OUT OF STEP WITH EVIDENCE: TASK PROGRESS STATE MISMATCH IN LONG-HORIZON TOOL-USE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon tool-use agents must turn intermediate results into task progress that later work can rely on. Reliability depends on keeping their treatment of completed and outstanding work aligned with the evidence. We study task progress state mismatch at points where local results become premises for subsequent work. Existing evaluations measure completion and partial progress, while trajectory diagnosis and state auditing examine errors, recovery, and evidence. Our analysis connects these perspectives by distinguishing what a local result supports, how the agent handles the relevant task state, and what happens to that state afterward. We introduce ORPMTF, an event-level framework that separates two judgments: whether the agent's updates stay within what the evidence permits and whether they enact the changes required by the task and evidence. We analyze 5,433 trajectories with 13,241 annotated state-update events across six task environments, combining descriptive analysis of machine annotations with close readings of selected trajectory segments. The annotation patterns distinguish two forms of mismatch: updates can remain within the permitted range while leaving required changes undone, or enact those changes while introducing unsupported changes. Close readings distinguish requirements unresolved during continued activity from the conditions restored by repair: recovering an artifact or access to one service need not satisfy the prerequisites of the next operation. Together, these analyses identify what agents can carry forward, what must remain open, and what needs revision before subsequent work relies on the resulting task state.

1 INTRODUCTION

Long-horizon tool-use agents are being developed to complete extended tasks without step-by-step human supervision, from repairing codebases to handling transactions across business applications. In these settings, agents must turn intermediate results into task progress that later work can rely on. Test evidence used to justify a code submission must apply to the submitted version. An agent relying on an update in a business application must track which account or transaction that update concerns. Agents must keep track of what is complete, what remains open, and what new evidence requires them to reconsider. When these judgments fall out of step with the evidence, agents can proceed on unsupported assumptions or leave necessary changes undone. Users may then need to revisit earlier steps to determine which results they can trust. Reliable task progress therefore matters both for completing the delegated work and for reducing the need for continuing human supervision.

To judge task progress reliably, agents must interpret local results against the task's remaining requirements. Consider a test that passes before further edits: the pass remains valid evidence about the earlier version, but does not establish that the submitted version works. The required update is to treat validation of the edited version as unresolved, not necessarily to rerun the test immediately. Recognizing an unmet requirement is distinct from completing the work needed to satisfy it. An agent can thus handle a failed call appropriately while leaving repair pending. Local correctness, the appropriateness of task-state updates, and final task outcomes require separate assessment.

Prior work on agent evaluation measures task completion and partial progress using completion scores and intermediate milestones (Ma et al., 2024; Lu et al., 2025). Research on error localization

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.

and process diagnosis identifies problematic steps and tracks error persistence and recovery (Zhao et al., 2026; Qi et al., 2026). Studies of state maintenance and evidence auditing also connect task requirements, verification, and execution (Zhang et al., 2026; Salas, 2026). Building on this work, we analyze task-state mismatch: agents may treat unsupported progress as established or leave required changes undone after a local result. An error label or progress score alone does not capture this distinction: the same error category can describe a failure handled appropriately or one that leaves a required change undone. Likewise, reporting a successful repair does not itself specify what was restored or which prerequisites for subsequent work remain unmet. The resulting diagnostic question concerns the state carried forward: which requirement remains open, which object or version a result supports, and whether the next operation's prerequisites are met.

We examine whether agents treat work as complete, open, or needing revision in line with available evidence. We refer to this consistency as *task progress state reliability*. Evidence that an item is incomplete requires the agent to stop treating it as complete, although several ways to investigate or fix it may be reasonable. Reliability requires both evidence-supported updates and enactment of required changes; satisfying one of these conditions does not guarantee the other. We call either failure a *task progress state mismatch* and trace whether it persists or is corrected in later work.

We introduce ORPMTF, an event-level framework connecting local evidence, task-state handling, and subsequent execution. We focus on trajectory points where results or feedback affect what the agent can treat as complete, keep open, or revise. For each event, we record the relevant task requirements (O) and local result (R), and identify the updates the evidence permits (P) and requires (M). We compare these with the updates reflected in the agent's observable behavior (T). We then trace how the resulting task state is used, revised, or repaired in later work (F). We judge state updates using contemporaneous evidence and track later developments separately. In Figure 1, node a can represent reopening validation after a code edit; Section 3 formally defines the fields.

We manually reviewed 600 trajectories across six task environments, checking interpretations of local breakdowns against trajectory evidence and refining category definitions and boundaries. This cross-case analysis informed the development of ORPMTF. We also analyze 5,433 trajectories with 13,241 machine-annotated state-update events covering application workflows, conversational service, software engineering, and terminal tasks. The corpus analysis characterizes combinations of the two judgments across environments and relates state alignment to task outcomes within model groups. Close readings examine distinctions these summaries cannot resolve: whether continued activity changes an outstanding requirement, whether work remains open after an operation error, and whether a repair restores the object, version, or access condition required by the next operation.

We make three contributions. First, we introduce ORPMTF, an analytical framework for event-level assessment of permission, required enactment, and subsequent use. Second, we combine manual review of 600 trajectories for framework development with a cross-environment corpus and descriptive analysis of machine annotations. Third, our case analyses distinguish operation failure from mishandled task state and local repair from satisfaction of the next operation's prerequisites, identifying both valid results to retain and requirements still unresolved at the next point of use.

2 Related Work

Benchmarks span application, service, software, and terminal tasks (Trivedi et al., 2024; Li et al., 2026; Barres et al., 2025; Khandpur et al., 2025; Yang et al., 2025; Huang et al., 2026; Merrill et al., 2026). Evaluation methods distinguish several aspects of performance. AgentBoard (Ma et al., 2024) measures partial progress, and ToolSandbox (Lu et al., 2025) evaluates state-dependent milestones. Ko et al. (2026) use an Epistemic Ledger to distinguish evidential support from agents' beliefs about constraint satisfaction. VIGIL (Chen et al., 2026) separates task completion in the world from the correctness of agents' terminal reports, while Procedure-Aware Evaluation (Cao et al., 2026) distinguishes goal achievement from procedural compliance. We place these comparisons at individual execution events, asking which task-state updates the task and available evidence permit or require, which changes the agent actually enacts, and how the resulting state is later used.

Failure diagnosis connects erroneous steps to their downstream consequences. AgentRx (Barke et al., 2026) uses constraint-violation records to localize critical failures. Failure as a Process (Zhao et al., 2026) and TRAJDEBUG (Qi et al., 2026) analyze how errors emerge, persist, and are resolved,

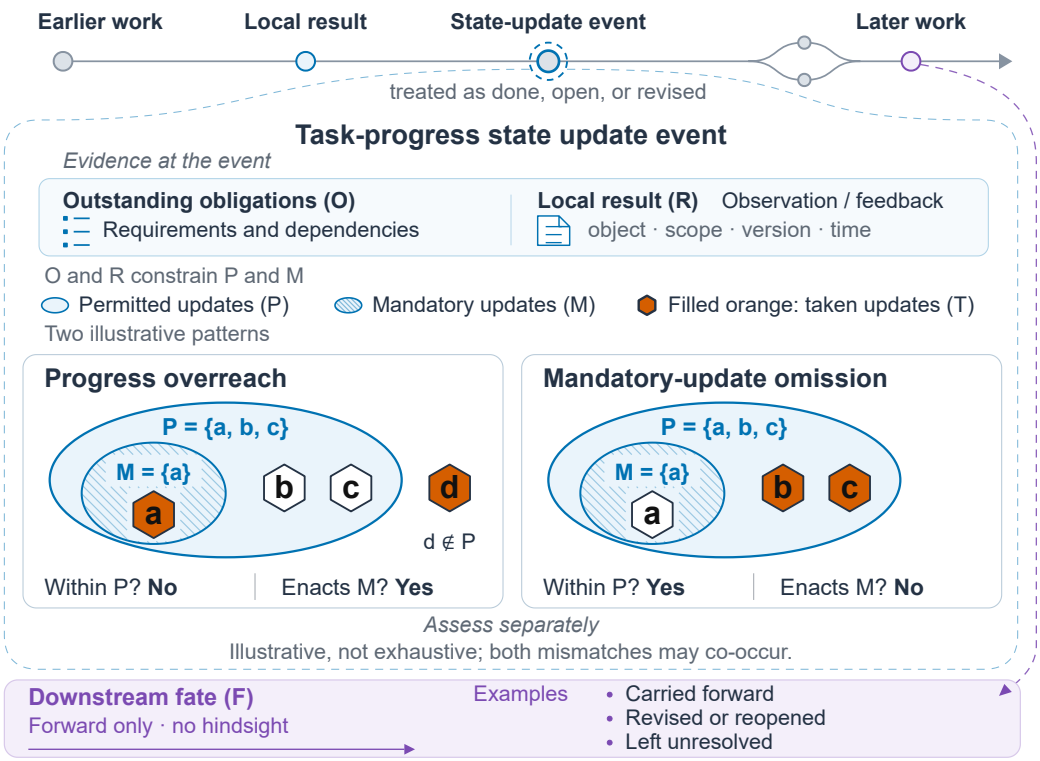


Figure 1: **ORPMTF and two mismatch patterns.** Filled orange hexagons mark updates taken; open hexagons mark updates not taken. Left: all required updates are made, but an unpermitted update is also made. Right: all updates made are permitted, but a required update is omitted. The two mismatches can co-occur.

while DRIFT (Wang et al., 2026) traces unsupported or conflicting claims that affect subsequent answer construction. We examine how agents handle the associated requirement: whether they keep unfinished work open or treat an unmet prerequisite as established. Operation failure can coexist with appropriate state handling, and operation success with unsupported claims of progress.

Task-state maintenance and verification methods connect requirements, evidence, and execution. iCORE (Zhang et al., 2026) links work-state transitions to obligations and supporting evidence, while Matrix (Salas, 2026) records provenance, verifies completion evidence, and marks dependent work as stale when governing rules or facts change. AgentGuardUtil (Bouchekir et al., 2026) identifies unmet policy obligations and guides revisions to proposed actions, and ContextFlow (Guo et al., 2026) coordinates task stages and repairs through evidence-grounded contracts. Building on these evidence–state links, ORPMTF separately assesses whether taken updates are permitted and required changes are enacted at the same event. These comparisons diagnose recorded trajectories, tracking subsequent use and repair without letting later outcomes revise event-time judgments.

3 A Diagnostic Framework for Task-State Reliability

We identify the task requirement affected by a local result, compare permitted and required updates with the agent's observable response, and separately track the state's subsequent use or revision.

3.1 From Local Breakdowns to State-Update Events

From a review of 600 trajectories across six environments, we organized ten working classes of local breakdowns using existing annotation vocabulary and model-proposed groupings (P01–P10; Table 14). The same class of breakdown can elicit appropriate state handling or an unsupported completion claim. Distinguishing these responses requires examining the affected requirement, ev-

idence, and response at a specific trajectory point. We therefore analyze task progress state update events; their class associations may be multiple or absent and provide descriptive context rather than determine state-update judgments. Appendices C and D detail development and the vocabulary.

3.2 State-Update Events and ORPMTF

A *task progress state update event* occurs when a local result affects how the agent treats, or should treat, a task requirement. An event need not involve an actual update: the agent may reuse a status, close or reopen a requirement, or leave a required change undone. We locate events through the requirement, result, and response: statements, state changes, or actions requiring a task status, rather than a tool call alone. Probing does not show that the agent treats its target condition as satisfied.

ORPMTF links O/R to permitted and required updates (P/M), actual handling (T), and later fate (F):

- **O: Outstanding obligations.** The task requirements and dependencies relevant at the event, including previously satisfied items whose continued validity is now at issue.
- **R: Local result.** The observation, feedback, artifact, or intermediate conclusion under consideration, with its support status and applicable object, scope, version, and time.
- **P: Permitted updates.** The task-state changes justified by O, R, and the evidence available at that point. P includes alternative updates and the conditions for combining these updates.
- **M: Mandatory updates.** The changes within P that the task requirements or evidence make necessary, such as withdrawing a refuted conclusion or reopening a dependent requirement. M is a subset of P; permission to advance does not itself require advancement.
- **T: Taken updates.** The task-state updates expressed by the agent's statements or behavior in the immediate observation window. T can be empty when the observation window establishes that no update occurred; if observation is insufficient, T remains unobservable.
- **F: Downstream fate.** How later work uses the event's task state after the immediate response, and whether that state is retained, revised, reopened, repaired, or left unresolved.

A task requires validation of the submitted code version (O). Suppose the old version passes a test, then edits produce a new version (R); we assess the response to those edits. P permits retaining the old result within its version scope and reopening validation; M requires treating the new version as unverified, without requiring an immediate retest. If the agent states "new version awaiting validation" (T), it stays within P and enacts M. If it declares the new version validated solely from the old test, it exceeds P and omits M. Checking the new version alone does not assert validation. F tracks subsequent use or revalidation of that version. Appendix E gives field-level evidence rules.

3.3 Assessing Mismatch and Subsequent Fate

Progress overreach occurs when T contains an update outside P or violates P's combination conditions. *Mandatory-update omission* occurs when an established required change in M remains unenacted, with sufficient opportunity and visibility to assess enactment. Annotation models derive P from task requirements, R, and contemporaneous evidence; they identify M from explicit rules, user instructions or authorization, and observed prerequisite changes, recording each update's basis. They reconstruct T from status statements, visible changes, or premises required by the immediate response. Two annotation models make these judgments independently, as described in Section 4.

P and M depend on contemporaneous evidence, T on the immediate response, and F on later execution. Later repair can restore the state without erasing an earlier mismatch; later failure alone does not establish one. We then trace its use, revision, repair, and any requirements left unresolved.

4 Data and Annotation

We combine machine-annotation summaries of the two state-update judgments with close readings that explain unmet requirements, downstream dependencies, and what repairs restore. Figure 2 distinguishes the distinct roles of developmental review, machine labels, and case interpretation.

4.1 Data Sources and Environment Selection

We selected six benchmarks with observable actions, intermediate results, and feedback: AppWorld and CLAW for application workflows, τ^2-bench for agent–user coordination, Terminal-Bench for terminal tasks, and DeepSWE and SWE-bench Multilingual for repository changes. Retained tool returns, service states, file changes, and validation records allow us to examine what each local result supports and which task states subsequent actions rely on. We reran tasks using a set of models: GPT-5.4 (OpenAI, 2026a); GPT-5.6 Luna, GPT-5.6 Sol, and GPT-5.6 Terra (OpenAI, 2026b); DeepSeek V4 Flash and Pro (DeepSeek-AI et al., 2026); MiMo V2.5 (Xiaomi MiMo, 2026); and MiniMax M3 (MiniMax, 2026). The corpus contains 5,433 trajectories and 13,241 machine-annotated events. Appendices A.1 and B detail the corpus, task sources, and models.

4.2 Dual Independent Machine Annotation and Adjustment

GPT-5.6 Luna performs Primary annotation and DeepSeek V4 Flash the Blind pass. Each labels local breakdowns (P01–P10) for descriptive context and ORPMTF events for state-update analysis, with located evidence spans. Blind starts after Primary completion, without access to Primary outputs or comparisons. Both read full trajectories, including source cues and later outcomes; output separation does not conceal later execution. P/M/T judgments must use event-time evidence.

We compare both layers, including breakdown labels, ORPMTF fields, and both state-update judgments. In this dual-pass procedure, agreements are consolidated; GPT-5.6 Sol adjudicates disagreements using the trajectory, both outputs, and their differences. It retains, revises, or rejects disputed annotations and marks unresolved judgments. Appendix F describes the annotation inputs, comparison targets, and checks on adopted records. These outputs remain machine-generated weak labels for retrieval and descriptive analysis, not human-validated judgments of task-state correctness.

4.3 Developmental Review and Material Screening

We reviewed 600 trajectories against prior machine annotations and source evidence, checking interpretations and revising local categories and boundaries. These decisions informed framework development; they concern the reviewed local annotations and their supporting evidence rather than human ORPMTF labels. Appendix C documents the review, its scope, and its recorded decisions.

We check readability, source linkage, completeness, and evidence locatability before analysis. Each analysis uses records supporting its required judgments; sufficient evidence for no event differs from insufficient evidence to decide. Unresolved records do not become no-event or no-mismatch cases. These checks establish material readiness for analysis, not semantic correctness (Appendix F).

5 Results

5.1 Task-State Mismatch and Task Outcomes

In all six environments, machine annotations include updates within P that leave M unenacted and updates beyond P that enact M (Table 5). The analysis uses 6,162 jointly judged events out of 13,241 annotated events. These combinations direct attention to different questions: which required change remains outstanding, and which adopted update lacks support. Reporting either axis alone would hide one of these combinations in the annotated data, obscuring which aspect of state handling needs examination. Section 6 examines the task requirements and evidence behind individual cases.

Among jointly judged events, mismatch combinations distinguish partial or absent enactment of required changes from updates outside permission. These problems can occur separately or together. A single alignment rate obscures which requirements remain unresolved and whether adopted states lack support. Table 1 separates these combinations by environment and model alongside task success; Appendices A.2 and A.4 document judgment coverage and common-task model profiles.

At trajectory level, the association with task success varies across environments. Across five environments, we compare 2,779 scored trajectories with jointly judged events. For each trajectory, we compute the fraction of these events within P and fully enacting M, then compare mean fractions between passed and failed runs. With equal model weights, passed runs exceed failed runs by 24.7

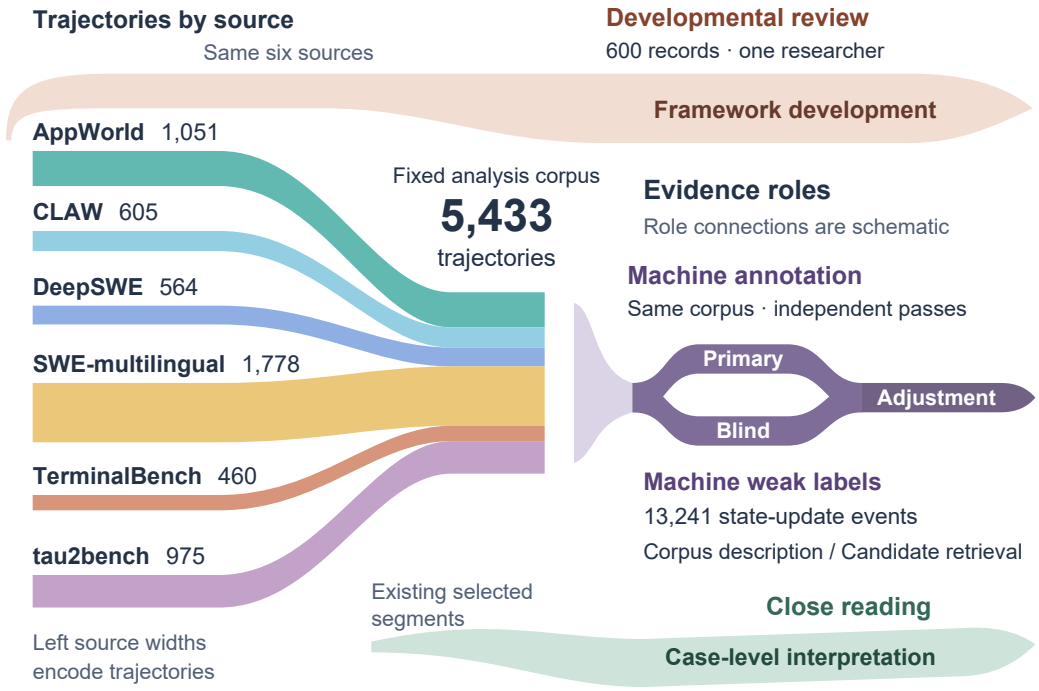


Figure 2: **Source composition and evidence roles.** Left ribbon widths encode counts in the 5,433-trajectory corpus. Links beyond the break denote nonexclusive roles, not subsets. One researcher's 600 developmental readings informed the framework but do not provide independent validation; their overlap with this corpus is unspecified. Independent Primary/Blind annotation and adjustment yield 13,241 machine-labeled events. Previously selected close readings support case interpretation.

percentage points in AppWorld, 20.2 in TerminalBench, and 17.6 in tau2bench. CLAW's smaller pooled gap is concentrated in two models. DeepSWE reverses from $+7.9$ points pooled to -3.0 with equal model weights, so pooled and model-equal comparisons yield opposite directions.

The DeepSWE pooled gap decomposes into a model-composition component of $+11.0$ percentage points and a within-model component of -3.1. The latter weights model-specific gaps by the average of each model's shares in the passed and failed groups, unlike the equal-model estimate above. DeepSeek V4 Flash and MiMo V2.5, with lower alignment fractions, account for 11 of 157 passed runs but 125 of 252 failed runs. Their unequal representation explains the positive pooled gap despite negative gaps within most models, distinguishing model composition from within-model gaps.

Excluding events with `CLOSE_TASK` in P/M/T operations or T behavior modes leaves the three pooled gaps positive. This tests dependence on annotated closure updates, not all late-trajectory events. Appendix I reports group counts, means, weights, and exclusion results for this analysis.

Recorded activity can coexist with a required change left undone. Among events within P with explicit M and resolved enactment judgments, 340 combine M not enacted with empty structured T records; 212 also record other activity (Figure 7; Appendix A.5). Empty structured T does not establish inactivity, and activity alone does not establish enactment. The following case readings examine which task requirement remains unmet and which state subsequent work relies on.

6 Case Analysis of Task-State Handling

Event distributions leave open what local work established, which state the agent carried forward, and what a repair restored. Selected case readings examine these relationships; Appendices A.5–A.7 provide further case-level analyses of recorded local activity, error handling, and recovery.

Table 1: **Task success and two-axis event composition by model and source.** The model-attributable subset contains 5,222 trajectories and 13,083 annotated events, including 6,085 jointly judgeable events. Task success is the percentage of scored trajectories passing each source's criterion. The six cells show percentages of jointly judgeable events J; event coverage is $100J/E$, with counts (J/E), for all annotated events E. ORPMTF cells use machine annotations; groups are not task-matched. Both colors use a shared linear 0–100% scale.

Model	Task success (%)	T within P: M enacted	T within P: M partial	T within P: M not enacted	T beyond P: M enacted	T beyond P: M partial	T beyond P: M not enacted	Event coverage % (J/E)
AppWorld								
GPT-5.4	86.5	88.8	1.3	3.1	0.4	0.9	5.4	88.2 (224/254)
GPT-5.6 Luna	78.3	94.4	0.6	0.0	0.0	0.0	5.1	53.1 (178/335)
GPT-5.6 Sol	94.4	96.6	1.0	1.0	1.5	0.0	0.0	84.6 (203/240)
GPT-5.6 Terra	88.2	97.4	0.4	0.9	0.0	0.0	1.3	63.7 (230/361)
DeepSeek V4 Flash	56.1	54.4	1.3	6.3	2.5	5.0	30.6	78.4 (160/204)
DeepSeek V4 Pro	86.2	82.6	0.0	0.7	2.2	1.4	13.0	75.8 (138/182)
MiMo V2.5	76.0	73.8	1.8	4.1	0.5	0.0	19.9	57.7 (221/383)
MiniMax M3	81.3	88.3	2.5	2.1	0.0	0.8	6.3	78.6 (239/304)
CLAW								
DeepSeek V4 Flash	70.0	27.2	0.0	1.8	5.3	18.4	47.4	67.1 (114/170)
DeepSeek V4 Pro	64.7	19.0	0.5	3.3	10.3	12.5	54.3	85.6 (184/215)
MiMo V2.5	67.1	47.3	0.9	4.5	6.8	12.2	28.4	69.2 (222/321)
MiniMax M3	68.0	60.9	1.7	15.5	6.3	3.4	12.1	57.4 (174/303)
DeepSWE								
GPT-5.4	52.0	80.6	0.0	12.9	0.5	0.0	5.9	46.2 (186/403)
GPT-5.6 Luna	67.2	76.7	1.2	17.4	0.0	1.2	3.5	49.9 (172/345)
GPT-5.6 Sol	72.7	69.4	2.5	24.0	0.0	0.0	4.1	43.8 (121/276)
GPT-5.6 Terra	69.6	77.1	0.5	12.6	0.0	0.0	9.8	49.1 (214/436)
DeepSeek V4 Flash	53.0	47.7	4.0	24.5	0.7	0.7	22.5	63.2 (151/239)
MiMo V2.5	7.8	49.1	2.5	20.2	0.0	0.6	27.6	56.6 (163/288)
SWE-multilingual								
GPT-5.4	73.2	81.8	1.3	6.5	0.0	1.3	9.1	15.5 (77/496)
GPT-5.6 Luna	79.8	82.7	2.7	6.7	0.0	0.0	8.0	9.7 (75/772)
GPT-5.6 Sol	85.3	91.7	3.6	3.6	0.0	1.2	0.0	23.6 (84/356)
GPT-5.6 Terra	76.6	87.5	6.3	4.2	2.1	0.0	0.0	22.3 (48/215)
DeepSeek V4 Flash	68.9	63.5	5.2	20.8	0.0	0.0	10.4	17.2 (96/557)
DeepSeek V4 Pro	70.0	58.8	7.8	15.7	0.0	0.0	17.6	15.2 (51/336)
MiMo V2.5	67.1	60.3	7.9	19.0	1.6	0.0	11.1	19.7 (63/320)
MiniMax M3	66.3	73.9	0.0	17.4	0.0	0.0	8.7	11.1 (69/620)
TerminalBench								
GPT-5.4	75.1	83.7	6.1	6.1	0.0	0.0	4.1	63.6 (49/77)
GPT-5.6 Luna	84.7	86.4	5.9	5.1	0.0	0.0	2.5	66.7 (118/177)
GPT-5.6 Sol	88.8	100.0	0.0	0.0	0.0	0.0	0.0	17.8 (21/118)
DeepSeek V4 Flash	56.9	66.7	4.4	4.4	0.0	0.0	24.4	33.3 (45/135)
DeepSeek V4 Pro	67.9	63.0	0.0	13.6	0.0	0.0	23.5	36.8 (81/220)
MiMo V2.5	65.8	56.7	1.7	8.3	1.7	6.7	25.0	53.1 (60/113)
MiniMax M3	66.0	63.2	2.6	21.1	0.0	0.0	13.2	41.3 (38/92)
tau2bench								
DeepSeek V4 Flash	88.4	90.9	0.2	2.5	1.9	0.4	4.0	48.5 (472/974)
DeepSeek V4 Pro	87.3	92.6	1.5	1.0	1.0	0.0	3.8	60.9 (391/642)
MiMo V2.5	83.7	88.3	0.2	2.8	1.3	1.1	6.4	52.9 (470/889)
MiniMax M3	84.1	83.6	2.1	6.6	1.7	0.4	5.6	67.6 (483/715)

6.1 Locating the State-Handling Problem

Local success and failure can obscure the state-handling problem that subsequent work inherits. A failed operation may leave its requirement correctly open, whereas a successful operation may support less than the agent treats as established. Table 2 distinguishes the required change left outstanding from the adopted state lacking support. The distinction matters because the two problems call for different corrections: preserving unfinished work for attention, or withdrawing an unsupported premise before further use. For X19 and X60, this preserves session and version distinctions already identified in earlier analyses; Q1C11 remains a `NO_DELTA` case. A local error may be immediately visible, while a mistaken completion state can survive into later planning and reporting. The point requiring attention may therefore lie between an operation and a later decision to reuse

its result, rather than at the operation itself. Locating the unsupported or outstanding requirement reveals where subsequent execution needs a different premise, even after the original operation has ended and its result has entered subsequent work as an assumed basis for further decisions.

Table 2: Selected cases and the task-state distinctions retained in their analysis. Summaries describe local activity, operation results, or earlier analyses; they do not form a common comparison baseline.

Case(s)	Local record or prior analysis	Task-state distinction
D10 / C04	D10: continued preparation; C04: artifact produced and checked	D10: enactment remains unresolved; C04: artifact observed, but full task correctness remains unresolved
Q2W11 / Q2W12	W11: integration rejected; W12: re-login claimed before execution	W11: integration kept open for correction; W12: unexecuted re-login used as an established premise
X60	Copy-back succeeded	A modified backup was restored; the original input was reconstructed later
X19	Phone access restored	A separate Venmo session was still required before the first Venmo use
Q1C11 (`NO_DELTA`)	Earlier case analysis documented the repair	Reorganization adds no new state-level judgment

6.2 From Local Activity to State Alignment

Visible activity provides an incomplete account of which task requirements have been satisfied. D10 contains frame extraction, OCR, and answer preparation, but no observed write to the requested `/app/solution.txt`. C04 contains a written and executed `/app/extract.js`, followed by checks of output format and selected values (Table 2). Preparation and a partially checked artifact establish different scopes of progress: D10 alone does not establish omission, and C04 does not establish full correctness. The broader risk arises when these scopes disappear from the state carried forward. If preparation is treated as delivery, or limited checks as complete verification, unfinished requirements are likely to receive less attention precisely because execution appears to be advancing. Diagnosis therefore asks which requirement remains open behind the ongoing visible activity.

An operation error becomes a different problem when unfinished work is carried forward as an established premise. In Q2W11, the agent acknowledged that a rejected integration patch had not been applied and split it for another attempt. In Q2W12, it relied on a claimed re-login that had not occurred; first use raised a `NameError` for an undefined token variable. It subsequently performed the login, but later claimed three payment requests and one payment had succeeded without corresponding calls, then invoked completion (Figure 3, group A). W11 kept the failed operation visible as unfinished work. W12 repaired one unsupported premise yet introduced others. Such claims are likely to displace the very checks needed to expose unfinished work: later execution can proceed from assumed completion rather than revisit the missing operation. This creates a route from a local discrepancy to premature task closure: a requirement drops out of consideration even though the work needed to satisfy it has not occurred. Correcting an immediate error alone does not ensure that later completion claims have evidence to support the state used by subsequent dependent work.

6.3 Recovery at the Point of Use

Repair restores particular conditions, not general readiness for subsequent execution. In Terminal-Bench X60, copying back a backup succeeded but restored a modified `INPUT.DAT`; the original input still had to be reconstructed for the program comparison. In AppWorld X19, phone-login repair enabled contact queries without establishing the separate Venmo session needed at first use. The attempted Venmo query raised a `NameError`; login and retry then succeeded (Figure 3, group B). Both cases separate a valid local result from the condition needed downstream: the required version in X60, and an independent session in X19. The repair need not be discarded, but its success cannot

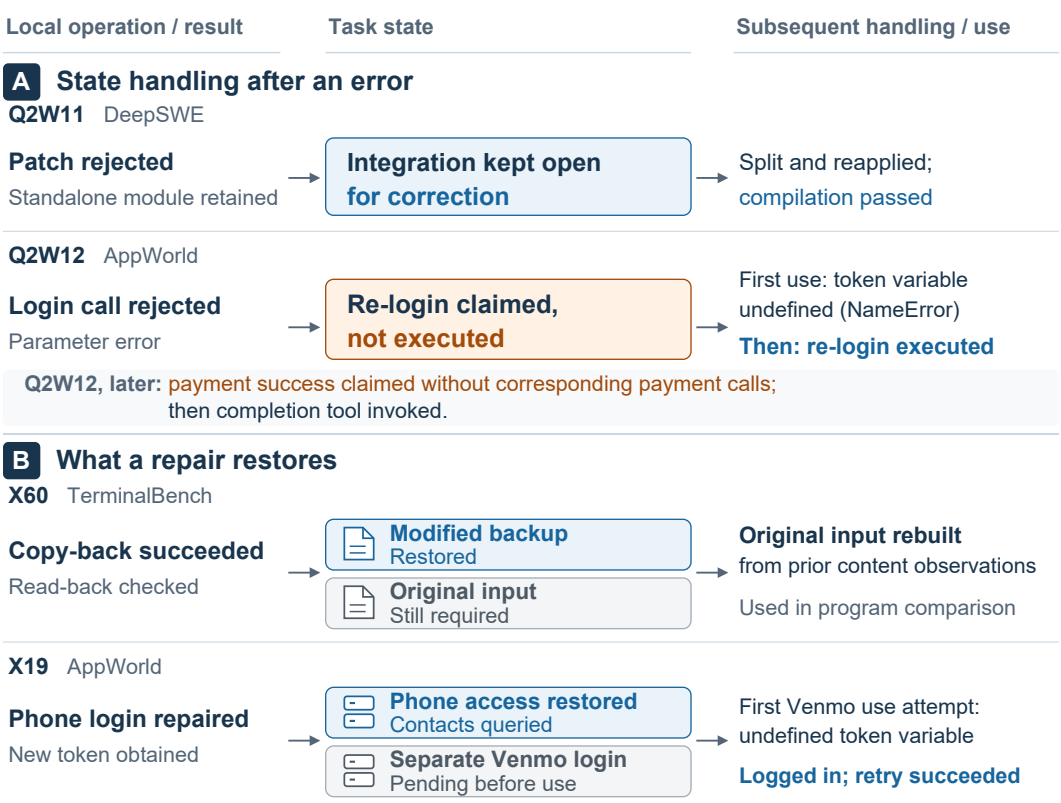


Figure 3: **Selected case readings.** Group A contrasts error handling; group B shows version and object boundaries of repair. Read each row from local result through task state to subsequent handling or use. Arrows follow recorded order; cases are not experimentally matched.

establish a different version or an unrelated prerequisite within the same task. The common problem is extending a repair beyond the conditions it actually restores at the next dependent operation.

Treating recovery as a task-wide reset is likely to reintroduce unsupported assumptions at the next dependency boundary. Retrying a repaired operation answers whether that operation now works; it does not settle every condition needed by later work. A single successful repair can therefore leave several independent prerequisites for subsequent work unresolved. In X19, retaining restored phone access while keeping the Venmo prerequisite open avoids both overextending the repair and undoing valid progress. Recovery analysis therefore follows the repaired state to its next use, checking the object, version, and prerequisite on which that use depends. This shifts attention from whether the agent has recovered to what it has recovered enough to do next. The relevant endpoint is a state whose remaining requirements are explicit, rather than a successful repair action taken in isolation.

7 Conclusion

We used ORPMTF to distinguish what local results establish, which task states agents adopt, and what subsequent work still requires. In machine annotations of 5,433 trajectories across six environments, jointly judged events include omissions within permission and overreach despite enactment. Selected close readings explain the task requirements behind these distinctions: an operation error can leave the agent correctly treating work as unfinished, while a successful repair can restore one condition without establishing another. These analyses identify required changes left outstanding, adopted states lacking support, and the scope of a repair. Diagnosing task-state mismatch therefore extends beyond whether local work succeeded or was repaired to what the resulting state supports at its next use. It distinguishes conditions already established from those still unresolved, while retaining results that remain valid for the relevant object, version, and subsequent operation.

AI USE STATEMENT

We used generative AI tools, including OpenAI Codex and GPT models, Anthropic assistants, and Kimi, to assist with framework and study-design discussions, literature review, analysis code, interpretation, figure preparation, drafting, translation, and language editing. AI-generated components of the analyzed data include LLM-agent outputs recorded in benchmark trajectories and machine-generated annotations and adjudications of retained trajectories. We distinguish these model outputs from tool and environment observations and from human judgments; machine annotations are treated as weak labels, not independently human-validated ground truth. Section 4 and the appendices describe the annotation procedures and the scope of human review. The authors made the substantive research decisions and take responsibility for the manuscript, analyses, and conclusions.

REPRODUCIBILITY STATEMENT

Sections 3–4 define the framework and describe the data and annotation procedures. Appendix A documents the analysis units, denominators, exclusion rules, and aggregation procedures; Appendices B–F provide source descriptions, codebooks, and annotation instructions. Appendix I gives outcome linkage rules, grouped statistics, and weighting formulas for the task-success analysis. We plan to publicly release the curated research data and derived annotations, with documentation of data provenance and annotation definitions, subject to the licenses and redistribution terms of the original sources.

ETHICS STATEMENT

We selected tasks from six existing benchmarks, ran agents on those tasks, and analyzed the resulting trajectories and annotations. We did not collect personal data directly from individuals. We credit the original datasets and their creators. Our planned data release will preserve source attribution and follow the applicable licenses and redistribution terms. Where redistribution is restricted, we will provide source references and only derived annotations that the source licenses permit us to share.

REFERENCES

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. AgentRx: Diagnosing AI agent failures from execution trajectories, 2026. URL https://arxiv.org/abs/2602.02475v2. Version 2.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982v1. Version 1.

Radouane Bouchekir, Damir Safin, and Tomas Bueno Momcilovic. From natural language policies to executable obligations: A verification harness for dependable in-car LLM agents, 2026. URL https://arxiv.org/abs/2608.23282v1. Version 1.

Hongliu Cao, Ilias Driouich, and Eoin Thomas. Beyond task completion: Revealing corrupt success in LLM agents through procedure-aware evaluation, 2026. URL https://arxiv.org/abs/2603.03116v1. Version 1.

Ying Chen, Lihuang Fang, Rui Jiang, Mingxu Wang, Zhifeng Gu, Lei Yi, and Jie Chen. Done, but not sure: Disentangling world completion from self-termination in embodied agents, 2026. URL https://arxiv.org/abs/2605.08747v4. Version 4.

DeepSeek-AI et al. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348. Technical report covering DeepSeek-V4-Pro and DeepSeek-V4-Flash.

Shuhan Guo, Kun Zhang, Haifei Liu, Xingyu Gao, Yongqi Zhang, Yaqing Wang, and Quanming Yao. ContextFlow: Hierarchical task-state alignment for long-horizon embodied agents, 2026. URL https://arxiv.org/abs/2605.19314v1. Version 1.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks. GitHub repository, 2026. URL https://github.com/datacurve-ai/deep-swe. Accessed 2026-09-08.

Kabir Khandpur, Kilian Lieret, Carlos E. Jimenez, Ofir Press, and John Yang. SWE-bench Multilingual. SWE-bench project website, 2025. URL https://www.swebench.com/multilingual.html. Accessed 2026-09-08.

Dayoon Ko, Jihyuk Kim, Sohyeon Kim, Haeju Park, Dahyun Lee, Gunhee Kim, Moontae Lee, and Kyungjae Lee. When is enough not enough? illusory completion in search agents, 2026. URL https://arxiv.org/abs/2602.07549v1. Version 1.

Chenxin Li, Zhengyang Tang, Huan Lin, Yunlong Lin, Shijue Huang, Shengyuan Liu, Bowen Ye, Rang Li, Lei Li, Benyou Wang, et al. Claw-Eval-Live: A live agent benchmark for evolving real-world workflows, 2026. URL https://arxiv.org/abs/2604.28139v2. Version 2.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

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. In *Advances in Neural Information Processing Systems 37*, volume 37, pp. 74325–74362. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.52202/079017-2365. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/877b40688e330a0e2a3fc24084208dfa-Abstract-Datasets_and_Benchmarks_Track.html.

Mike A. Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, H. Raj, I. Bercovich, Lin Shi, J. Shin, Thomas Walshe, E. K. Buchanan, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868v1. Version 1.

MiniMax. MiniMax M3: Frontier coding, 1M context, native multimodality—all in one model, June 2026. URL https://www.minimax.io/blog/minimax-m3. Official release announcement, June 1. Accessed September 24, 2026.

OpenAI. Introducing GPT-5.4, March 2026a. URL https://openai.com/index/introducing-gpt-5-4/. Official release announcement, March 5. Accessed September 24, 2026.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition, July 2026b. URL https://openai.com/index/gpt-5-6/. Official release announcement for Sol, Terra, and Luna, July 9. Accessed September 24, 2026.

Yunjia Qi, Zehua Yin, Xintong Shi, Hao Peng, Songyuanyi Lu, Yixian Liu, Richeng Xuan, Yuhong Liu, Zhichao Hu, Xiaozhi Wang, et al. TRAJDEBUG: Tracing error lifecycle to identify critical failures in long-horizon agent trajectories, 2026. URL https://arxiv.org/abs/2608.06346v1. Version 1.

Jesus Salas. Correct is not governed: Provenance integrity in agentic workflows, 2026. URL https://arxiv.org/abs/2608.12761v1. Version 1.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Jiaming Wang, Ziteng Feng, Jiang-Tao Wu, Ruihao Li, Qianqian Xie, Yuxiang Ren, He Zhu, Xueming Han, Fan-Yu Meng, Jun-Lan Feng, et al. Where do deep-research agents go wrong? span-level error localization in agent trajectories, 2026. URL https://arxiv.org/abs/2606.02060v2. Version 2.

Xiaomi MiMo. MiMo-V2.5, 2026. URL https://huggingface.co/XiaomiMiMo/MiMo-V2.5. Official model card. Accessed September 24, 2026.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents, 2025. URL https://arxiv.org/abs/2504.21798v2. Version 2.

Zuyuan Zhang, Hanqing Yang, Carlee Joe-Wong, and Tian Lan. Auditing emergent LLM-agent collaboration through cooperation-obligation coupling, 2026. URL https://arxiv.org/abs/2607.27429v1. Version 1.

Xiangxin Zhao, Han Li, Shuaiting Li, Tianyi Zhao, Earl T. Barr, Federica Sarro, and He Ye. Failure as a process: An anatomy of CLI coding agent trajectories, 2026. URL https://arxiv.org/abs/2607.09510v1. Version 1.

A Detailed Distributional and Process Analyses

This appendix collects the corpus inventory, distributional analyses, and developmental case records supporting Section 5. Quantitative summaries describe machine annotations.

A.1 Corpus Inventory

The frozen six-environment snapshot (Table 3) covers 1,062 distinct known tasks; 117 trajectories with unresolved task identity remain outside task-level aggregations. Of 13,241 update events, 6,162 have applicable judgments on both assessment axes.

Table 3: Frozen corpus inventory. Core events and update events are distinct annotation units and need not correspond one-to-one.

Environment	Trajectories	Core events	Update events
AppWorld	1,051	1,272	2,316
CLAW	605	1,168	1,009
DeepSWE	564	1,863	1,987
SWE-multilingual	1,778	3,890	3,681
TerminalBench	460	636	955
tau2bench	975	1,217	3,293
Total	5,433	10,046	13,241

The 35 working items used in the following machine-annotation summaries are listed here for reference. Their K-codes are grouped under P01–P10 for display; this derived catalog is separate from the 60 operational children in Appendix D.

Table 4: Short labels for the 35-item derived phenomenon catalog.

Code	Working item	Code	Working item
K01-01	Task scope narrowed	K06-04	Dependent call before prerequisite
K01-02	Acceptance criteria lowered	K07-01	Required component missing
K01-03	Substitute action for task	K07-02	Internal state-record mismatch
K01-04	Conditional authority breached	K07-04	Preserved state damaged by change
K01-05	Required repair left at diagnosis	K07-05	Transfer breaks object semantics
K02-01	Wrong entity, attribute or record	K07-06	Allowed boundary input mishandled
K02-02	Wrong time or version	K07-07	Outputs violate a required relation
K02-03	Extra objects changed	K07-08	Edit reported successful; target unchanged
K03-01	Distinct states conflated	K07-09	Redundant state change after success
K03-02	Missing observation treated as positive	K08-01	Verification misses requirements
K03-03	Unread data treated as normal	K08-02	Verification design cannot test claim
K03-04	Structured observation misread	K09-01	Unsupported fact or advice
K04-01	Feedback not incorporated	K09-02	Uncertainty given precise assurance
K05-01	Oscillation despite actionable path	K09-03	Partial evidence overgeneralized
K05-02	Uninformative checks or retries	K09-04	Details and aggregate conflict
K06-01	Tool-contract format violation	K09-05	Claim/promise-action mismatch
K06-02	Concurrent calls despite serial rule	K10-01	Partial work closed as complete
K06-03	Reply and tool call improperly mixed		

A.2 Joint Composition of the Two Assessment Axes

Table 5: Joint composition of machine-annotated update events. J/E gives events with judgments on both axes over all update events; the six cells are percentages of J, rounded to one decimal. Partial denotes partial enactment of M.

		T within P			T beyond P		
Environment	J/E	Enacted	Partial	Not enacted	Enacted	Partial	Not enacted
AppWorld	1618/2316	85.4	1.2	2.2	0.8	0.9	9.5
CLAW	694/1009	39.9	0.9	6.5	7.3	11.1	34.3
DeepSWE	1007/1987	67.8	1.6	17.9	0.2	0.4	12.1
SWE-multilingual	565/3681	75.2	4.2	11.9	0.4	0.4	8.0
TerminalBench	419/955	73.3	3.3	8.4	0.2	1.0	13.8
tau2bench	1859/3293	88.9	1.0	3.3	1.5	0.5	4.9

Table 6: Recorded judgment coverage. Categories are mutually exclusive under the ordering described in the text; they summarize saved fields without relabeling events.

Source	E	J	M-empty/NA	Conflict	Unobs.	Other
AppWorld	2316	1618	623	1	12	62
CLAW	1009	694	307	1	4	3
DeepSWE	1987	1007	906	1	72	1
SWE-multilingual	3681	565	2791	1	304	20
TerminalBench	955	419	515	0	7	14
tau2bench	3293	1859	1358	0	11	65
Total	13241	6162	6500	4	410	165

Table 6 distinguishes joint judgment from other recorded conditions. We first identify J, then explicit empty M with inapplicable fulfillment, then remaining conflicted judgments, remaining unobservable judgments, and other records. This ordering prevents double counting; it does not reconcile conflicting fields. The M-empty category includes 46 records with unobservable permission and two with conflicted permission. Other records include missing, noncanonical, or inconsistent fields, including seven inapplicable-fulfillment entries without explicit empty M. Thus exclusion from J is not uniformly an observation gap, and no excluded category is treated as evidence of alignment.

A.3 Within-Environment Model-Pair Comparability

The 104 within-environment pairs each share at least ten tasks. Table 7 summarizes their task overlap and support-composition distances. Matching tasks alone does not control execution settings or resolve model-version uncertainty, so these comparisons do not isolate model effects.

Table 7: Within-environment model pairs. Each pair compares common tasks with equal task weights; G is the number of common tasks and TV is the total-variation distance between support-attribution compositions.

Environment	**Pairs**	**G range**	**TV range**
AppWorld	28	48–148	0.178–0.897
CLAW	6	57–86	0.181–0.251
DeepSWE	15	79–90	0.125–0.759
SWE-multilingual	28	72–250	0.085–0.875
TerminalBench	21	10–76	0.193–0.950
tau2bench	6	120–204	0.184–0.558

A.4 Model Profiles on Common Tasks

Figures 4–6 summarize local-phenomenon profiles, two-axis judgments, and cross-environment differences. All comparisons use nominal model groups and at least ten common tasks within each

environment, with equal weights across tasks and across runs within a task. Each model pair has its own common-task set; these sets differ across environments.

Local-phenomenon profiles. Figure 4 summarizes the direction of task-weighted support-attribution differences across same-environment references for each of the 37 model–environment groups and 35 items in Table 4. A cell records higher with no lower comparisons, lower with no higher comparisons, both directions, or equality throughout. The first two categories allow ties. These direction summaries retain reference dependence without averaging differences across distinct common-task sets.

Coverage and joint mismatch. Figure 5 retains all 104 within-environment pairs. Coverage h is the task-weighted share of trajectories with at least one event judged on both axes ($J > 0$). For each trajectory, we compute the fraction of its J events labeled both T beyond P and M not enacted, assigning zero when $J = 0$. Averaging these fractions with the same task weights and dividing by h gives the conditional share θ. Thus h measures judgment availability, while θ describes the joint-mismatch cell among judgeable trajectories; partial enactment is a separate cell. These quantities differ from pooled event coverage J/E and pooled cell percentages in Table 1.

Cross-environment directions. Figure 6 compares source-specific support-attribution differences for 28 fixed model pairs. Its 5,390 entries cover 35 items across 154 model-pair–environment-pair combinations. Differences within ± 0.01 form the near-zero band. If both differences lie outside it, we distinguish opposite signs from same signs; same-sign cases split at a smaller-to-larger magnitude ratio of 0.5 (at least 0.5 versus below 0.5). Two further categories have one or both differences within the band. Counts and row fractions describe these comparisons, not independent trials or significance tests. Changing environments also changes tasks, so the profiles do not isolate model or environment effects.

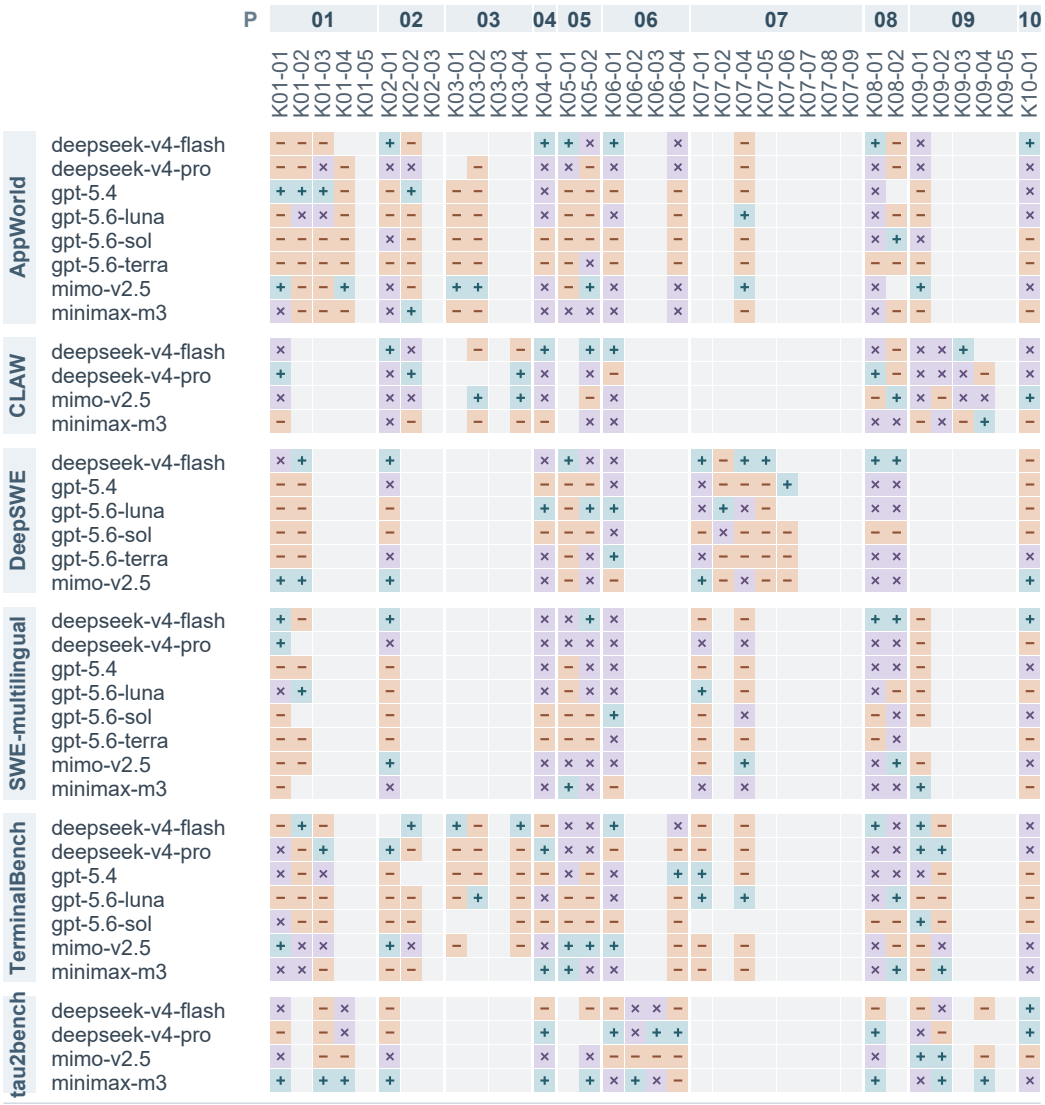


Figure 4: Local-phenomenon direction profiles across same-environment references. Rows retain full model names; columns use the 35-item catalog. Symbols distinguish higher-only, lower-only, mixed, and equal configurations. Each reference comparison uses its own common tasks.

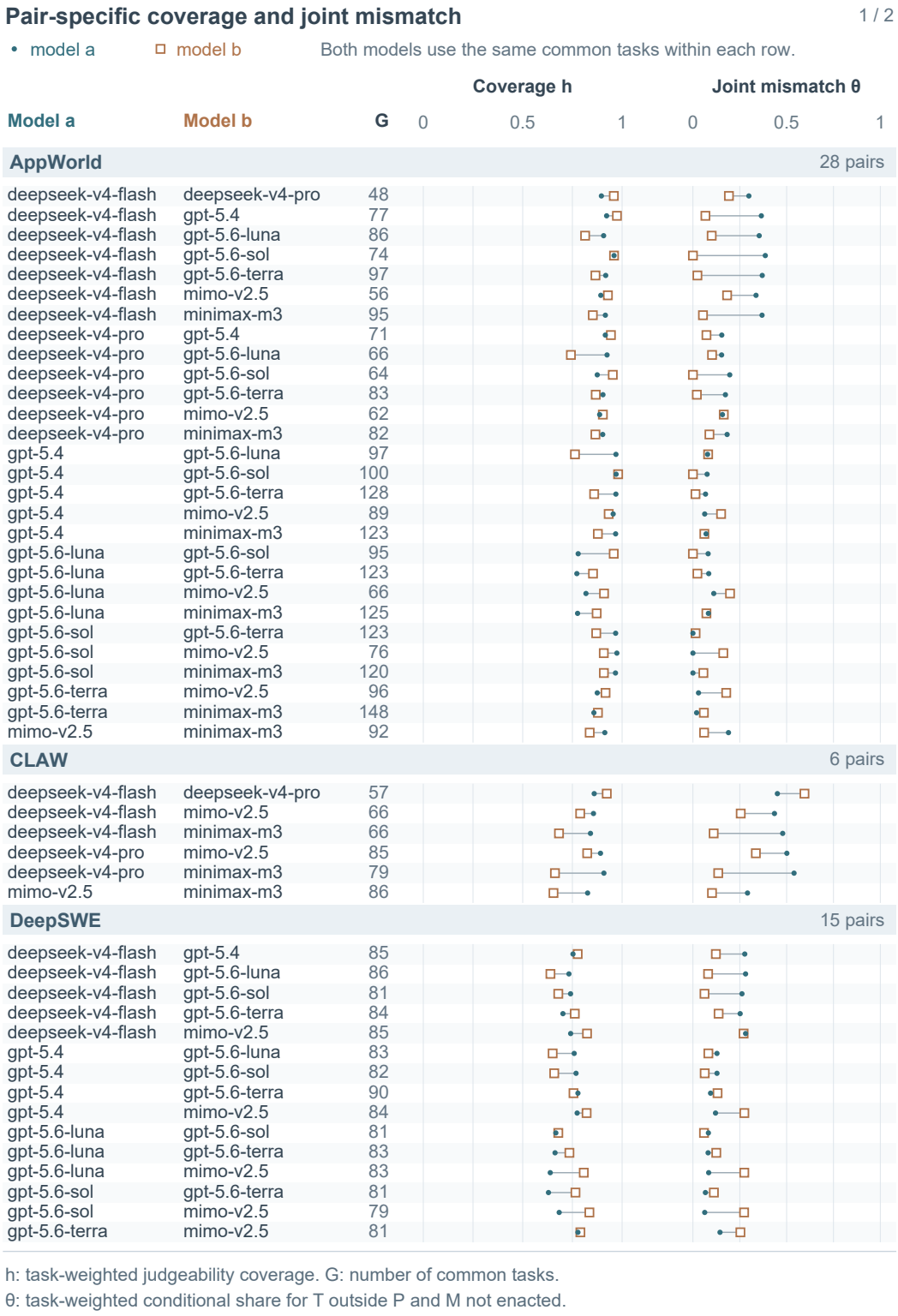
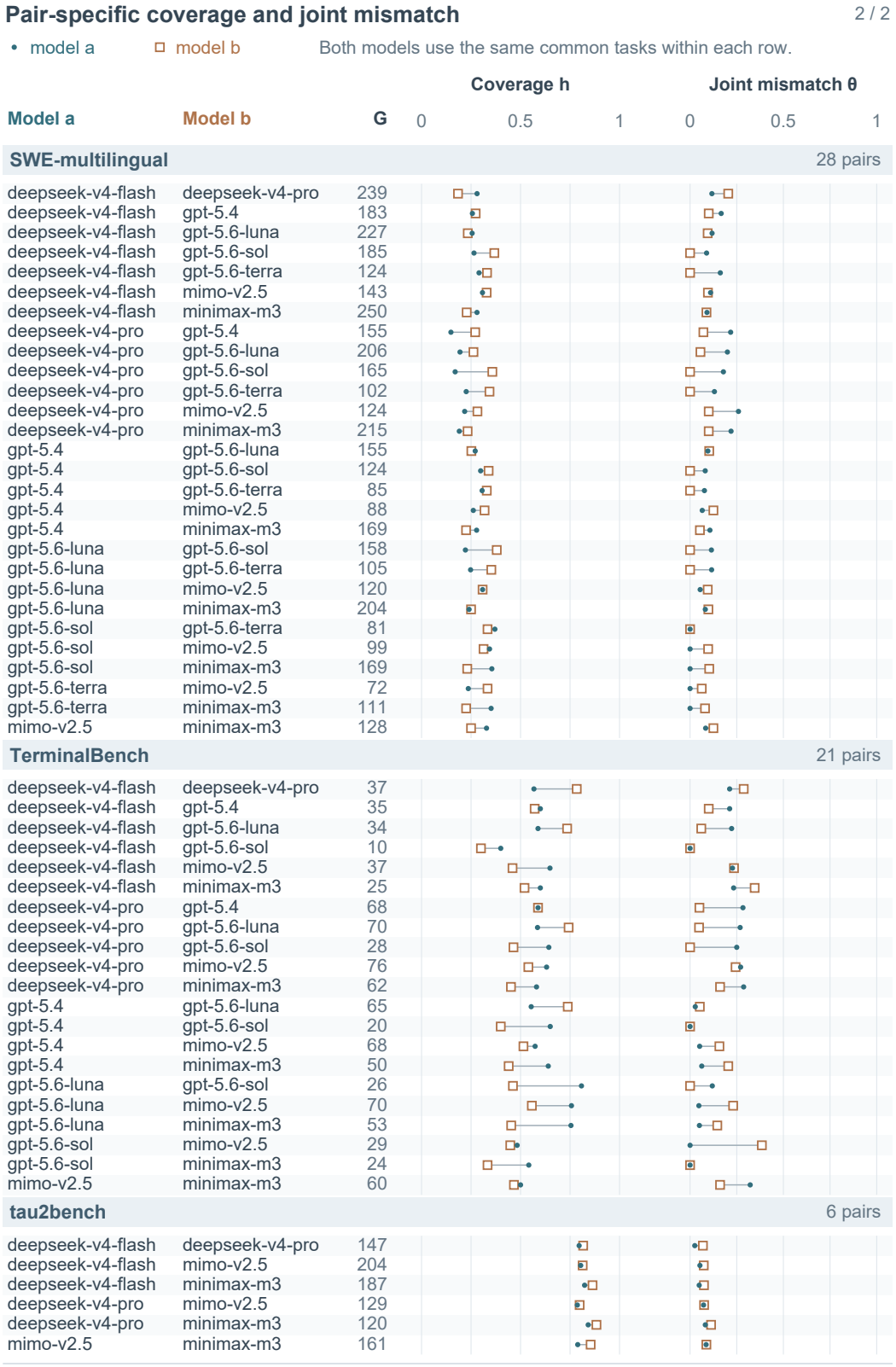


Figure 5: Common-task coverage h and conditional joint mismatch θ. Plate 1 of 2: AppWorld, CLAW, and DeepSWE. Filled circles and open squares identify the two models; segments connect their values, not confidence intervals. G gives the number of common tasks.



Model a	Model b	G
SWE-multilingual		28 pairs
deepseek-v4-flash	deepseek-v4-pro	239
deepseek-v4-flash	gpt-5.4	183
deepseek-v4-flash	gpt-5.6-luna	227
deepseek-v4-flash	gpt-5.6-sol	185
deepseek-v4-flash	gpt-5.6-terra	124
deepseek-v4-flash	mimo-v2.5	143
deepseek-v4-flash	minimax-m3	250
deepseek-v4-pro	gpt-5.4	155
deepseek-v4-pro	gpt-5.6-luna	206
deepseek-v4-pro	gpt-5.6-sol	165
deepseek-v4-pro	gpt-5.6-terra	102
deepseek-v4-pro	mimo-v2.5	124
deepseek-v4-pro	minimax-m3	215
gpt-5.4	gpt-5.6-luna	155
gpt-5.4	gpt-5.6-sol	124
gpt-5.4	gpt-5.6-terra	85
gpt-5.4	mimo-v2.5	88
gpt-5.4	minimax-m3	169
gpt-5.6-luna	gpt-5.6-sol	158
gpt-5.6-luna	gpt-5.6-terra	105
gpt-5.6-luna	mimo-v2.5	120
gpt-5.6-luna	minimax-m3	204
gpt-5.6-sol	gpt-5.6-terra	81
gpt-5.6-sol	mimo-v2.5	99
gpt-5.6-sol	minimax-m3	169
gpt-5.6-terra	mimo-v2.5	72
gpt-5.6-terra	minimax-m3	111
mimo-v2.5	minimax-m3	128
TerminalBench		21 pairs
deepseek-v4-flash	deepseek-v4-pro	37
deepseek-v4-flash	gpt-5.4	35
deepseek-v4-flash	gpt-5.6-luna	34
deepseek-v4-flash	gpt-5.6-sol	10
deepseek-v4-flash	mimo-v2.5	37
deepseek-v4-flash	minimax-m3	25
deepseek-v4-pro	gpt-5.4	68
deepseek-v4-pro	gpt-5.6-luna	70
deepseek-v4-pro	gpt-5.6-sol	28
deepseek-v4-pro	mimo-v2.5	76
deepseek-v4-pro	minimax-m3	62
gpt-5.4	gpt-5.6-luna	65
gpt-5.4	gpt-5.6-sol	20
gpt-5.4	mimo-v2.5	68
gpt-5.4	minimax-m3	50
gpt-5.6-luna	gpt-5.6-sol	26
gpt-5.6-luna	mimo-v2.5	70
gpt-5.6-luna	minimax-m3	53
gpt-5.6-sol	mimo-v2.5	29
gpt-5.6-sol	minimax-m3	24
mimo-v2.5	minimax-m3	60
tau2bench		6 pairs
deepseek-v4-flash	deepseek-v4-pro	147
deepseek-v4-flash	mimo-v2.5	204
deepseek-v4-flash	minimax-m3	187
deepseek-v4-pro	mimo-v2.5	129
deepseek-v4-pro	minimax-m3	120
mimo-v2.5	minimax-m3	161

h: task-weighted judgeability coverage. G: number of common tasks.
θ: task-weighted conditional share for T outside P and M not enacted.

Figure 5: Common-task coverage and joint mismatch (plate 2 of 2).

How model-pair differences vary across environments

Counts across 35 local phenomena × eligible environment pairs.
Each row compares model a − model b; bars show the fraction of N.

Model a	Model b	E	N	Same sign similar size	Same sign unequal size	Opposite signs	Only one outside ±0.01	Both within ±0.01
deepseek-v4-flash	deepseek-v4-pro	5	350	8	12	22	92	216
deepseek-v4-flash	gpt-5.4	4	210	12	20	8	52	118
deepseek-v4-flash	gpt-5.6-luna	4	210	14	15	11	55	115
deepseek-v4-flash	gpt-5.6-sol	4	210	13	14	7	43	133
deepseek-v4-flash	gpt-5.6-terra	3	105	5	9	4	20	67
deepseek-v4-flash	mimo-v2.5	6	525	18	15	24	151	317
deepseek-v4-flash	minimax-m3	5	350	11	19	29	78	213
deepseek-v4-pro	gpt-5.4	3	105	5	5	5	20	70
deepseek-v4-pro	gpt-5.6-luna	3	105	6	7	4	28	60
deepseek-v4-pro	gpt-5.6-sol	3	105	5	4	3	16	77
deepseek-v4-pro	gpt-5.6-terra	2	35	3	0	2	7	23
deepseek-v4-pro	mimo-v2.5	5	350	12	12	27	86	213
deepseek-v4-pro	minimax-m3	5	350	7	17	14	84	228
gpt-5.4	gpt-5.6-luna	4	210	6	10	10	50	134
gpt-5.4	gpt-5.6-sol	4	210	5	0	0	44	161
gpt-5.4	gpt-5.6-terra	3	105	4	0	3	18	80
gpt-5.4	mimo-v2.5	4	210	12	19	10	56	113
gpt-5.4	minimax-m3	3	105	4	4	6	22	69
gpt-5.6-luna	gpt-5.6-sol	4	210	6	6	8	32	158
gpt-5.6-luna	gpt-5.6-terra	3	105	2	2	1	20	80
gpt-5.6-luna	mimo-v2.5	4	210	12	12	11	59	116
gpt-5.6-luna	minimax-m3	3	105	4	5	4	26	66
gpt-5.6-sol	gpt-5.6-terra	3	105	0	0	4	18	83
gpt-5.6-sol	mimo-v2.5	4	210	18	14	5	49	124
gpt-5.6-sol	minimax-m3	3	105	5	1	1	20	78
gpt-5.6-terra	mimo-v2.5	3	105	5	12	4	22	62
gpt-5.6-terra	minimax-m3	2	35	1	1	1	6	26
mimo-v2.5	minimax-m3	5	350	13	12	21	100	204

E: eligible environments; N = 35 × E(E − 1)/2. Each source requires G ≥ 10.
First three columns: both differences exceed 0.01 in absolute value.
Similar size: smaller magnitude ≥ half the larger; unequal size: less than half.
Differences use task-weighted support intensity, not task-success rates.

Figure 6: Cross-environment directions of model-pair support-attribution differences. Counts cover 35 items over all eligible environment pairs; bars show fractions of each row total N. E is the number of eligible environments, with at least ten common tasks per environment. The magnitude threshold is descriptive.

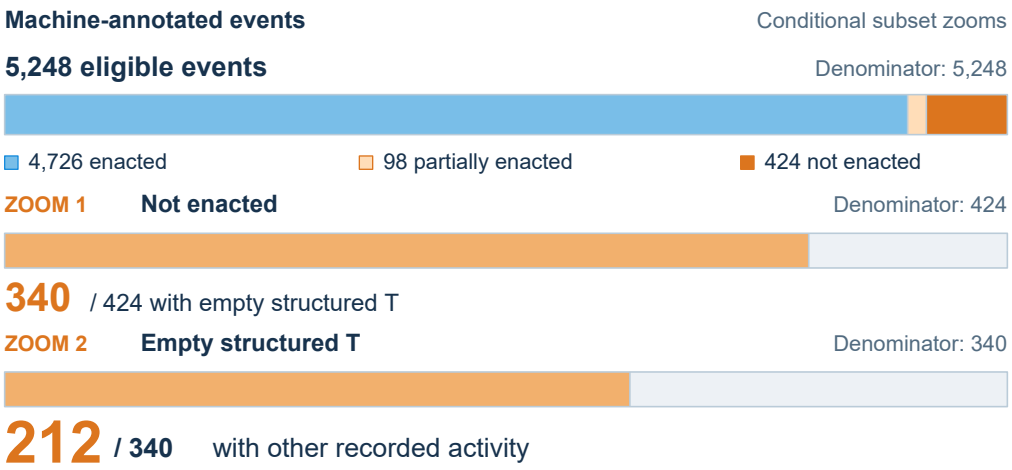


Figure 7: **Recorded activity within nested annotation subsets.** The initial subset contains machine-labeled events within P with explicit M and resolved enactment judgments; 50 otherwise eligible events with unresolved judgments are excluded. Each successive bar zooms in on the highlighted subset above, using its own displayed denominator.

A.5 Recorded Activity and Mandatory Updates

Figure 7 shows recorded activity in a subset of machine-labeled events with M not enacted and empty structured T. An empty T record does not establish inactivity, and activity alone does not establish enactment. The relevant question is whether the activity performs the mandatory change due at that point; initiating a re-check satisfies M only when initiation itself is required. Table 9, under Activity and Existing diagnosis, documents the illustrative case evidence and its limits.

A.6 Close Readings of Operation-Error Windows

Table 8 records twelve purposefully selected developmental windows; Section 6.2 uses Q2W11/Q2W12 to contrast responses to operation errors. Each label pair reports permission alignment followed by mandatory-update enactment; NA means not applicable. Original labels guided retrieval. The readings excluded termination payloads and do not establish benchmark success or corpus proportions.

Q2W05 retains broad and narrow readings of the mandatory change. For Q2W07/Q2W08, the state update and due mandatory change remain unresolved despite recorded rejection and correction; Q2W07's displayed tool name may also reflect redaction. Q2W12's PARTIAL permission label is historical, not a category in the current permission axis. Later filtering excluded the specific Q2W08/Q2W12 retrieval hits while retaining their trajectories through other rules; it supplied no new readings or changes to corpus labels.

Table 8: Original and developmental reading labels for twelve selected windows.

ID	Environment	Original labels	Reading labels	Operation and reading note
Q2W01	AppWorld	YES/YES	YES/NA	Prose in code caused a SyntaxError.
Q2W02	AppWorld	YES/YES	YES/YES	Plan–execution gap and NameError.
Q2W03	DeepSWE	YES/NA	YES/YES	Six patch-format rejections despite a zero return code.
Q2W04	AppWorld	NO/NO	YES/YES	Repeated empty-code emissions.

ID	Environment	Original labels	Reading labels	Operation and reading note
Q2W05	DeepSWE	YES/NO	YES/YES (broad); YES/NA (narrow)	Patch-marker contamination and inflated hunk counts; reading depends on mandatory-update scope.
Q2W06	AppWorld	YES/NO	YES/YES	Valid batch writes; corrected an empty-code slip.
Q2W07	tau2bench	YES/NO	NO/NO	Rejected tool calls, including two in one turn; tool identity and state-level interpretation unresolved.
Q2W08	AppWorld	NO/NO	NO/NO	Parameter transferred across APIs; later correction, state-level interpretation unresolved.
Q2W09	SWE-multilingual	YES/NA	YES/YES	Hunk-count mismatch; switched to `git apply`.
Q2W10	DeepSWE	YES/NA	YES/YES	Reproduction-file creation failed; repeated retries, then fallback.
Q2W11	DeepSWE	YES/YES	YES/YES	Logged the rejected integration, checked the four target files, and split the patch.
Q2W12	AppWorld	YES/YES	PARTIAL/NO	Relied on an unexecuted re-login; later made unsupported payment and completion claims.

A.7 Diagnostic Distinctions and Evidence Limits

Table 9 gives the evidence and limits behind the main-text distinctions in Table 2. The developmental comparisons used coarse activity records, earlier axis labels, and previously accepted case analyses. They combine state-oriented representation, close reading, and correction of earlier labels; they do not isolate a representation effect or compare implemented literature baselines.

Across 17 comparisons with coarse records, the historical dispositions were 13 supported delta, three no delta, and one insufficient evidence. Three additional rearrangements of accepted analyses yielded no delta. These entries mix stages and units, so the counts are not an improvement rate. The insufficient-evidence entry concerns the initial Q2 selection records; later readings do not replace that earlier disposition.

Table 9: Case evidence supporting diagnostic distinctions and their limits. Rows group related observations, not independent comparisons.

Theme	Case evidence	Diagnostic distinction and limit
Activity	Q1D10 records preparation without an observed artifact write; Q1C04 records artifact production and limited checks. Q1D05 later completes a deferred item; Q1D09 ends with incomplete observation.	Activity, delivery, and requirement verification differ. Q1D10/Q1D09 do not establish omission; Q1C04 does not establish full correctness. Later completion does not justify earlier deferral.
Existing diagnosis	Q1C11's earlier analysis records testing, defect exposure, and repair. Accepted X19/X60 analyses already identify the remaining session requirement and version-recovery problem.	Reorganizing these analyses yields no new diagnosis. These no-delta results concern accepted analyses, not their coarser records. Repair alone does not establish full requirement coverage.

Theme	Case evidence	Diagnostic distinction and limit
Operation errors	Q2W11 treats the rejected integration as unresolved and attempts correction. Q2W12 relies on an unexecuted re-login and later makes unsupported payment and completion claims.	Rejection and unsupported state adoption call for different judgments. Q2W07/W08 remain unresolved at the state level despite subsequent correction (Appendix A.6).
Mandatory-update scope	Q2W01/W03/W09/W10 change applicability judgments; Q2W05 retains broad and narrow readings (Table 8).	Applicability depends on the requirement and due time. Not applicable differs from both enacted and not enacted; these readings supply no universal timing rule.
Independent prerequisite	X19 restores phone access, but the missing Venmo session surfaces on first use as a NameError, not an API refusal.	Venmo access was required before that use, not immediately after phone repair. Later segments are unreadable, leaving benchmark success unknown.
Version recovery	X60's copy-back succeeds, but read-back shows 22 bytes rather than the earlier 15. The agent then reconstructs prior content.	Copy success does not establish target-version recovery. Reconstruction relies on recorded observations and write echoes, not an independent original copy; final evaluator success remains unknown.

The separate Q3 review covered 60 cases and 78 episodes, of which 54 cases and 64 episodes were judgeable. It supported no instance of a verified repair followed by stale use of the same object. This remains an evidence limit within the reviewed set, not evidence of absence elsewhere; episodes are not machine-annotated update events.

B Data Sources and Environment Selection

B.1 Data sources and model coverage

The profiles below identify the six sources and their evidence channels; observability depends on each retained trajectory.

AppWorld. AppWorld tasks span interconnected applications and APIs, with state-based tests that check task completion and unintended changes while allowing multiple valid solutions (Trivedi et al., 2024).

CLAW. CLAW denotes Claw-Eval-Live, whose service and workspace workflows provide execution traces, audit logs, service state, and post-run artifacts, with deterministic and semantic grading (Li et al., 2026).

τ^2-bench. This source, recorded as tau2bench, combines conversation and tool use; in its dual-control telecom setting, both the agent and simulated user can act on shared state (Barres et al., 2025).

Terminal-Bench. Recorded as TerminalBench, this source covers terminal tasks with command output, file changes, and completion tests; the published benchmark description also specifies human-written reference solutions (Merrill et al., 2026).

DeepSWE. DeepSWE denotes the Datacurve software-engineering benchmark: tasks drawn from open-source repositories, with isolated execution environments and programmatic verifiers (Huang et al., 2026).

SWE-bench Multilingual. Recorded as SWE-multilingual, it pairs issues with repository snapshots across programming languages, using fail-to-pass tests for fixes and pass-to-pass tests for previously passing behavior (Khandpur et al., 2025).

Table 10 lists recorded model labels by source. Two τ^2-bench trajectories carry GPT-5.6 Luna labels supported only by paths; they remain in this roster but are excluded from the model-attributable subset in Table 1.

Table 10: Recorded trajectory-generating model labels by source.

Data source	Models
AppWorld	GPT-5.4; GPT-5.6 Luna, Sol, and Terra; DeepSeek V4 Flash and Pro; MiMo V2.5; MiniMax M3
CLAW	DeepSeek V4 Flash and Pro; MiMo V2.5; MiniMax M3
DeepSWE	GPT-5.4; GPT-5.6 Luna, Sol, and Terra; DeepSeek V4 Flash; MiMo V2.5
SWE-bench Multilingual	GPT-5.4; GPT-5.6 Luna, Sol, and Terra; DeepSeek V4 Flash and Pro; MiMo V2.5; MiniMax M3
Terminal-Bench	GPT-5.4; GPT-5.6 Luna and Sol; DeepSeek V4 Flash and Pro; MiMo V2.5; MiniMax M3
τ^2-bench	GPT-5.6 Luna; DeepSeek V4 Flash and Pro; MiMo V2.5; MiniMax M3

B.2 Source provenance and evidence scope

We link annotations to their source through dataset and native task or trajectory identifiers, run or attempt lineage, source location, and event checkpoint. The records distinguish generating models from annotating models and preserve the selected annotation version (Appendix F). Unknown historical execution settings remain unspecified. Task identity and trajectory identity have separate inclusion requirements (Section B.3). Native rewards and completion flags retain their source-specific meanings; hidden grading information cannot substitute for the task requirements, available evidence, and observed response used to judge P/M/T.

B.3 Data roles, analysis units, and inclusion

The 600-record human review informed framework development (Appendix C); machine annotations support retrieval and descriptive summaries (Appendices F and A); independent validation remains planned (Appendices G–H).

We distinguish trajectories, annotation records, events, lineages, revisions, and comparison pairs when counting observations. Events from the same trajectory are not independent replications. Revised files and reused cases add no observations; comparison pairs are analytical contrasts, not new records. Inclusion requires a resolved trajectory-source link and locatable evidence for the intended judgment; task-level aggregation also requires a resolved task identity. An intermediate judgment may be retained when later fate is unresolved. The manual distinguishes no-event, empty-update, and unobservable judgments (Appendix E).

C Development of the Analytical Framework

C.1 Human review during framework development

We developed the framework through a review of 600 trajectory records across six sources, conducted by one author with prior machine annotations and source information visible. AI assisted evidence retrieval, summaries, and revision proposals; the author checked interpretations against the evidence and made the decisions. The final review decisions comprise 561 confirmations, 18 minor repairs, and 21 disagreements across the 600 reviewed records. These decisions informed framework development and concern local annotations rather than human ORPMTF judgments.

C.2 Distinctions from developmental review

Table 11 shows four distinctions retained during review and the design choices they informed.

Table 11: Developmental distinctions and their implications for framework design.

Review distinction (record identifiers)	**Subsequent design implication**
An unsuccessful attempt can inform a later attempt; repetition alone does not establish avoidable non-progress (P18-DEF-001–006; P18-HS-0022).	Separate local error from later handling and recovery.
Usable feedback was not sufficiently integrated into later reasoning, action, or reporting (P18-DEF-007; P18-HS-0021).	Compare required changes with actual uptake.
The submitted payment-method identifier was correct, but confirmation named another method (P18-HS-0118).	Separate action-established and reported state.
The case lacked result inspection or state readback (P18-HS-0420); the definition excludes calls whose authoritative returns establish the required effect (P18-DEF-011).	Derive verification needs from task rules and available evidence.

We combined these judgments with existing vocabulary and model-proposed groupings into ten working parent classes (Appendix D). The classes describe local breakdowns; process outcomes, external blockers, and evidence limitations remain separate. A trajectory may support multiple classes or none.

C.3 From local descriptions to event analysis

Local classes describe breakdowns but do not determine which task-state updates the evidence warrants. ORPMTF therefore requires separate event judgments (Appendix E); the 600-record review informed its design but did not produce human ORPMTF labels. The representation distinguishes permission from necessity and tests actual updates against each independently, retaining compatibility conditions on permitted bundles. An absent update counts as omission only when a required change, an opportunity, and an adequate observation window are established.

The representation also separates a result's validity at the event from its later use: subsequent misuse does not revise earlier judgments. Planned independent validation locks those judgments before revealing later behavior and fate, and assesses reconstruction at supplied checkpoints rather than exhaustive event discovery (Appendix G).

D Local Failure Classes and Submechanism Codebook

D.1 Scope and evidence requirements

The ten working parent classes organize local breakdowns; the registry's sixty child identifiers specify the controlled annotation vocabulary. A separate, machine-derived 35-item catalog supports phenomenon retrieval and descriptive summaries (Table 4). It shares the P01–P10 headings but is not a subset or remapping of the sixty children. Both vocabularies provide descriptive context; ORPMTF assesses state updates independently. Canonical registry identifiers govern where display names differ. Assignment requires a visible class-relevant condition, an agent-attributable deviation meeting the class criterion, a direct event and same-trajectory span resolving to the relevant native object or position, and material contribution to the local problem. Terminal failure is unnecessary. Additional classes need distinct evidential propositions, even if they share a passage. Adequately reviewed material with no qualifying class supports no-positive; missing or conflicting evidence remains unresolved. Unmatched descriptions remain extension candidates outside this registry.

D.2 Parent classes and controlled submechanisms

Each entry gives the identifying criterion; Table 12 consolidates comparisons between neighboring classes.

P01: Task contract alignment failure. P01 requires an available task requirement and an agent decision that replaces, omits, weakens, or violates it.

`proxy_goal_substitution`: The agent replaces the requested goal with an intermediate metric, an easier proxy, or a different completion condition. Selecting the wrong concrete target while retaining the goal is a P02 issue. `instruction_constraint_nonadherence`: The agent violates an explicit instruction, prohibition, output constraint, or procedural requirement. An interface-schema violation belongs to P06 when that is the direct defect. `requirement_or_deliverable_omission`: A required task element or deliverable is absent from the planned or delivered work. An attempted component with defective implementation calls for P07 instead. `workset_scope_misbinding`: The agent forms the wrong set of objects or subtasks to process, excluding required work or including work outside the requested scope. Confusing two individual items within a correctly formed workset is P02. `acceptance_criteria_weakening`: The agent substitutes a weaker acceptance condition without supporting authorization. Keeping the original criterion but failing to check it is a P08 question. `authorization_condition_violation`: The agent acts before obtaining a required approval, confirmation, or permission. An external permission denial alone does not establish this violation.

P02: Goal–context binding failure. P02 requires a recoverable intended referent and selection of a different referent, or action before a resolvable target ambiguity is settled. The canonical identifier is `P02_TARGET_CONTEXT_BINDING_FAILURE`; "goal–context binding failure" is its main-text name.

`entity_attribute_referent_misbinding`: An operation or claim concerns the wrong entity, attribute, field, or pronoun referent. Misinterpreting the value of the correct attribute is P03. `file_channel_session_misbinding`: The agent directs work to the wrong file, channel, session, conversation, transaction, or execution context. `identity_or_state_key_mismatch`: A native identifier, object key, state key, or identity chain points to the wrong item. A wrong locator introduced by an annotator is an annotation defect, not an agent failure. `evidence_time_slice_misbinding`: The agent uses evidence from the wrong version, checkpoint, time window, or pre-action/post-action state. `state_transition_target_mismatch`: A validly expressed mutation affects the wrong target. A defective mutation of the intended target is P07. `action_under_unresolved_target_ambiguity`: The trajectory provides enough information to resolve the target, but the agent acts before doing so. Clarifying genuine ambiguity or refraining from mutation is a boundary case.

P03: Observation–state interpretation failure. P03 requires an observation about the intended target and a visibly incorrect interpretation in reasoning, action, or reporting. A missing observation does not establish a misread.

`tool_output_misread`: The agent reads an actual tool return incorrectly. A claimed observation that did not occur belongs under P09. `state_value_miscalculation`: The agent calculates a quantity, aggregate, or derived state value incorrectly from available evidence. `state_misclassification`: The agent assigns the wrong category or status to an observed state. If no classification was made, this entry does not apply. `cross_source_conflict_mishandling`: The agent reconciles visible conflicting sources through unsupported precedence or synthesis. `state_synthesis_or_grounding_failure`: The agent combines observations into an incorrect account of the task state, despite referring to the intended object.

P04: Feedback integration and adaptation failure. P04 requires separately locatable actionable feedback and subsequent behavior that fails to incorporate it. A first error, vague failure message, or justified investigation of an unreliable checker is insufficient.

`user_correction_not_integrated`: A clear user correction is not reflected in subsequent behavior. `tool_failure_feedback_not_integrated`: A tool error or returned constraint identifies a needed change, but the agent repeats an incompatible plan or action. `validation_counterevidence_not_integrated`: A test or verification result contradicts the current hypothesis, yet the agent retains it without resolving the conflict. `failed_prerequisite_ignored`: A prerequisite visibly fails, but dependent work proceeds as though it held. Diagnosis of an uncertain prerequisite failure does not by itself meet this criterion. `stale_assumption_`

or_action_persistence_after_feedback: An assumption or action pattern persists after actionable counterevidence.

Changing course and encountering a different obstacle is not nonadaptation; repetition alone does not establish ignored feedback.

P05: Exploration and progression control failure. P05 requires avoidable non-progress despite sufficient information to avoid the work: the activity neither resolves a live uncertainty nor advances a relevant state. Trajectory length alone is insufficient.

recurrent_self_induced_repair_cycle: The agent repeatedly recreates and repairs the same self-caused problem. A single error followed by effective repair is excluded from this entry. action_repetition_without_progress: Substantively identical actions recur without new information or state progress. A justified retry after a transient infrastructure failure is excluded. deliberation_loop_without_execution: The agent repeatedly revisits an actionable decision without taking the necessary next action, despite having sufficient information. diagnostic_loop_without_implementation: Diagnosis continues after a supported repair is available, without implementing it. Each diagnostic step that narrows a live hypothesis remains legitimate. noninformative_oververification_churn: Repeated tests or checks add no material coverage or evidential support. Independent checks required by the task protocol are excluded. verified_state_rollback_or_postverification_overexploration: The agent undoes a verified state or continues unnecessary exploration after the relevant uncertainty is resolved. New evidence that invalidates the earlier verification can justify renewed work.

Two similar repetitions offer weak evidence; three or more noninformative repetitions, repeated recreation of a side effect, or rollback of verified work are stronger screening cues, never count-only rules. Recovery attempts that incorporate new information do not by themselves satisfy P05, even when several attempts are needed.

P06: Tool and interaction protocol failure. P06 requires an available interface contract and an emitted request that violates it; an unfavorable result alone is insufficient.

wrong_tool_or_endpoint_selection: The intended target is correct, but the selected tool or endpoint cannot perform the intended operation. api_argument_schema_mismatch: Argument names, types, nesting, or structure violate the interface schema. Schema-valid arguments referring to the wrong object are a P02 issue. action_payload_omission: The request lacks a required payload or field. Omission of a task-level deliverable is P01. serialization_or_invocation_format_violation: Escaping, encoding, serialization, command syntax, or invocation format violates the protocol. An accepted equivalent representation is not a violation. tool_call_ordering_violation: The agent makes calls in an order that violates interface prerequisites. call_response_pairing_or_concurrency_violation: The agent violates response identity, call–response pairing, concurrency, or replay requirements.

Tool unavailability is an external boundary, not itself an interface violation.

P07: Action realization and result integrity failure. P07 requires an observable semantic defect in an action's result or an artifact; missing verification alone is insufficient.

artifact_semantic_defect: Code, configuration, a data transformation, or an external-state artifact violates its intended semantics. silent_partial_implementation: An apparently complete implementation provides only a material subset of the required behavior. A requirement omitted before implementation is more directly addressed by P01. action_result_corruption: The action produces a corrupt, inconsistent, or semantically invalid result. no_op_or_incomplete_mutation: An attempted state change has no effect or changes only part of the intended target state. implementation_integration_failure: A locally plausible implementation does not integrate correctly into the larger system or workflow. A failure to test integration, without an observed integration defect, supports only a verification inquiry. side_effect_control_or_duplicate_irreversible_action: The agent fails to control side effects or duplicates an irreversible action. A pairing defect without a realized side effect remains

a P06 issue. `object_attribute_preservation_failure`: A mutation changes or destroys an attribute that the task requires it to preserve.

The artifact qualifier records `code`, `configuration`, `data_transform`, or `external_state`. Factual communication falls under P09, including communication deliverables. External failure alone does not establish an agent-side defect.

P08: Verification sufficiency failure. P08 requires a materially necessary check that is due, with an opportunity to perform or consult it, but is omitted, insufficient, invalid, or stale.

`required_verification_omission`: The agent omits a check required by the task or the material risk of the action. A reviewer preference for additional assurance is insufficient. `verification_coverage_gap`: A performed check misses a relevant requirement, branch, item, or state. `invalid_verification_design`: The check cannot test the property at issue or relies on an invalid oracle. Unresolved disagreement about a checker's authority remains a boundary or evidence issue. `post_action_state_not_checked`: After a state-changing action, the agent performs neither needed result inspection nor target-state readback. An authoritative return that establishes the required state can make a separate readback unnecessary. `authoritative_result_not_checked_or_stale_verification_reused`: The agent fails to consult an authoritative result or reuses a prior check after a relevant change.

P09: Evidence–claim integrity failure. P09 requires an identifiable externalized claim unsupported by or conflicting with trajectory evidence. A bare evidence gap without a claim is insufficient. Missing support must be judged within a sufficiently available source record, not inferred from a missing excerpt.

`unsupported_extrapolation_under_uncertainty`: The claim extends beyond what uncertain evidence supports. A qualified claim that remains within those limits is excluded. `unsupported_quantitative_or_causal_inference`: A numerical, comparative, or causal inference lacks supporting evidence. `premature_epistemic_commitment`: The agent expresses a definite conclusion before the investigation supports it. A tentative hypothesis or proposed check is excluded. `source_absent_fact_fabrication`: The agent asserts a concrete fact, rule, number, state, or plan for which the authorized source material provides no basis. `fabricated_tool_observation`: The agent claims a tool call, output, or observation that did not occur or is unsupported by the available trajectory. `observed_state_misrepresentation`: The report contradicts a visibly observed state. An inferred internal misunderstanding alone does not establish a separate report-integrity event. `unsupported_action_recommendation`: An action recommendation exceeds the evidence supporting it. Executing an unauthorized action is a separate P01 question. `action_report_misalignment`: The report of an action, result, or achieved state conflicts with the action or state visible in the trajectory.

Unsupported claims do not establish intentional deception.

P10: Final-closure alignment failure. P10 requires terminal behavior—stopping, submission, handoff, or a completion claim—despite evidence that a closeout requirement remains unmet. Honest reporting of a blocker or incomplete work does not by itself qualify.

`premature_stop_before_terminal_state`: The agent ends work while a required terminal condition remains unmet. An explicitly requested pause is excluded. `completion_claim_before_terminal_state`: The agent claims completion while a necessary terminal condition remains unmet. `terminal_action_or_submission_nonexecution`: A required final send, save, submission, commit, or equivalent action is not executed. An executed action with a defective artifact is a P07 issue. `completion_protocol_nonexecution`: The agent skips a required acceptance, closeout, handoff, or completion procedure. `invalid_workaround_treated_as_completion`: A known invalid or temporary workaround is treated as satisfying the final requirement. An explicitly temporary workaround without a completion claim is excluded. `incomplete_terminal_handoff`: The final handoff omits material information needed to assess completion or continue the task.

D.3 Adjacent classes and multiple assignments

Table 12 distinguishes classes by the directly evidenced deviation; assignments are neither sequential nor mutually exclusive.

Table 12: Evidence separating adjacent working classes.

Classes	Distinguishing evidence
P01 / P02	Wrong workset or requirement versus wrong referent within understood work.
P02 / P03	Evidence from the wrong object or time slice versus misreading the correct observation.
P03 / P04	Initial misreading versus nonadaptation after distinct actionable feedback.
P04 / P05	Unheeded usable feedback versus independently evidenced avoidable non-progress.
P02 / P06 / P07	Wrong target with an interface-compliant request; intended target with an interface violation; intended target with an interface-valid but semantically defective result.
P07 / P08	Observed result defect versus inadequate checking without an established defect.
P03 / P09	Demonstrated observation-to-state misreading versus an independently unsupported or contradicted external claim.
P08 / P10	Inadequate checking versus a terminal act leaving a required condition unmet.
P09 / P10	Unsupported factual claim versus a completion claim while a required terminal condition remains unmet; both require distinct evidential propositions.

Assign multiple classes only for distinct evidenced deviations, not duplicate descriptions or mere consequences. Misinterpretation followed by ignored actionable correction can support P03 and P04 at different points; an interface-compliant action on the wrong target does not add P06. Legacy labels aid retrieval but do not determine assignments.

D.4 Process, boundary, evidence, and context fields

Table 13 records companion fields for local annotations. Process outcomes describe the trajectory's course; ORPMTF F separately follows the state associated with a particular event (Section E.6).

Table 13: Fields recorded separately from local breakdown classes.

Field	Recorded distinctions
Process outcome	Recovery success, normal immediate repair, progressive recovery, state-progression stall, partial progress, verified-state rollback, high recovery cost, coherent completion.
External or task boundary	Environment, tool, dependency, permission, infrastructure, harness, evaluator/checker, source-availability, and task-target ambiguity conditions.
Evidence status	Supported positive, reviewed no-positive, insufficient evidence, boundary-only, missing annotation, conflicting evidence, and unreviewed machine weak label.
Interaction context	Shared evidence, independent co-occurrence, user correction, explicit authorization, state-changing actions, concurrency, multi-condition entanglement, and causal chain.

Repair does not erase a local error; an external blocker can coexist with failure to use actionable feedback, but a positive class requires separate agent-side evidence. Reviewed absence and unavailable evidence remain distinct. Context flags add no class: even `CAUSAL_CHAIN` is not causal evidence by itself, and temporal order alone establishes no local cause.

D.5 Connecting local classes to ORPMTF

Table 14 connects the working classes in Section D.2 to event-level inspection questions. Class associations determine neither P/M/T nor either comparison; an event may have several independently supported associations or none. Section E.7 specifies mapping roles and evidence rules. Planned independent review introduces classes only after event reconstruction (Appendix G), unlike existing joint machine labels.

Table 14: Working classes and ORPMTF inspection questions, not empirical findings. Field definitions appear in Section 3.2.

Working class	**Event-level question**
P01 Task contract alignment failure	Does O capture outstanding requirements, and do P/M use the correct task contract?
P02 Goal–context binding failure	Do O and R concern the same task object and state version?
P03 Observation–state interpretation failure	What does R support, and are P/M warranted?
P04 Feedback integration and adaptation failure	Does new evidence require an update in M, and does T implement it?
P05 Exploration and progression control failure	Is leaving the state unchanged within P, and does it fulfill M? Permission to advance need not make advancement mandatory.
P06 Tool and interaction protocol failure	What did the action produce as R, including failure feedback, and how does it affect P/M?
P07 Action realization and result integrity failure	Does R support the intended state update, and should downstream obligations remain open or be reopened?
P08 Verification sufficiency failure	How does evidence bound P, and does it require further verification and a corresponding update in M?
P09 Evidence–claim integrity failure	Does the T conveyed by a claim exceed P or omit required uncertainty or outstanding issues?
P10 Final-closure alignment failure	Does terminal T lie within P and implement all of M, including required handling of outstanding O?

E ORPMTF Fields and Annotation Manual

The six fields specify a task progress state update event. Their controlled categories belong to individual fields, not an additional failure hierarchy. Appendix F describes machine production; Appendix G describes the separately planned staged validation.

E.1 Event boundaries and the annotation unit

Apply the event definition in Section 3.2: identify a distinct R and relevant O, then assess immediate handling T through explicit references, visible mutations, directly evidenced dependencies, or behaviorally required minimum premises. A sufficiently observed window can establish empty T; missing observation leaves T unobservable.

Record `overlay_event_id` and `event_checkpoint_locator`. Split events when R, object/version, or a consequential time point changes, or when combining independently assessable obligations would obscure different P/M/T judgments. Group actions implementing one indivisible disposition; extra spans or class labels do not create events. Repair belongs in the earlier event's F and may also form a new event if it independently qualifies. An inspected scope may contain multiple events or none.

E.2 O: OUTSTANDING OBLIGATIONS

O records event-relevant task requirements as checkable sentences, not an exhaustive solution plan. Each entry in o_items_json contains o_id, o_text, o_level, o_state_before, o_dependency_ids, and o_basis_locator_bundle. IDs are unique within the event and reused by P, M, and T.

The controlled levels are ATOMIC_ITEM, COMPOSITE_SUBTASK, WHOLE_TASK, CROSS_ITEM_DEPENDENCY, and UNOBSERVABLE. The pre-event states are NOT_REGISTERED, OPEN, PARTIALLY_SATISFIED, SATISFIED, BLOCKED, WAIVED_OR_WITHDRAWN, RECHECK_REQUIRED, CLOSED, CONFLICTED, and UNOBSERVABLE.

Absence from an excerpt does not establish NOT_REGISTERED; distinguish it from open and unobservable states using the reviewed record. Include satisfied or closed items when their validity is at issue. Dependencies require an explicit rule or necessary task relation, not chronological proximity. If an obligation or prior state cannot be reconstructed, retain the uncertainty rather than infer the missing information from the outcome.

E.3 R: LOCAL RESULTS AND THEIR LIMITS

R states the narrowest relevant proposition supported or contested by the available material. Its kind, r_kind, is one of OBSERVATION_OR_FEEDBACK, ACTION_EFFECT_OR_ARTIFACT, VERIFICATION_RESULT, INTERMEDIATE_CONCLUSION_OR_DECISION, BLOCKER_OR_CONSTRAINT_FINDING, OTHER, or UNOBSERVABLE. The proposition itself is recorded in r_proposition.

r_support_status is ESTABLISHED, PARTIALLY_ESTABLISHED, NOT_ESTABLISHED, CONFLICTED, or UNOBSERVABLE. It assesses the proposition, not the whole local work contract: an operation's failure may be established, while a success response may establish request acceptance without establishing the intended effect.

The result's limits are recorded separately:

Object. r_object identifies the file, entity, transaction, session, branch, or other target. r_object_binding is MATCHED, MISMATCHED, AMBIGUOUS, NOT_APPLICABLE, or UNOBSERVABLE, relative to the obligation's object.

Scope. r_scope describes actual coverage. r_scope_status is MATCHED, LIMITED, OVERBROAD, AMBIGUOUS, NOT_APPLICABLE, or UNOBSERVABLE.

Version. r_version records an identifiable artifact or state version. r_version_status is CURRENT, OLD_OR_DIFFERENT, NOT_APPLICABLE, or UNOBSERVABLE.

Time. r_time_anchor identifies when the result held or was checked. r_time_status is CURRENT, STALE, NOT_YET_VALID, NOT_APPLICABLE, or UNOBSERVABLE, relative to the event checkpoint.

Conditions. r_validity_conditions records relevant authorization, prerequisites, or continuing restrictions. r_condition_status is SATISFIED, PARTIAL, UNSATISFIED, REVOKED, CONFLICTED, NOT_APPLICABLE, or UNOBSERVABLE.

Retain visible identifiers even when their currency is unknown; do not infer missing descriptions from later outcomes. An ESTABLISHED test result may have LIMITED coverage: the limit constrains P without falsifying R. Record evidence for the proposition and its boundaries in r_locator_bundle; later overuse does not revise this assessment.

E.4 P AND M: PERMISSION AND OBLIGATION

E.4.1 CONSTRUCTING PERMITTED UPDATES

P describes justified task-state updates, not executable actions or a single optimal plan. Bound it using O, R, and checkpoint-available task instructions, user authorization or feedback, environmental state, and dependencies; the same R may permit different updates under different task requirements.

Use p_status=BOUNDED_SINGLE or BOUNDED_MULTIPLE when the permitted range can be bounded, and CONFLICTED or UNOBSERVABLE otherwise. The transition entries in p_updates_json contain transition_id, o_id, from_state, to_state, update_operation, and condition_id. Conditions that apply to a transition must be stated and supported.

The update-operation vocabulary is:

NO_CHANGE_OR_KEEP_OPEN: retain the relevant state or leave a requirement open. ADVANCE_PARTIAL: register supported partial progress. SATISFY_ITEM: treat the specified item as satisfied. BLOCK_OR_ESCALATE: register a blocker or escalate the unresolved requirement. DOWNGRADE_OR_WITHDRAW: reduce or retract an earlier conclusion or status. REOPEN_OR_RECHECK: reopen an item or mark it as requiring renewed checking. CLOSE_TASK: close the task under the applicable completion conditions. OTHER or UNOBSERVABLE: preserve an explicitly described alternative or an unresolved update.

These operations describe state handling, not commands: marking recheck pending does not perform a check. Record any required check in O and assess its fulfillment separately.

Record the basis in p_basis_text and p_basis_locator_bundle; p_excluded_stronger_update identifies a stronger claim or disposition that the evidence excludes.

The entries in p_updates_json are transition atoms, not freely combinable options. p_constraints_json records conditional alternatives, mutual exclusions, and updates required together. "T within P" requires both permitted atoms and a taken bundle satisfying these constraints; atom-wise permission alone is insufficient.

E.4.2 Identifying mandatory updates

M identifies updates required at the event. Each m_updates_json entry references a p_transition_id, with displayed transition text copied from P. For observable, non-conflicted judgments, M is a subset of P.

Set m_presence to NONE, ONE_OR_MORE, CONFLICTED, or UNOBSERVABLE. The basis categories in m_basis_types_json are EXPLICIT_TASK_OR_COMPLETION_RULE, USER_FEEDBACK_OR_AUTHORIZATION, VERIFICATION_OR_VISIBLE_STATE_CHANGE, DEPENDENCY_INVALIDATION, OTHER_EXPLICIT_BASIS, and UNOBSERVABLE. The accompanying m_basis_text and m_basis_locator_bundle explain why the cited condition requires this update now. The source category alone does not establish necessity.

Several permissible next actions can share a required state disposition: the agent may check, ask, or pause while keeping new-version validation unresolved. Permission and efficiency preferences alone do not establish necessity. If task evidence cannot establish the required change, retain a conflicted or unobservable M rather than infer no requirement or an omission. Visible prerequisite invalidation may require withdrawing dependent completion status; record affected obligations in m_affected_dependency_ids. If a candidate M update is absent from P, reconcile P/M before assessing T; resolve conflicting task rules before assigning a determinate fulfillment judgment.

A prerequisite due point. In X19 (Appendix A), repairing phone access does not establish a separate Venmo session: the earlier code block stopped before that login. The unresolved login matters when execution reaches its first dependent Venmo use, not as a demand to complete every remaining operation immediately after phone repair. The attempted use fails at an undefined token; later login and retry belong to F or a later event. A fulfillment judgment therefore needs the prerequisite, its due point, and the intervening response window. If the visible action could instead be a probe and no commitment is established, retain that ambiguity rather than infer an omission from the error alone.

E.5 T and the two assessment axes

E.5.1 Recording taken updates

T describes what the agent enacts in the immediate observation window. t_behavior_text explains the behavior, and t_updates_json records its obligation-linked transitions using o_id,

from_state, to_state, and update_operation. The immediate evidence is recorded in t_behavior_locator_bundle.

Behavioral modes and evidential expression are separate. The modes in t_behavior_modes_json are PROBE_OR_VERIFY, CONTINUE_OR_REUSE, REDO_OR_REPEAT, REPORT, MARK_OR_CLOSE_ITEM, CLOSE_TASK, WITHDRAW_OR_REOPEN, ESCALATE, NO_VISIBLE_UPDATE, OTHER, and UNOBSERVABLE. These describe what is visible without determining whether it is permitted or mandatory.

The expression modes in t_expression_modes_json are: EXPLICIT_STATEMENT: the agent states the relevant task status. VISIBLE_STATE_MUTATION: an observable action changes the relevant task state. BEHAVIORALLY_REQUIRED_PREMISE: the observed behavior necessarily presupposes a specific task state. NONE: no task-state update is expressed in the inspected window. AMBIGUOUS: more than one state interpretation remains plausible.

t_minimum_presupposition states the minimum premise required by the behavior's semantics, not an inferred hidden belief. Tests, searches, and reversible probes do not automatically assert their target conditions. If exploration or recovery remains a plausible alternative to a task-state commitment, record AMBIGUOUS and the alternative in t_competing_interpretation.

The canonical no-update form uses t_behavior_modes_json=[NO_VISIBLE_UPDATE], t_updates_json=[], and t_expression_modes_json=[NONE]. Production records may instead pair an observed behavior, such as probing, with an explicit empty update set. Both require a visible window with no established state transition: activity alone does not establish T. Empty T is neither a no-event record nor proof of omission; do not fill it with a fabricated NO_CHANGE_OR_KEEP_OPEN transition.

Action and report can differ. In the reviewed payment example P18-HS-0118, the tool action submits the correct Mastercard payment-method identifier, but the confirmation names a Visa card. The submitted identifier and the reported method are distinct observable facts: the inaccurate report does not erase the correct action, and the correct action does not validate the report. T retains separate requirement-linked entries when both fall within one checkpoint; a later independently assessable report is a separate event under Section E.1. Neither representation turns the Visa claim into an observed payment mutation, nor does submission alone establish settlement. This distinguishes action-established and report-adopted states without inferring a hidden belief.

E.5.2 Comparing permission and fulfillment

Table 15 defines two separate judgments: t_within_p tests whether T stays within P, including bundle constraints; m_realized_in_t tests whether T implements M. When M is explicitly empty, fulfillment is NOT_APPLICABLE regardless of T or opportunity. Otherwise, YES, NO, and PARTIAL require an established nonempty M.

Table 15: Controlled values and decision rules for the two ORPMTF judgments.

Field	Controlled value	Decision rule
Permission	YES	P and T are established, and the taken updates form a permitted bundle.
	NO	A taken update lies outside P or the bundle violates an applicable constraint.
	UNOBSERVABLE	P or T cannot be determined well enough for comparison.
	CONFLICT	The permission rules remain conflicted.
Fulfillment	YES	Every mandatory update is implemented.
	NO	None is implemented, with sufficient opportunity and visibility.
	PARTIAL	Some, but not all, of several mandatory updates are implemented.
	NOT_APPLICABLE	M is explicitly empty: m_presence=NONE.

Field	Controlled value	Decision rule
	UNOBSERVABLE	M is not established as empty, and evidence needed to judge M, T, or implementation opportunity is unavailable.
	CONFLICT	The mandatory-update rules remain conflicted.

These reporting conventions do not recode historical labels. Preserve transition-level correspondence and an observation window sufficient to judge the required update. Truncation or a gap does not establish omission.

An unchanged state can remain within P while omitting M; conversely, an agent can implement M for one obligation while making an impermissible closure claim about another. Empty T alone determines neither permission nor omission: P must be assessable, and omission requires a nonempty M with sufficient opportunity and visibility.

comparison_basis_note identifies compatible, offending, and missing transitions and their evidence, without adding an overall verdict or combined score. Identifier matching supports mechanical checks; natural-language compatibility and necessity require semantic judgment. Section E.7 distinguishes event-level uncertainty from record-level dispositions.

E.6 F: RECORDING DOWNSTREAM FATE

F records the state's observed fate after the immediate update window: later use, explicit dependency, exposure, response, and terminal status.
f_dependency: ISOLATED_NO_FURTHER_USE, CONTINUED_USE, EXPLICITLY_PROPAGATED, WITHDRAWN_FROM_DEPENDENCIES, or UNOBSERVABLE.
explicit_dependency_status: NONE_VISIBLE, CONTINUED_WITHOUT_EXPLICIT_DEPENDENCY, EXPLICIT_SINGLE, EXPLICIT_MULTIPLE, WITHDRAWN, or UNOBSERVABLE.
f_exposure: NOT_EXPOSED_IN_TRACE, AGENT_SELF_DETECTED, USER_EXPOSED, TOOL_OR_VERIFICATION_EXPOSED, ENVIRONMENT_OR_SOURCE_EXPOSED, EXTERNAL_REVIEW_ONLY, or UNOBSERVABLE.
f_response: NO_RESPONSE, ACKNOWLEDGED_ONLY, WITHDRAWN_OR_REOPENED, REPAIRED_OR_REVERIFIED, ESCALATED, PARTIAL_OR_FAILED_RECOVERY, FULL_RECOVERY, or UNOBSERVABLE.
f_terminal: REMAINED_OPEN_OR_HONESTLY_BLOCKED, REOPENED_AND_PENDING, CLOSED_AFTER_RESOLUTION, CLOSED_WITH_ISSUE_UNRESOLVED, NO_TERMINAL_DECISION, TRACE_CENSORED, or UNOBSERVABLE.

f_summary preserves the sequence, including intermediate attempts; f_locator_bundle anchors it. CONTINUED_USE requires evidence of reuse, while EXPLICITLY_PROPAGATED and EXPLICIT_SINGLE/EXPLICIT_MULTIPLE require directly evidenced dependency links. Temporal order alone establishes neither. Acknowledgment is not repair; attempted or asserted repair is not verified recovery; external review is not agent detection. Censoring may leave terminal fate unknown despite observed reuse or repair.

F does not revise earlier O/R/P/M/T judgments. Later evidence may describe what happened to the earlier state or establish a new event, but successful recovery cannot retroactively permit an earlier unsupported update.

E.7 EVIDENCE, ABSTENTION, AND LINKS TO LOCAL CLASSES

A locator identifies the source object and supporting passage or state observation, preserving speaker/tool attribution and object binding. Supply a step, message, line range, or structured pointer with enough local content for verification. Separate locators may support the result, its binding, and the applicable rule. Keep contrary evidence in counterevidence_locator_bundle and plausible alternatives in competing_explanations_json, assigning each item to its availability stage.

Appendix G specifies planned staged validation at supplied checkpoints, not exhaustive event discovery. Within that protocol, `stage_a_frozen` and `stage_b_frozen` record judgment locks, and `stage_c_complete` records completion. Evidence-release records must establish what reviewers could see; these flags alone prove neither staged concealment nor semantic correctness. Do not backfill them for full-trajectory production as evidence of unperformed staged review.

The controlled `abstention_reason` values are `LOCATOR_UNRESOLVED`, `NO_DISTINCT_RESULT`, `TASK_REQUIREMENT_UNCLEAR`, `R_BINDING_UNCLEAR`, `P_NOT_BOUNDED`, `M_RULE_CONFLICT`, `T_NOT_UNIQUELY_INFERABLE`, `DEPENDENCY_NOT_VISIBLE`, `FATE_NOT_VISIBLE`, `COMPOSITE_INSEPARABLE`, `BLINDING_LEAK`, `OTHER`, and `NONE`. A missing later fate need not invalidate an otherwise supported P/T comparison. Keep uncertainty at the affected field or event rather than converting it into a negative label.

Record-level dispositions include mapped events, no event within reviewed windows, insufficient evidence, source/access blocks, superseded records, and out-of-scope material. A no-event record needs its inspected scope, reason, and evidence; it differs from an event with empty T. Neither an empty event array nor zero positive local classes establishes trajectory-wide absence of events.

In the planned independent review, Stage D links local annotations to the reconstructed event through `phase18_parent_ids` and `phase18_submechanism_ids`. The roles in `phase18_projection_roles_json` are `ANTECEDENT`, `LOCUS`, `MANIFESTATION`, `CONSEQUENCE`, `RECOVERY`, `BOUNDARY`, and `OUT_OF_SCOPE`; an empty mapping is valid. `phase18_mapping_note` explains multiple or disputed roles. These links neither establish causality nor determine P/M/T from a class name.

Keep identity, readable source version, review role, annotation mode, allowed use, and human-validation status outside blinded semantic records. Documented wrappers or field aliases must preserve logical content and references; missing applicable evidence is not "not applicable." Administrative checks establish readiness, not human confirmation.

E.8 Worked developmental case

Illustrative judgment. Consider the response immediately after edits to previously tested code. The old test supports the old version; the task requires validation of the submitted version. P permits retaining the old result within its scope and reopening validation, while M requires treating the new version as unverified. An explicit "awaiting validation" status enacts M without requiring an immediate retest. Declaring the new version validated solely from the old test exceeds P and omits M. F then records whether later work relies on that claim, withdraws it, or revalidates the version.

Figure 8 illustrates an event-level reconstruction from a developmental case, separating supported local work, the agent's state update, and its observed aftermath.

F Machine Annotation and Record Preparation

Machine Core judgments provide local-breakdown context; ORPMTF annotations support event-level state-update analysis. Appendices D–E define their semantics. This appendix specifies the inputs, output separation, comparison targets, and record checks for the procedure described in Section 4.

F.1 Source binding and reading scope

Each assignment binds one trajectory through the identity fields in Section B.2, its source path, and any required native-object selector. Annotators read the full trajectory in native order and record direct evidence for both layers. Selection records identify the adopted version; unresolved identity conflicts remain held. Access to the complete trajectory does not make later outcomes evidence for an earlier P/M/T judgment: the annotation instructions assign those judgments to event-time evidence and F to subsequent execution.

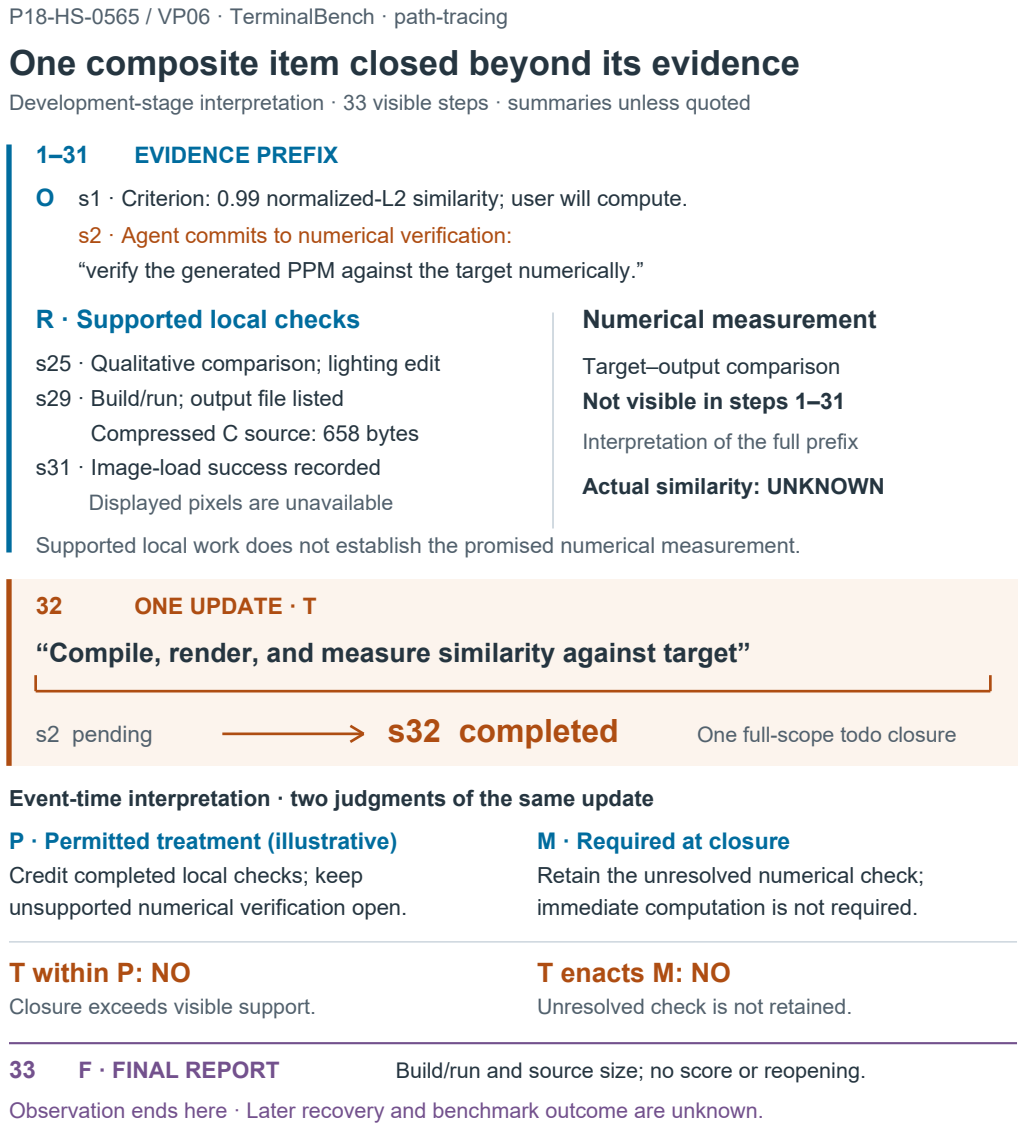


Figure 8: **Supported local checks coexist with unsupported composite closure.** Developmental case P18-HS-0565/VP06 (TerminalBench); steps are one-based. Absence of a numerical target–output comparison reflects the accepted reading of the full visible prefix s1–31, not an individual tool response or hindsight from s33. At s32, the agent closes a compile/render/measure todo without asserting a score ≥ 0.99. P is illustrative, not exhaustive; M requires keeping the unresolved measurement obligation open, not immediate computation. Both NO judgments concern one event. F ends at s33; actual similarity, benchmark success, and later recovery remain unknown. This reconstruction is not independent human validation or an endorsement of the ORPMTF mapping by the earlier human review.

F.2 Independent annotation and adjudication

Primary (GPT-5.6 Luna) and Blind (DeepSeek-V4-Flash) use the same trajectory and annotation specification, without access to each other's outputs. Their records are saved separately before comparison. Agreements are consolidated; disagreements are submitted to GPT-5.6 Sol with the trajectory, both annotations, and the comparison. Adjudication retains, revises, or rejects disputed judgments and preserves unresolved interpretations.

Here, double-blind denotes separation of the two annotation outputs, not concealment of source identity or later trajectory content. It differs from the staged evidence release specified for future independent validation in Appendix G.

F.3 Comparison targets

Comparison covers both layers. Core decisions are checked with their rationales, event links, and supporting spans. ORPMTF comparison concerns event boundaries, affected requirements, local results, permitted and mandatory updates, actual handling, and downstream fate, including both assessment axes. Events are interpreted through their source evidence and affected requirements, not just matching identifiers. No-event and unobservable records retain the distinct meanings defined in Appendix E.

The saved record retains the adopted interpretation and its evidence. Structural normalization can reconcile equivalent field names or wrappers, but does not resolve differences in necessity, permission, or behavioral meaning. Those differences require a source-based judgment; an unresolved interpretation is not converted into a negative label.

F.4 Admission gates

Table 16 summarizes checks on saved outputs.

Table 16: Material-admission checks.

Gate	Required check
G0: identity	Account for every assigned slot and verify its native task, trajectory, and attempt binding.
G1: provenance	Recover the saved annotation, dispatch identity, reading scope, and adopted version, not merely a completion summary.
G2: Core	Check every parent decision and rationale, positive-event links, and referenced objects; unresolved is not negative.
G3: ORPMTF	Check applicable event content, obligation items, hard requirements, axes, and internal references.
G4: localization	Resolve each locator to same-source evidence with correct quotation, speaker, and tool-call attribution.
G5: fidelity	Confirm that normalization preserves evidence objects and adopted components match the saved final output.
G6: verification	The coordinator rechecks G0–G5 against saved material and records conflicts or unperformed checks.

The dictionary has 84 logical definitions: 20 record-level, one candidate-window, 57 event-level, and six within each obligation item, not 84 JSON keys per event. Removing four administrative event definitions leaves 53 content definitions subject to fixed applicability conditions. Each event must reach at least 90% applicable-content coverage and pass every hard requirement; obligation items are checked separately. Administrative keys or other events cannot compensate for missing content. Unknown non-hard applicability contributes to the denominator upper bound without numerator credit; this lower-bound coverage must still reach 90%. Unknown hard conditions or missing required content block admission. Permitted explicit empties differ from missing fields; no-event records receive separate boundary checks rather than event scores.

Locators for counterevidence, fate, and boundaries must meet G4. Adapters may normalize equivalent wrappers, aliases, JSON presentation, or locator syntax without changing content or evidence objects. Ancestor pointers, adjacent messages, or similar text elsewhere cannot substitute for the declared evidence; conflicting keys or aliases remain unresolved. Mechanical checks verify references and explicit structural contradictions, not compatibility between non-identical natural-language statements. Preserve comparison bases and local evidence, stating when semantic support was not reevaluated. Admission establishes material readiness, not semantic validity or human agreement.

F.5 Consolidation and retained provenance

Batches retain event-bearing candidates, justified boundaries, and held rows with separate Core and ORPMTF statuses. Coordinator checks precede admission. Dated snapshots link each adopted record to its source and assignment, evidence, reading scope, prior-output access, component changes, and acceptance decision. When Core and ORPMTF come from different versions, each component retains its predecessor and cross-layer links. Analyses count only the adopted record, excluding superseded outputs, held candidates, and archive copies.

G Protocol for Future Independent Validation

This protocol is planned, not completed, and is not the procedure used to produce the present results. The proposed validation would test the definitions through independent human judgments on material separated from development. Appendix E supplies field semantics; the procedure here controls sampling, information exposure, first judgments, and reporting.

G.1 Sample and instrument pilot

The planned cohort comprises 48 cases, including a 12-case instrument pilot: 16 strong candidates, 16 boundary or near-negative cases, 12 main-account versus fallback contrasts, and four control or no-positive cases. Purposive selection will cover at least four environments, with no dataset supplying more than one third. Cases or linked lineages used to develop the construct or revise the instrument are excluded from formal validation, regardless of identifier changes. The 48-case target is therefore not a guaranteed formal-validation denominator; pilot eligibility and formal counts must be reported separately. Machine labels guide retrieval, not judgments.

Eligible cases require resolved identities and readable, locatable windows. Source failures are neither positives nor negative controls; readable material may leave specific fields unobservable. Selection provenance stays outside review packets. Counts follow Section B.3; same-trajectory events are not independent replications.

Expansion beyond the pilot requires at least 10/12 valid, locatable judgments and no systematic source, structure, or instruction defect. Assessors and counting rules must distinguish supported events, no-event, field-limited judgments, substantive abstention, and material failure. Validity does not require a mismatch. The pilot tests no update versus missing observation, mandatory versus optional change, and commitment versus exploration. Systematic defects stop affected assessment; at most one documented protocol-error revision is allowed. Changed materials or instructions keep pilot and formal findings separate. Expansion cannot pursue favorable results, and pilot passage does not pass the empirical gates.

G.2 Independent reviewers and controlled materials

Reviewers will make first judgments without seeing each other's records. Stage-specific packets conceal later behavior and outcomes, rather than asking reviewers to ignore visible continuations. Opaque identifiers replace administrative identities; packets omit prior machine verdicts, development judgments, model identity, class labels, selection reasons, and outcome metadata while retaining task instructions, environment rules, and observations. The administrator controls assignments and release order to prevent leakage across shared trajectories or lineages and retains source mappings and stopping points for locator checks. Recognizable task or environment cues remain documented limits on anonymity.

G.3 STAGED RECONSTRUCTION

Table 17: Evidence release and locked records in the proposed independent validation protocol.

Stage	Revealed material	Record locked before proceeding
A	Requirements, relevant environment information, prefix through candidate R	O/R/P/M, evidence, competing readings, provisional eligibility limits
B	Immediate subsequent behavior, not later continuation or outcome	T, final event eligibility, permission and mandatory-fulfillment judgments against frozen P/M
C	Later trajectory and available terminal material	F, locators, and censoring/observability limits
D	Local parent and child annotations	Evidence-based class links to the completed reconstruction

Table 17 specifies staged release. At A, the prefix bounds permitted updates or allowed next steps, not all task solutions. At B, a taken update or sufficiently visible no-update window establishes final event eligibility. Supplied-window assessment tests reconstruction, not unrestricted event discovery. Appendix E governs compatible P bundles, M's necessity basis, observable T, and omission opportunity.

Reviewers also lock local work quality against its own contract and evidence before F or class labels appear; its exact lock point requires prespecification (Section G.6). A is locked before B, and B before C; later information cannot revise earlier judgments. Section E.7 distinguishes judgment locks from evidence-release records. Reviewer independence and staged concealment remain separate safeguards.

G.4 ABSTENTION, DEFECTS, AND ADJUDICATION

Field-specific uncertainty and no-event decisions follow Appendix E; missing F need not invalidate an earlier supported comparison. A material or blinding defect discovered after locking stops the affected judgment without replacing its first record. Corrections identify affected material and fields; authorized reassessments remain separate. Agreement uses first independent records. Adjudication preserves both evidence-backed readings and records its decision, rationale, changed fields, and residual uncertainty. Consensus cannot fill source gaps or justify earlier P/M/T from later outcomes.

G.5 REPORTING AND EMPIRICAL GATES

Reports will account for selection, exclusions, pilot/formal membership, source or protocol failures, and valid, invalid, unobservable, and disputed judgments. They will retain first field judgments, separate permission and mandatory-fulfillment disagreements, fate/censoring, and adjudication changes. Every statistic needs its unit and denominator; recurring sources require event, record, trajectory, and lineage counts. Fields excluded from a statistic remain visible in coverage. Agreement metrics, semantic equivalence rules, and thresholds must be fixed before collection; identifiers or wording alone do not determine agreement.

Four tests govern claims. H1 tests contrasts among local work quality, state handling, and fate. H2 tests independent reconstruction against preregistered agreement and abstention criteria, especially P/M/T judgments made before outcomes are revealed. H3 tests distinct relations: each needs a predicate, supported instance, boundary, and control or recovery case. The broad claim requires at least three predicates; cross-environment applicability cannot depend on redefining them. H4 tests diagnostic gain against native and combined baselines (Appendix H). Review judgments remain separate from machine labels, and reconstruction alone supplies no H4 result.

The strict prefix audit is a fallback only with stable reconstruction and diagnostic gain. Without gain, the contribution is protocol-level; failed key-field reconstruction suspends the corresponding empirical claim.

G.6 Implementation Parameters and Reporting Scope

Before validation, fix case/pilot membership within Section G.1's quotas; necessity observation windows and the local-quality lock point; cross-packet assignments and releases; reviewer staffing, qualifications, training/calibration, and adjudicator access; and unresolved-conflict handling. Section G.5 specifies statistical and coverage requirements. The future report must document these settings, protocol versions and deviations, and field-level results.

H Protocol for Future Diagnostic-Gain Comparisons

H.1 Target and native baselines

The planned H4 comparison tests changes in case inclusion/decidability, first-mismatch localization, recovery judgment, or next-step disposition. Additional fields or relabeling alone do not count. Formal pairs, comparator versions, and results remain prospective; Appendix A.7 reports separate developmental comparisons.

Before pairs are frozen, source review must establish each baseline's version, native analytical unit, supported questions, evidence requirements, locators, and adaptations. Reviewers credit all distinctions supported by its specification, including necessity, implicit commitments, invalidation, and repair, without importing ORPMTF distinctions. Native first judgments precede ORPMTF exposure and retain evidence, scope limits, and unresolved questions. This manual comparison assesses representational coverage, not published-system performance; unavailable inputs or instrumentation limit applicability rather than demonstrate superiority.

H.2 Pair eligibility and families

Mechanical matches are candidate keys; formal pair identifiers follow eligibility review. Before judging gain, freeze each pair's primary family, separate control role, baseline set/versions, controls, and residual differences. Require reconstructable event boundaries, comparable local contracts, and a common diagnostic question. Record task/dataset identity, visibility, terminal category, and reasons for cross-task grouping; match result type, stage, obligation density, verification access, and continuation where relevant.

Exclude unequal outcome dependence or evidence access, unobservable-versus-supported-positive contrasts, and confounding residual differences. Shared class labels do not equate closure-only and intermediate events. Separate controls from intended contrasts: hold terminal category constant for family B, but preserve fate differences for D. Retain ineligible candidates.

Table 18: Candidate pair families and controls for the proposed diagnostic-gain comparison.

Primary family	**Intended contrast and controls**
A: similar result, different state handling	Comparable objects, scopes, versions, and local contracts; different permitted, mandatory, or taken updates.
B: similar final outcome, different intermediate mismatch	Matched task/event evidence and terminal category; different state relations. A changed requirement or dependency counts only if it changes a decision.
C: local error versus state mismatch	Contained local error without mismatch versus locally correct work with mismatch; judge local quality and both state relations independently of machine labels.
D: similar mismatch locus, different fate	Comparable earlier mismatch locations, different continuation/recovery; preserve the earlier judgment and credit process baselines that distinguish fate.

Assign one primary family before review, distinct from the later primary gain category. A no-gain control is a separate role, outside minimum family coverage; record its prespecified expectation separately from its observed result.

H.3 Assessment sequence and information access

Screen candidates and freeze families, controls, baselines, and pairs before native review. ORPMTF review, combined-baseline assessment, delta classification, and gate evaluation follow. Both representations answer the same question with comparable evidence; disclose method-native access differences, and never supply decisive evidence to only one side. Native reviewers see neither expected deltas nor ORPMTF explanations; ORPMTF reviewers see neither native judgments nor expected deltas. Exposure controls cover overlapping packets and shared lineages.

ORPMTF follows Appendix G's staged release, local-quality lock, and separate P/T and M/T comparisons. Inclusion and localization use event-time evidence; recovery may use later evidence without revising earlier P/M/T. Preserve native first records after reveal, separate from adjudication. Defects and reassessment follow Section G.4.

H.4 Decisions and delta outcomes

A material decision change needs evidence and a representation-level distinction. Reviewer disagreement alone is insufficient: adjudication must ask whether the baseline already expresses the corrected decision.

Qualifying changes concern: (1) *inclusion/decidability*, including boundary, abstention, or observability decisions under comparable evidence; (2) *first-mismatch localization*, the earliest supported event or locus within the inspected scope, not a cause inferred from later failure; (3) *recovery*, the affected state, recovery status, or remaining repair requirement, distinguishing acknowledgment and attempts from evidenced recovery; or (4) *next-step disposition*, a warranted continuation, revalidation, withdrawal, reopening, rebinding, escalation, closure, or unresolved status under an applicable rule. A proposed disposition does not establish intervention benefit. Assign one primary category in this fixed order; other changes remain secondary flags, not independent demonstrations.

Table 19: Outcome rules for the proposed diagnostic-gain comparison.

Outcome	**Rule**
`MATERIAL_DELTA`	At least one supported change in the four decisions, subject to native/combined coverage review.
`PARTIAL_DELTA`	Useful detail or narrower uncertainty without a decision change.
`NO_DELTA`	Unchanged diagnosis without that addition.
`UNRESOLVED`	Insufficient specifications, controls, or authorized evidence for a determinate comparison.

Freeze decision vocabulary and equivalence rules, including the distinction between changed observability and reduced uncertainty within an unchanged decision. Report pair invalidity as eligibility, not a fifth outcome, and retain unresolved evidence without treating it as negative. Appendix A.7's developmental comparisons use different baselines and three historical values (`DELTA_SUPPORTED`, `NO_DELTA`, `COMPARISON_INSUFFICIENT`); their `NO_DELTA` is not interchangeable with the outcome here.

H.5 Individual and combined coverage

Assess each applicable baseline and the prefrozen set jointly, recording the fields and native operations used by the strongest composed account. Credit a combination that reaches the same decision even if no component does alone. A new distinction, access rule, or inference absent from the components constitutes an extension, not native or composed coverage. Each candidate gain retains both decisions, evidence, the changed field, and the strongest individual and combined explanation; final delta follows this audit.

H.6 H4 Gate and Prospective Reporting Scope

The gate requires formal review of at least three of four primary families; distinct prefrozen pairs supporting at least two primary gain categories across at least two environments; and completed adjudication of at least one prespecified no-gain control. Secondary flags do not count. Every applicable high-overlap baseline needs a native first judgment and individual/combined coverage assessment (Section H.5), leaving material diagnostic decisions the combined baseline set cannot reproduce without loss.

Reports retain candidate-to-pair flow, the frozen denominator, family/control roles, baseline versions and locators, first judgments, eligibility, disagreements, outcomes, coverage audits, and gate decisions. Partial/no gains and unresolved/invalid comparisons remain denominator-linked. Reused sources require underlying event and trajectory counts, not additional independent evidence. Report controls' expected and observed results separately. Gate failure cannot trigger negative-pair replacement, family changes, baseline switching, or expansion to obtain a pass. Any passing claim concerns the named decisions, representations, and cases, not improved task performance; Section G.5's limits also govern P–T-only results.

Before execution, fix the source-grounded comparator roster, pair manifest, reviewer assignments, method-specific inputs, decision-equivalence criteria, and disputed-coverage adjudication under Appendix G. Literature-driven baseline additions must precede pair freezing.

I Task Outcomes and State-Alignment Analysis

I.1 Inputs and outcome linkage

The comparison starts from the model-attributable records underlying Table 1. Excluding SWE-multilingual, whose outcome linkage was unavailable for this analysis, leaves 3,518 trajectories from five sources. Native evaluation metadata assign 2,381 successes and 1,072 failures; 65 outcomes remain unresolved. Of the scored trajectories, 128 have no annotated event and 546 have events but no jointly judged event. The comparison therefore includes 2,779 trajectories: 1,993 successes and 786 failures.

Outcomes are joined to a unique source-bound trajectory and attempt, with its native task identifier and evaluation record. A task identifier alone is insufficient when several runs exist. ORPMTF events attach to that same trajectory key. Missing or ambiguous outcome links remain unresolved rather than becoming failures.

AppWorld uses the evaluator's success boolean from a successfully read evaluation result. CLAW uses its completed-run pass decision, with the native completion threshold of 0.75. DeepSWE and TerminalBench use binary trial reward (1 or 0). For tau2bench, success is official reward 1 within numerical tolerance, and infrastructure interruptions without a valid score remain unresolved. These are source-specific acceptance rules, not a pooled cross-environment performance measure.

I.2 Trajectory means and model weights

For trajectory i, let J_i be the number of events with nonempty M and resolved judgments on both axes, and A_i the number within P that enact M. The trajectory statistic is $a_i = A_i/J_i$, defined only when $J_i > 0$. Within each source and outcome group, we average a_i equally across eligible trajectories:

$$\bar{a}_y = \frac{1}{n_y} \sum_{i:Y_i=y,\, J_i>0} a_i, \qquad \Delta = 100(\bar{a}_+ - \bar{a}_-).$$

Here $+$ and $-$ denote native success and failure; Δ is in percentage points. This is not an event-pooled percentage: trajectories with many events do not receive greater weight.

Table 20: Source-level success comparisons. Means are percentages; pooled and model-equal differences are percentage points.

Source	n_+	$\bar{a}_+$	n_-	$\bar{a}_-$	Pooled Δ	Equal Δ
AppWorld	739	90.86	148	56.66	+34.19	+24.70
CLAW	338	35.38	160	28.44	+6.94	+6.26
DeepSWE	157	60.23	252	52.38	+7.85	-2.97
TerminalBench	133	84.37	122	57.32	+27.05	+20.18
tau2bench	626	88.37	104	69.17	+19.20	+17.57

Table 21 supplies the corresponding model-level counts and means. Model-equal differences average the within-model success–failure differences over models represented in both outcome groups, using weight $1/K$. All 29 source–model groups in this comparison have such support. Each source is analyzed separately.

Table 21: Outcome-linked trajectory groups. N is the input group size; Scored has a resolved native outcome. n_+ and n_- additionally require $J_i > 0$. Means are percentages of the trajectory-level within-permission/enacted share; differences are percentage points.

Model	N	Scored	n_+	$\bar{a}_+$	n_-	$\bar{a}_-$	Δ
AppWorld							
DeepSeek-V4-Flash	102	98	52	64.74	38	29.39	+35.36
DeepSeek-V4-Pro	88	87	71	80.28	8	68.75	+11.53
GPT-5.4	133	133	111	90.93	18	58.33	+32.60
GPT-5.6 Luna	129	129	84	97.62	16	59.38	+38.24
GPT-5.6 Sol	126	126	117	97.93	5	80.00	+17.93
GPT-5.6 Terra	161	152	124	97.72	15	93.33	+4.38
MiMo-V2.5	101	100	69	86.13	23	53.03	+33.10
MiniMax-M3	155	155	111	92.49	25	68.00	+24.49
CLAW							
DeepSeek-V4-Flash	110	110	68	22.30	27	25.93	-3.62
DeepSeek-V4-Pro	167	167	99	16.67	52	16.35	+0.32
MiMo-V2.5	173	173	101	44.88	50	31.67	+13.22
MiniMax-M3	155	150	70	60.83	31	45.70	+15.13
DeepSWE							
DeepSeek-V4-Flash	92	91	5	33.33	60	35.03	-1.69
GPT-5.4	99	99	36	64.70	42	71.76	-7.06
GPT-5.6 Luna	91	91	29	64.59	30	57.97	+6.62
GPT-5.6 Sol	88	88	40	58.50	17	60.92	-2.42
GPT-5.6 Terra	103	103	41	62.13	38	65.10	-2.97
MiMo-V2.5	90	90	6	33.33	65	43.63	-10.30
TerminalBench							
DeepSeek-V4-Flash	40	40	12	61.51	12	76.39	-14.88
DeepSeek-V4-Pro	82	81	22	82.58	27	33.33	+49.24
GPT-5.4	72	72	23	95.65	19	65.79	+29.86
GPT-5.6 Luna	75	72	27	95.06	30	73.11	+21.95
GPT-5.6 Sol	31	31	11	100.00	3	100.00	+0.00
MiMo-V2.5	82	77	21	71.43	19	45.61	+25.81
MiniMax-M3	65	65	17	76.47	12	47.22	+29.25
tau2bench							
DeepSeek-V4-Flash	249	241	178	89.99	21	82.22	+7.77
DeepSeek-V4-Pro	182	173	126	89.82	19	75.00	+14.82
MiMo-V2.5	254	239	163	90.14	33	67.17	+22.97
MiniMax-M3	223	220	159	83.60	31	58.88	+24.72

I.3 DEEPSWE COMPOSITION DECOMPOSITION

Let u_m and v_m be the success and failure means for model m, and $s_m = n_{m,+}/n_+$ and $f_m = n_{m,-}/n_-$ its shares of the two groups. The pooled difference separates exactly into:

$$\Delta = 100 \sum_m \frac{s_m + f_m}{2}(u_m - v_m) + 100 \sum_m (s_m - f_m)\frac{u_m + v_m}{2}.$$

The first term is a within-model component and the second a composition component. For DeepSWE they are -3.15 and $+11.00$ percentage points, yielding the pooled $+7.85$. Equal model weights yield -2.97 points; common pooled weights $(n_{m,+}+n_{m,-})/(n_+ + n_-)$ yield -3.36. Flash and MiMo together contribute 11/157 successful versus 125/252 failed trajectories, explaining how their unequal representation affects the pooled comparison. These are arithmetic decompositions, not causal model effects.

I.4 EXCLUDING ANNOTATED CLOSURE UPDATES

An event is closure-related here when `CLOSE_TASK` appears in its P, M, or T update operations, or its T behavior modes. We remove those events and recompute A_i, J_i, and eligibility; trajectories with no remaining jointly judged event leave this comparison. This tests dependence on explicit closure annotations, not all submissions, late actions, or terminal feedback.

Table 22: After excluding annotated closure events. The event denominator and trajectory eligibility are recomputed. Means are percentages and differences are percentage points.

Source	n_+	$\bar{a}_+$	n_-	$\bar{a}_-$	Δ
AppWorld	375	86.93	79	60.40	+26.54
CLAW	171	51.02	79	44.09	+6.93
DeepSWE	118	82.93	195	71.09	+11.84
TerminalBench	98	89.67	94	66.49	+23.19
tau2bench	606	89.32	102	68.89	+20.43

The AppWorld, TerminalBench, and tau2bench pooled differences remain positive under this exclusion.