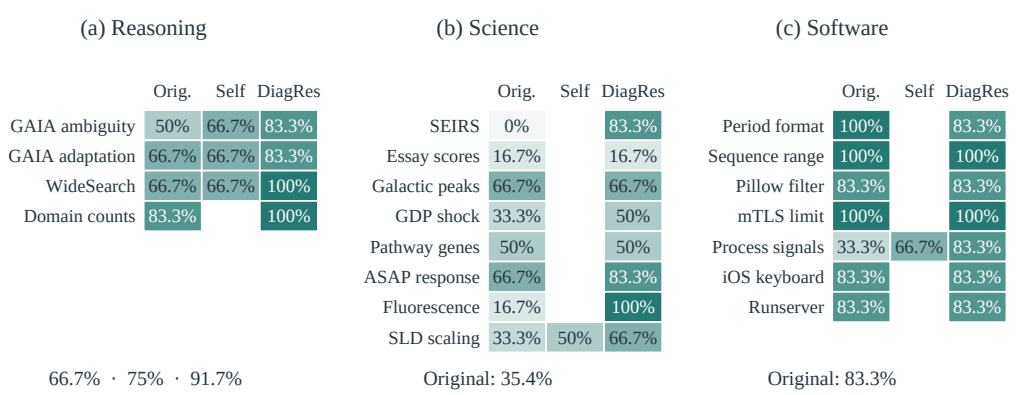


Figure 5: **Feedback-guided revision.** Each displayed cell gives correctness (%) on six questions per source. Relative to self-revision, DiagRes improves correctness by 16.7 percentage points in the predefined reasoning group. Unknown answers count as unresolved.

6 Discussion and Limitations

Our results suggest that diagnostic quality depends on what a record enables a later reader to decide, not only whether its original conclusion is correct. Failure labels can remain unchanged while the evidence needed for subsequent use is lost, and restoring missing premises can recover later judgments. The equal-budget study further shows that information quantity alone is insufficient: records with the same fact budget and individual-decision accuracy can differ in whether they support a complete decision. Reusable diagnosis therefore requires preserving coherent grounds for later judgments under the reader, source access, and budget that define its use.

The findings also separate forms of reuse that are easily conflated. Template similarity does not determine check applicability: similar failures may require different procedures, while the same procedure can transfer across different patterns. Likewise, unchanged citations do not guarantee continued validity when new evidence or changed dependencies alter the warranted judgment. Revision introduces a related tension: useful records must add missing premises without discarding conditions already needed for later decisions. Together, these results suggest that diagnostic reuse depends on preserving not only *why* a judgment holds, but also *where* and *when* its supporting evidence and procedures remain applicable.

These conclusions remain bounded by the evaluation setting. The strongest natural selection evidence is for DiagRes-S, while applicability and reassessment rely more heavily on controlled and temporal studies. Experiments cover diverse trajectories and software-update settings, but remain largely software-oriented and use model rather than human readers. Moreover, DiagRes is conditional on evaluator-specified questions, requirements, admissible evidence, source access, and reading budgets, so its scores are not context-free measures of diagnostic quality. Future work should test human and cross-domain use, compare alternative diagnostic evaluators directly, and study the cost trade-offs among richer records, retrieval, and reassessment.

7 Conclusion

This work asks a different (perhaps underexplored) question of agent diagnosis: not only whether a diagnosis is correct when produced, but whether it preserves enough information for correct later decisions. We show that task outcomes, failure labels, shared patterns, and unchanged citations can each collapse distinctions that matter for subsequent use. Diagnostic Resolution operationalizes these distinctions through reader-conditional tests of support, applicability, and reassessment, requiring answers to be correct and grounded in evidence bound to the relevant requirement, execution, and version. Across selection, controlled information loss, restoration, and revision, the results point to a common conclusion: reusable diagnosis depends less on retaining more information than on preserving the complete grounds needed for later judgments. DiagRes makes this requirement explicit and measurable.

AI USE STATEMENT

Large Language Models (LLMs) were used in this work as assistive tools for writing and polishing the manuscript. Specifically, LLMs were used to improve the clarity and fluency of the text and to assist with LaTeX formatting (e.g., tables and minipages).

REPRODUCIBILITY STATEMENT

We provide the DiagRes evaluation toolkit and scripts in the supplementary material. The toolkit supports custom diagnostic records, readers, and assessment criteria.

REFERENCES

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. Agentrx: Diagnosing ai agent failures from execution trajectories. *arXiv preprint arXiv:2602.02475*, 2026.

Daniel Deutsch, Tania Bedrax-Weiss, and Dan Roth. Towards question-answering as an automatic metric for evaluating the content quality of a summary. *Transactions of the Association for Computational Linguistics*, 9:774–789, 2021. doi: 10.1162/tacl_a_00397. URL https://aclanthology.org/2021.tacl-1.47/.

Jon Doyle. A truth maintenance system. Technical Report AIM-521, Massachusetts Institute of Technology, 1979. URL https://hdl.handle.net/1721.1/5733.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6465–6488. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.398. URL https://aclanthology.org/2023.emnlp-main.398/.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1307–1323, 2020. doi: 10.18653/v1/2020.findings-emnlp.117. URL https://aclanthology.org/2020.findings-emnlp.117/.

Yuyao Ge, Lingrui Mei, Zenghao Duan, Tianhao Li, Yujia Zheng, Yiwei Wang, Lexin Wang, Jiayu Yao, Tianyu Liu, Yujun Cai, et al. A survey of vibe coding with large language models. *arXiv preprint arXiv:2510.12399*, 2025.

Milan Gritta, Debjit Paul, Xiaoguang Li, Lifeng Shang, Jun Wang, and Gerasimos Lampouras. Process evaluation for agentic systems. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 2678–2692, 2026.

Hamel Husain and Shreya Shankar. AI Evals: Everything you need to know. Hamel's Blog, 2026. URL https://hamel.dev/blog/posts/evals-faq/. Published September 18, 2026; updated September 21, 2026.

Yeonjun In, Mehrab Tanjim, Jayakumar Subramanian, Sungchul Kim, Uttaran Bhattacharya, Wonjoong Kim, Sangwu Park, Somdeb Sarkhel, and Chanyoung Park. Rethinking failure attribution in multi-agent systems: A multi-perspective benchmark and evaluation. *arXiv preprint arXiv:2603.25001*, 2026.

Jiazheng Li, Emine Yilmaz, Bei Chen, and Dieu-Thu Le. Towards self-improving error diagnosis in multi-agent systems. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2063–2077, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of ai research. *Nature*, 651(8107):914–919, 2026. doi: 10.1038/s41586-026-10265-5. URL https://www.nature.com/articles/s41586-026-10265-5.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Dover: Intervention-driven auto debugging for llm multi-agent systems. *arXiv preprint arXiv:2512.06749*, 2025.

Kazuki Nakayashiki. When stale constraints go unchecked: Budgeted verification failures in inherited agent memory, 2026. URL https://arxiv.org/abs/2608.25553.

OpenAI. Introducing SWE-bench Verified, August 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Hezhe Qiao, Hanghang Tong, Ee-Peng Lim, Bing Liu, and Guansong Pang. Verifymas: Hypothesis verification for failure attribution in llm multi-agent systems. *arXiv preprint arXiv:2605.17467*, 2026.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4902–4912, 2020. doi: 10.18653/v1/2020.acl-main.442. URL https://aclanthology.org/2020.acl-main.442/.

Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, Alex Wang, and Patrick Gallinari. QuestEval: Summarization asks for fact-based evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6594–6604. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.529. URL https://aclanthology.org/2021.emnlp-main.529/.

Lin Shi, Haowei Lin, Zixuan Zhu, Xiaoyue Zhou, and Xiang Li. Introducing harbor-index: A compact, diverse, challenging, and high-quality benchmark for agentic evaluation. https://harbor-index.org/blog/harbor-index, 2026. Terminal-Bench blog.

Haofei Sun and Lin He. When memory updates but behavior does not: Repairing implicit stale dependencies in personalized agent responses, 2026. URL https://arxiv.org/abs/2608.01619.

Yawen Wang, Wenjie Wu, Junjie Wang, and Qing Wang. From flat logs to causal graphs: Hierarchical failure attribution for llm-based multi-agent systems. *arXiv preprint arXiv:2602.23701*, 2026.

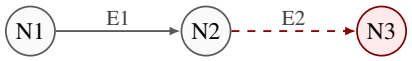
Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025.

A Representative Cases and Supporting Evidence

Each case links its task and observed sequence to a diagnostic distinction. Node labels (N1, N2, ...) and edge labels (E1, E2, ...) restart in each case and their meanings appear below the graph. Solid edges denote the stated execution or evidence relationship. Dashed edges mark a missing justification or a contradiction, and red nodes mark the affected claim or output. Blue nodes mark an improvement or a conclusion supported by new evidence. Source locations are listed in the supplementary case guide.

A.1 Django: Reporting Unobserved Controls

A Django task requests a regression test for the development server's address formatting. The agent's first assertion fails, preventing two later controls from executing. After acknowledging that their results are unknown, the agent reports both as passing. The task verifier accepts the submission because the regression test exposes the expected failure.



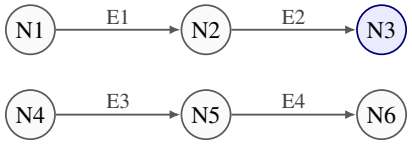
Nodes. N1: first assertion fails. **N2**: later controls are unobserved. **N3**: final report says the controls passed.
Edges. E1: the failure stops the test before the controls. **E2** (dashed): the reported passes lack execution evidence.

Figure 6: Unexecuted Django controls are reported as passing.

Diagnostic pattern. Task acceptance coexists with an unsupported report. Checking the report requires the stopping assertion and execution order.

A.2 SEIRS: A Repair Outside the Scoring Inputs

The task asks for a fast epidemic-model solver. A specified patch returns the exact solution when both initial exposed and infected populations are zero. Analytic checks improve from 4/8 to 8/8; all 56 positive-infection control outputs remain identical, and both versions conserve mass at all 64 settings. The scorer uses positive initial populations, bypasses the patch, and returns zero for both versions on the same failing integration path.



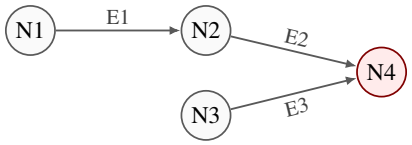
Nodes. N1: zero-infection input. **N2**: exact-solution guard. **N3**: 4/8 to 8/8 analytic passes. **N4**: positive scoring input. **N5**: new guard bypassed. **N6**: reward remains zero.
Edges. E1: the input enters the new guard. **E2**: the patch corrects the output. **E3**: the scoring input excludes the guard. **E4**: the unchanged path preserves the score.

Figure 7: A local SEIRS repair lies outside the scorer's input coverage.

Diagnostic pattern. A local correction is invisible to a scorer that never exercises it. Each analytic check passes when every state component is within $10^{-8} + 10^{-5}|y_j^{\mathrm{ref}}|$ of the reference solution. The supplementary local-repair records specify the grid and scoring reconstruction.

A.3 QUTEBROWSER: A SUCCESSFUL-EXIT LABEL MISMATCH

The task requires the state label `exited successfully` after a process exits normally. The agent deliberately keeps `successful` instead. Its local assertion checks the old label, and the tool output reports `state: successful`. Comparing the required and observed strings establishes the mismatch.



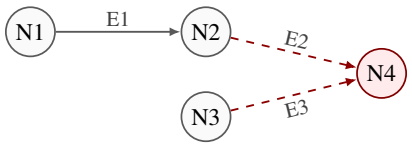
Nodes. N1: decision to keep the old label. **N2**: observed `successful`. **N3**: required `exited successfully`. **N4**: output violates the requirement.
Edges. E1: the decision preserves the old output. **E2**: the observed value supplies one side of the comparison. **E3**: the task supplies the required value.

Figure 8: Keeping the old qutebrowser label violates the required output.

Diagnostic pattern. An incorrect choice propagates to the output. The applicable check compares the observed output with the task's required literal.

A.4 EXPENSIFY: INFERRING HISTORICAL PREFERENCE

The task asks which proposed fix addresses a menu option that fails to open when Enter is pressed. The agent inspects the current `PopoverMenu` implementation, selects a matching proposal, and claims this confirms the maintainers' past preference. Establishing that history requires a decision record linking the proposal to the maintainers' choice; the supplied evidence lacks that link.



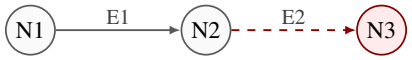
Nodes. N1: current menu code. **N2**: code resembles the selected proposal. **N3**: historical decision record absent. **N4**: report claims past preference is confirmed.
Edges. E1: code inspection establishes resemblance. **E2** (dashed): resemblance does not establish past preference. **E3** (dashed): the required historical link is missing.

Figure 9: Present code does not establish a historical maintainer preference.

Diagnostic pattern. An unsupported historical inference reaches the final report. Verifying it requires a dated decision or revision record, independently of the proposal's technical merits.

A.5 DJANGO: AN ASSERTION THAT WAS NEVER REACHED

The task requests regression tests for the development server's address formatting. In this run, an address assertion precedes an assertion about arguments passed to a mocked server runner. The traceback stops at the address assertion, but the final report claims the runner arguments were confirmed. The test body and traceback establish that the later check never ran.



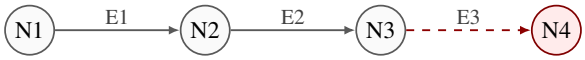
Nodes. N1: address assertion fails. **N2**: runner-argument assertion is unreached. **N3**: report claims the arguments were confirmed.
Edges. E1: the first failure halts execution. **E2** (dashed): the unreached assertion cannot support the reported confirmation.

Figure 10: An earlier Django failure blocks the assertion cited in the report.

Diagnostic pattern. The report converts a planned check into a claimed observation. Verification traces whether execution reached that operation. This is a separate run from Appendix A.1.

A.6 SEIRS: A Bypassed Solver Shortcut

In an epidemic-solver trajectory, the agent reports that a comparison over 40,000 time units used a steady-state shortcut and matched the reference. The active implementation enters that shortcut only above 50,000, so this input bypasses it. A later revision changes the threshold, making the active code version essential to interpreting the report.



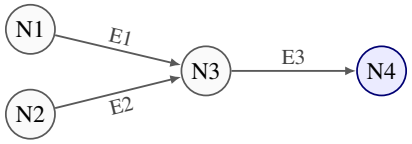
Nodes. N1: time span 40,000. **N2**: active threshold above 50,000. **N3**: shortcut bypassed. **N4**: report claims the shortcut ran.
Edges. E1: the input is checked against the active guard. **E2**: the false condition excludes the shortcut. **E3** (dashed): the report contradicts that execution condition.

Figure 11: The SEIRS input bypasses the shortcut claimed in the report.

Diagnostic pattern. A numerical comparison is attributed to a branch it cannot exercise. Checking the claim requires the operation's code version, input, and entry condition.

A.7 Flask: Newly Available Constructor Evidence

The requirement is to raise `ValueError` for an empty Blueprint name with otherwise valid arguments. Initially, the supplied sources omit the constructor, leaving the judgment unknown. The diagnosis registers the missing file and source inventory/version. An update exposes the constructor's `if not name:` branch and `ValueError`; source inspection then supports the requirement. The issue and history passages cited initially remain unchanged.



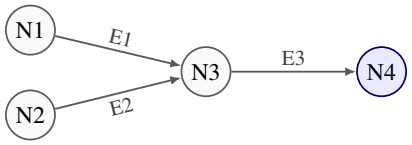
Nodes. N1: missing constructor registered. **N2**: constructor becomes available. **N3**: judgment selected for review. **N4**: rejection of the empty name is supported.
Edges. E1: the registration identifies what to watch. **E2**: new availability triggers review. **E3**: the new source resolves the judgment.

Figure 12: New Flask constructor evidence resolves a previously unknown judgment.

Update pattern. New evidence changes an unresolved judgment without changing its old citations. Registering missing evidence makes that update discoverable.

A.8 Astropy: A Changed Equality Method

The requirement is that an unrecognized unit compared with `None` returns false. Initially, equality lets a `TypeError` escape. The diagnosis registers the unit implementation and equality method. The updated method catches the error and returns `NotImplemented`, allowing the comparison to resolve to false. The saved source-based reference therefore changes from defect to clear.



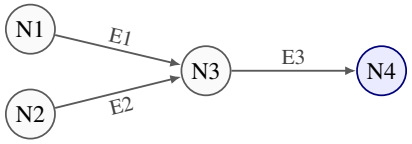
Nodes. N1: equality method registered. **N2**: method gains error handling. **N3**: judgment selected for review. **N4**: source inspection supports the corrected comparison.
Edges. E1: the registration identifies the relevant method. **E2**: the method change selects review. **E3**: inspection establishes the changed behavior.

Figure 13: An updated Astropy equality method changes the defect judgment.

Update pattern. A changed implementation invalidates the old defect judgment. The method dependency triggers review; inventory/version-only routing misses the change because neither dependency was registered for this judgment.

A.9 XARRAY: NEWLY VISIBLE ITERATOR CODE

The requirement is that a rolling iterator yields a label and window at each step. Initially, documentation describes this interface, but the implementation is missing, leaving the judgment unknown. The diagnosis registers source inventory/version alongside the documentation and issue. The update exposes `yield (label, window)`, and source inspection changes the reference judgment to clear.



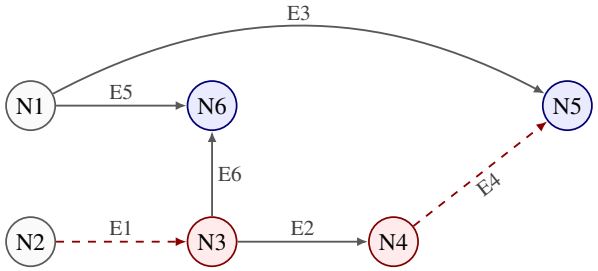
Nodes. N1: inventory/version dependencies registered. **N2**: iterator implementation becomes visible. **N3**: judgment selected for review. **N4**: label–window interface is supported.
Edges. E1: the registration tracks available sources. **E2**: changed availability selects review. **E3**: the iterator code resolves the interface question.

Figure 14: Newly visible Xarray code resolves the iterator-interface judgment.

Update pattern. A source-availability update can resolve a judgment even when the initial record did not name the method. Inventory/version dependencies make this update visible to the review procedure.

A.10 PANDAS: CORRECT FAILURE LABEL, WRONG RUN'S EVIDENCE

A merge optimization adds an integer-factorization fast path that crashes on empty arrays. A controlled diagnostic record substitutes evidence from another run whose tests complete but whose performance falls below the threshold. The diagnosis still says "failed," now with the wrong reason. A separate reader chooses a performance-only follow-up; complete, reordered, and locator-only records instead lead it to choose empty inputs.



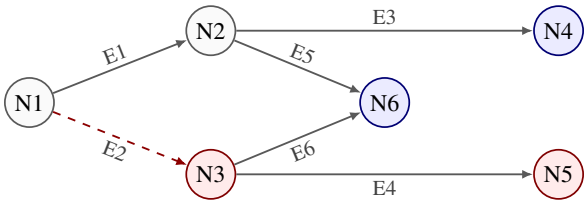
Nodes. N1: target's empty-input crash. **N2**: another run's performance shortfall. **N3**: target diagnosis adopts that run's observation. **N4**: performance-only check selected. **N5**: empty `int32` probe reproduces the crash. **N6**: evidence and binding reject the diagnosis.
Edges. E1 (dashed): controlled substitution misattributes the observation. **E2**: the altered record guides a fresh reader's check choice. **E3**: replay of the target code confirms the defect. **E4** (dashed): the selected check omits this targeted probe. **E5–E6**: comparing target evidence and claimed grounds exposes the mismatch.

Figure 15: Evidence from another Pandas run diverts the next check.

Diagnostic pattern. Label agreement preserves the outcome while losing the premise needed to choose the next check. The replay executes the extracted fast path; the hypothetical performance change is not executed.

A.11 PILLOW: AN INTERMEDIATE RULE CANNOT DETERMINE THE FINAL REWARD

A filter optimization produces an intermediate binary reward from `opt_commit`, then a wrapper writes the final reward using the continuous score and a strict > 2.400 threshold. Both rules return zero for the target. Replacing the final rule with the intermediate one therefore preserves the target's failure label but loses the rule needed for other runs.



Nodes. N1: intermediate binary reward. **N2**: final threshold overwrites the reward. **N3**: controlled record presents the intermediate rule as final. **N4**: final rewards correctly resolved as 0 and 1. **N5**: two safe abstentions. **N6**: evidence and binding reject the substituted rule.
Edges. E1: the wrapper determines the final scoring stage. **E2** (dashed): the record assigns the intermediate rule the wrong role. **E3**: the retained threshold resolves both follow-ups. **E4**: the missing final rule prevents resolution. **E5–E6**: the source rule contradicts the record's claimed basis.

Figure 16: An intermediate Pillow reward rule obscures the final scoring rule.

Diagnostic pattern. Agreement on one outcome does not establish that a scoring rule transfers. Another observed run has binary reward 1 but continuous score 0.9754 and final reward 0; a controlled score of 2.5 yields 1. The material record causes abstention, not an incorrect pass.