REPROSADDLER: REPRODUCING AGENT FAILURES FOR HARNESS OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Improving a deployed language-model agent means targeting the failures its users actually encounter, not those already covered by benchmarks or training data. Such failures depend on more than the task. The same request may succeed for one user and fail for another, depending on the files, settings, and history in their environment. Yet this environment cannot be inspected, since privacy constraints prevent it from being shared or copied, and the remaining record of what the user reported and what the agent did cannot pin down why the agent failed. Without an identified cause, a failure cannot guide repair of the agent harness (its prompts, tools, and execution logic). We introduce REPROSADDLER, which turns user dissatisfaction signals, namely textual feedback and interaction records, into fresh executable failure scenarios that serve as optimization data without access to the user environment. A trusted compiler first fixes a de-identified, value-free failure specification that the generator cannot revise. The generator then rebuilds the workflow in a public donor environment and proposes switchable conditions as hypotheses about the cause of the failure. Validation against a control with all conditions off, followed by trace-guided refinement and ablation, ensures that each scenario reproduces the observed discrepancy rather than an arbitrary error. Experiments on two complex benchmarks show that REPROSADDLER generates data that effectively resolves user failures without accessing any private information.

1 INTRODUCTION

Language-model agents increasingly operate over persistent user state. They send messages, modify calendars, search workspaces, and coordinate actions across applications. Their reliability depends not only on the underlying model but also on the *harness* that supplies instructions, exposes tools, manages context, and controls execution. Harness optimization treats these external components as editable objects and uses task executions and feedback to retain useful changes (Lee et al., 2026; Park et al., 2026). An optimizer, however, can only repair the weaknesses that its own task suite exposes. The failures that matter for a deployed agent are the ones its users actually run into, and those are by construction the ones the developer's tasks did not anticipate.

Deployment leaves evidence of such failures, but only a projection of it. What reaches the developer is the request, the user's report of what went wrong, and the agent's tool calls. What does not is the environment those calls ran against, such as the mailbox, contact list, calendar, and documents the agent actually read, since this is precisely the part that privacy policy protects. This gap would be harmless if a failure were a property of the request alone, but it is not. Consider a request to email the attendees of a meeting (Figure 1). If the requester is on the attendee list, the correct behavior is to leave them out because they already know the details, and an agent that includes them fails. If the requester is not on the list, the identical request carries no such trap and the agent succeeds. The same request, harness, and model thus produce opposite outcomes, decided by state the developer cannot inspect. An error label on the request therefore under-determines the repair. It does not say which environment condition the harness failed to handle, and a harness edited against an unidentified condition is as likely to regress as to improve.

We therefore study *failure reproduction for harness optimization*: given privacy-bounded evidence of a user failure, construct a fresh, executable scenario in which the same seed harness exhibits the same specified behavioral discrepancy. The reconstructed scenario contains no user data. Its environment is

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

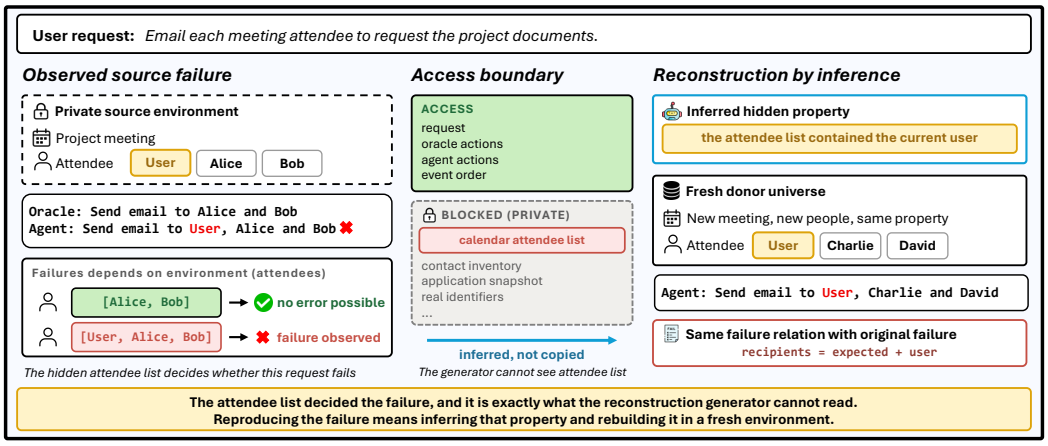


Figure 1: Failure lives in the environment, not in the request. The identical request to email the attendees of a meeting succeeds or fails depending on attendee list, which is private environment we cannot access. The generator observes only the request, the user's report of what went wrong, and the agent's tool calls, while other environments remain private.

assembled from public donor material and is accepted only when it recreates the reported discrepancy, which makes it a testable hypothesis about the hidden condition rather than a guess about the user's data. Reproduction is more demanding than generating a difficult task. The agent could fail because the generated task is impossible, an identifier is invalid, the oracle is inconsistent, or an unrelated failure occurs earlier. Moreover, reproducing the right local error inside a much simpler micro-task removes the workflow context needed to learn a useful repair. Both *failure fidelity* and *workflow fidelity* therefore matter.

REPROSADDLER fixes the target before generation. From the observable evidence, it records the operation the agent should have performed and the one it performed, stated over roles such as the requester rather than real names, plus a value-free workflow blueprint when available. The generator cannot edit this target, so it cannot drift toward the easiest failure. It then rebuilds the workflow in a public environment and adds switchable conditions, each a candidate explanation of the hidden condition, such as placing the requester on the attendee list. With all conditions off, the seed harness must succeed, so a failure with conditions on comes from the conditions rather than a broken task. Trace-guided revision and ablation keep only the conditions needed to make the specified discrepancy the first failure.

Our evaluation on GAIA2 and Workspace-Bench holds the harness optimizer fixed and varies only the data it optimizes on. We compare generated reconstructions with the unmodified seed, random tasks, unrelated failures, observable-only evidence, and the privileged original source tasks. The main question is whether making the evidence executable adds value beyond reading it.

In summary, our main contributions are as follows:

1. We identify a gap between deployed agents and their improvement: the failures worth fixing are tied to user environments that cannot be seen or copied, so the evidence that does reach the developer is not something a harness optimizer can act on.
2. From evidence containing no user data, REPROSADDLER constructs a new environment where the reported failure happens again, and accepts it only when the reconstructed conditions demonstrably cause that failure.
3. We show these reconstructions are useful. Optimizing a harness on them repairs the original user failures, making reproduction a testable claim rather than an assumption about good training data.

2 RELATED WORK

Prompt, system, and harness optimization. Automatic prompt engineering searches over instructions using generated candidates, textual feedback, and evolutionary histories (Zhou et al., 2023;

Pryzant et al., 2023; Agrawal et al., 2026). DSPy optimizes declarative language-model programs, and agent-system search extends optimization to functions and workflows (Khattab et al., 2024; Zhang et al., 2024; 2025a; Hu et al., 2025). SWE-agent illustrates the importance of an agent–computer interface (Yang et al., 2024), and Meta-Harness and AutoSaddler make harness components explicit optimization targets (Lee et al., 2026; Park et al., 2026). All of these assume a set of tasks the optimizer may execute at will. We address the step before that. When the failure to be repaired occurred in a user environment that cannot be accessed, the question is how the optimizer obtains an executable task at all.

Learning from experience without weight updates. Reflexion retains verbal feedback, ExpeL extracts reusable knowledge from experience, and Voyager accumulates executable skills (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024). ReasoningBank studies experience-based reasoning memory, the Darwin Gödel Machine evolves agent code, and surveys place these systems within the broader self-evolving agent landscape (Ouyang et al., 2026; Zhang et al., 2026b; Gao et al., 2026). What they retain is a summary, a piece of knowledge, or a patch, none of which can be executed again as the original experience. REPROSADDLER retains a reconstructed scenario together with the evidence that it reproduces the target failure, so another optimizer can re-run the failure itself without the private source snapshot.

Failure reproduction under an information boundary. Reproducing a failure from partial field evidence is a long-standing problem in software engineering. Prior work synthesizes executable reproductions from bug reports, crash traces, and lightweight field records (Jin & Orso, 2012; Kang et al., 2023; Soltani et al., 2017; Mündler et al., 2024), and a related line reproduces field failures while deliberately withholding the user data that triggered them (Castro et al., 2008; Clause & Orso, 2011). Our setting shares this structure but differs in two ways. What must be reproduced is an interaction between an agent and a stateful environment rather than a deterministic program state, and the purpose is to produce optimization data for a harness rather than to confirm a fix. Work on de-identification and privacy-preserving synthetic data (Yue et al., 2023; Stadler et al., 2022) builds releasable data without exposing source records, typically by preserving distributional utility. We preserve a single named failure relation instead of a distribution, which gives us an executable acceptance criterion: the reconstruction either reproduces the relation or it is rejected.

Failure attribution and intervention-driven debugging. Failure-attribution methods ask which component or step caused an unsuccessful trajectory (Zhang et al., 2025b). AgenTracer investigates agentic failure attribution, and DoVer uses interventions for debugging multi-agent systems (Zhang et al., 2026a; Ma et al., 2026). These methods assume the failing trajectory and its environment are in hand and localize the cause inside them. We start without that environment and build a new one in which the same discrepancy occurs again. Interventions therefore serve to test candidate conditions inside the generated scenario, not to recover the private cause of the original failure.

Synthetic data, curricula, and environment design. Self-Instruct demonstrates the value of model-generated instruction data (Wang et al., 2023). Automated curriculum learning chooses training experience according to changing learning needs (Graves et al., 2017), and unsupervised environment design constructs solvable environments that challenge a learner (Dennis et al., 2020). Their selection criteria, such as diversity, difficulty, and regret, are properties of the task itself. Ours is external. An accepted scenario must reproduce one particular source failure relation and must keep the workflow in which that failure arose. A reduced task that preserves the local error while removing the workflow is locally faithful and still useless, because it no longer carries the context a repair has to generalize over.

Environment-grounded agent benchmarks. AgentBench, WorkArena, and τ-bench evaluate agents in interactive environments with tool use and state-dependent objectives (Liu et al., 2024; Drouin et al., 2024; Yao et al., 2025). GAIA2 emphasizes dynamic and asynchronous environments, and Workspace-Bench provides large heterogeneous workspaces with file dependencies (Froger et al., 2026; Tang et al., 2026). We use these environments as the supply of donor material and keep each benchmark's native harness and evaluator. Whether a reconstruction counts is decided by the fixed failure relation it must re-trigger, not by the fidelity of the environment it runs in.

3 PRELIMINARIES AND PROBLEM SETTING

Agentic Scenario and Harness. We consider agentic tasks that require multi-step reasoning, repeated tool use, and interaction with an application environment that already holds the user's prior data. A scenario is a pair $x = (u, E)$ consisting of a user request u and an initial environment state E, together with a reference specification y^* against which the outcome is judged. Following Park et al. (2026), we write the harness as H_θ with

$$\theta = (\theta_{\text{prompt}}, \theta_{\text{tool}}, \theta_{\text{middleware}}) \in \Theta, \tag{1}$$

and keep the model weights fixed throughout. Executing the agent induces $(\tau, \hat{y}) \sim P_\theta(\cdot \mid x)$, where τ is the execution trace and $\hat{y}$ the final output, and a task-level metric $\mu(\hat{y}, y^*) \in [0, 1]$ scores the result. For a finite set D, we obtain the empirical harness performance as

$$\widehat{J}_D(\theta) = \frac{1}{|D|} \sum_{(x_i, y_i^*) \in D} \frac{1}{R} \sum_{r=1}^{R} \mu(\hat{y}_{i,r}, y_i^*), \qquad (\tau_{i,r}, \hat{y}_{i,r}) \sim P_\theta(\cdot \mid x_i). \tag{2}$$

where R is the number of repeated executions per scenario, averaged to account for the stochasticity of agent execution. We denote the frozen seed harness by θ_0. We group environments into universes, disjoint sets of environments that share no users or stored data.

Harness Optimization from Execution Records. Running H_θ on a set of scenarios D yields execution records $\mathcal{E}(D) = \{(x, y^*, \tau, \hat{y})\}$, and a harness optimizer $\mathcal{O}$ turns them into an updated harness,

$$\theta' \sim \mathcal{O}\big(\theta, \mathcal{E}(D), D_{\text{dev}}\big), \tag{3}$$

where D_{dev} is a fixed development set, disjoint from the source failures and from the final evaluation set, that the optimizer uses for regression and generalization checks when it decides which candidate harness to retain. We hold $\mathcal{O}$ fixed in this work, so D is the only thing that varies. What we study is where D comes from when the failures to be repaired occurred in environments we cannot access.

Evidence Projection and the Privacy Boundary. A deployed failure case i arose in a private scenario $x_i^{\text{src}} = (u_i^{\text{src}}, E_i^{\text{src}})$ executed under the seed harness, leaving a trace τ_i^{src}. A trusted projector Π produces the evidence

$$\rho_i = \Pi\big(\tau_i^{\text{src}}\big) = (O_i, A_i, B_i), \tag{4}$$

where O_i and A_i are de-identified action structures for what the oracle required and what the agent did, and B_i is an optional value-free blueprint of the workflow. The projection keeps requests, actions, and their order, and drops application state, raw tool returns, and identifiers. The privacy boundary of this work is

$$E_i^{\text{src}} \notin \text{range}(\Pi), \tag{5}$$

so the environment never appears in the evidence. Entities appear as roles, and values needed only for equality tests appear as opaque constraints (Appendix C).

Observed Failure Relation. The observed failure is a typed discrepancy fixed before generation begins,

$$F_i = \Delta(O_i, A_i). \tag{6}$$

For instance, $A_{\text{recipients}} = O_{\text{recipients}} \cup \{\texttt{current_user}\}$ specifies an extra-recipient error rather than a broad "email failure." For any trace τ we write $M_{F_i}(\tau) \in \{0, 1\}$ for whether that trace exhibits F_i.

Failure Reproduction. A reconstruction of case i is a new scenario $x_i^{\text{gen}} = (u_i^{\text{gen}}, E_i^{\text{gen}})$ with oracle $y_i^{*\text{gen}}$, assembled entirely from public donor material rather than from E_i^{src}. It is a faithful reproduction under two requirements. *Failure fidelity* asks that executing the seed harness on it produce a trace with $M_{F_i}(\tau) = 1$ as defined above. *Workflow fidelity* asks that the agent actually carry out the workflow recorded in B_i instead of reaching the same error by a shorter route. A scenario that triggers the target error inside a one-step task would still match F_i, but it drops the surrounding steps in which the failure originally arose, and a harness repair learned there need not transfer back to the real case. We write D_{gen} for the set of reconstructions that pass acceptance and provide the acceptance procedure in Section 4.

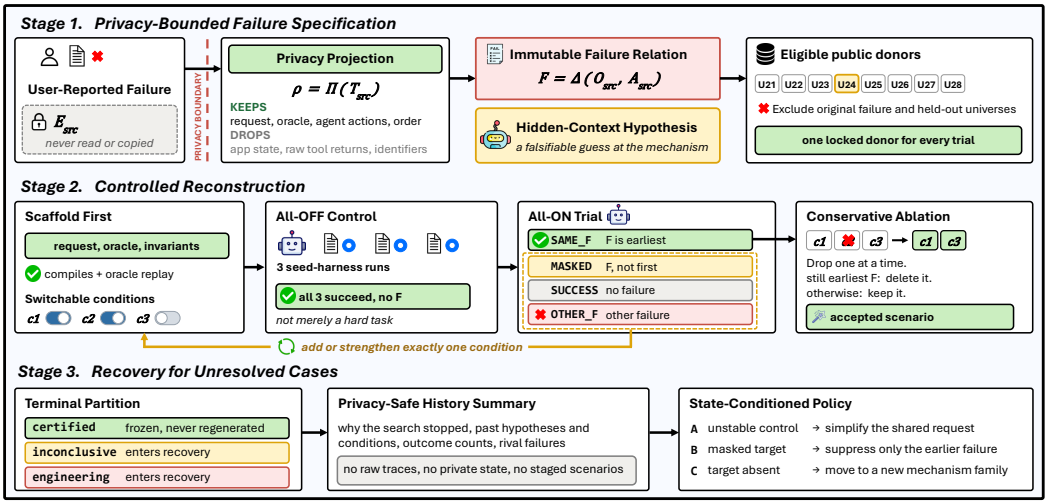


Figure 2: Overview of REPROSADDLER. Stage 1 fixes the failure specification and locks a public donor. Stage 2 constructs a valid scaffold, tests switchable conditions against an all-OFF control, and refines or reduces them using execution feedback. Stage 3 recovers unresolved cases using search history while preserving the acceptance criteria.

Problem Statement. Given evidence $\{\rho_i\}$ subject to Eq. 5, construct a set of reconstructions D_{gen} satisfying failure and workflow fidelity, such that the resulting harness

$$\theta_{\text{gen}} \sim \mathcal{O}\big(\theta_0, \mathcal{E}(D_{\text{gen}}), D_{\text{dev}}\big) \tag{7}$$

maximizes $\widehat{J}_{D_{\text{src}}}(\theta_{\text{gen}})$, where D_{src} is the prespecified evaluation set of original failures, fixed in advance and not restricted to the cases on which reconstruction succeeded.

4 PROPOSED METHOD: REPROSADDLER

REPROSADDLER builds an executable reproduction of a reported failure in a public donor environment in three stages (Figure 2). It fixes the target failure (Section 4.1), constructs and tests a reproduction (Section 4.2), and recovers unresolved cases under the same acceptance rule (Section 4.3). The accepted scenarios form D_{gen} for downstream harness optimization. Prompts are listed in Appendix F.

4.1 FIXING THE FAILURE SPECIFICATION

Before anything is generated, we must decide what counts as a successful reproduction. If the target were defined after seeing generated outcomes, the search could drift toward whichever failure is easiest to produce, so this stage fixes the target and selects the environment in which it will be rebuilt. The input is the privacy-bounded evidence $\rho_i = (O_i, A_i, B_i)$ defined in Eq. 4. The compiler fixes the typed failure relation $F_i = \Delta(O_i, A_i)$ and the available workflow constraints in B_i before generation, and neither may be changed to accommodate a generated outcome. For the introductory email example, the oracle excludes the requester and F_i specifies their erroneous inclusion among the recipients. The generator can read this relation, but cannot inspect the source attendee list or any other private application state.

The reconstruction agent proposes a hidden-context hypothesis explaining how F_i could arise. In the example, it hypothesizes that the requester appears on the attendee list and the agent copies the list without applying the required exclusion. This hypothesis guides construction rather than establishing a fact about the source environment. One public donor is selected to support the required operations and roles, excluding the source and final-evaluation universes. That donor remains locked throughout the case's search and recovery, while declared conditions may modify its generated state. The selection rule and evidence representation are detailed in Appendix A.1.

4.2 Constructing and Testing a Reproduction

Scaffold first, conditions second. The generator first builds a scaffold G_i containing the request, donor-role bindings, oracle specification, and invariants needed to preserve the workflow. The scaffold must compile and its reference solution must pass oracle replay. Only then are switchable conditions attached. These conditions may modify task wording, donor state, or both, but cannot remove a required oracle obligation, encode the answer, or instruct the agent to fail. Adding the requester to a donor meeting's attendee list is one such condition.

Let C_i be the current condition set and $S \subseteq C_i$ the enabled subset. The compiler materializes

$$x_i[S] = \mathrm{Compile}\big(E_i^{\mathrm{donor}}, G_i, C_i, S\big). \tag{8}$$

All-OFF, $S = \emptyset$, retains the full scaffold workflow with the conditions disabled, whereas all-ON enables the complete set. The associated oracle and failure matcher share the same role bindings. Every tested configuration must satisfy the mechanical, invariant, and oracle-replay checks denoted by $V(x) = 1$, which are described in Appendix A.2.

Controlled reproduction. Each behavioral trial runs the frozen seed harness $R = 3$ times from reset copies of the compiled initial state. The all-OFF control passes, $\mathrm{Base}_i = 1$, only when all runs complete the required workflow successfully without any detected failure. This ensures that the unmodified scaffold was solvable by the seed harness in the observed runs. Only after this gate passes do we interpret the all-ON results.

Let $\Phi_i(\tau)$ be the ordered sequence of detected failure relations, and let $M_{F_i}^{\mathrm{first}}(\tau) = 1$ indicate that its first element matches F_i. We also write $W_i(\tau) = 1$ when the trace preserves the workflow constraints available in B_i, with F_i at the target decision as the only permitted deviation. Checking the executed workflow prevents a shorter task from supplying a spurious reproduction. A candidate is admitted when

$$\begin{aligned} \mathrm{Accept}_i = 1 \iff\ & \mathrm{Base}_i = 1 \wedge V\big(x_i[C_i]\big) = 1 \\ & \wedge \exists r \in \{1, \ldots, R\} : \quad W_i(\tau_r^{\mathrm{on}}) = 1 \wedge M_{F_i}^{\mathrm{first}}(\tau_r^{\mathrm{on}}) = 1. \end{aligned} \tag{9}$$

Thus, all three control runs must succeed, while at least one all-ON run must reproduce the target as its earliest detected failure. If another failure precedes the target, that run is treated as masked: the earlier error may have altered the subsequent execution. This rule supplies an operational reproduction criterion, without claiming a guaranteed failure probability or identifying the private source cause.

Refinement and ablation. A valid all-ON batch is classified as `SAME-F`, `MASKED`, `SUCCESS`, or `OTHER-F`, in that precedence order (Appendix A.3). Only `SAME-F` satisfies the reproduction requirement. Otherwise, the controller compares generated traces with the projected source actions around the target decision and uses one observed discrepancy to justify revising exactly one condition. Successful or off-target trials motivate adding or strengthening a condition, whereas masked trials motivate a condition edit that suppresses the specific preceding failure. Every revision is recompiled and tested through both gates.

Once a set reproduces F_i, conservative ablation tests one-condition deletions under the same protocol. A condition is deleted only if the reduced set still qualifies as `SAME-F`; otherwise it is retained. Deletions that produce only successful runs support necessity within the tested set, whereas deletions that introduce another failure leave necessity unresolved. The final accepted scenario and oracle enter D_{gen} with their reproduction evidence and ablation status. Detailed revision appear in Appendix A.4.

4.3 Recovering Unresolved Cases

A bounded search does not reproduce every case, and an unresolved case does not mean that no reproducible mechanism exists. The primary search partitions cases into certified, inconclusive, and engineering-failed outcomes. Certified cases are frozen, and recovery revisits only the rest with a larger budget. It uses a summary of previous hypotheses, conditions, outcome counts, and stopping reasons rather than raw traces or staged scenarios, and chooses its next step by why the search stopped. An unstable control prompts simplification of the shared request, a masked target prompts suppression of the preceding failure, and an absent target prompts a different mechanism family. The

Table 1: Performance of the six arms and two ablations on target failures and a held-out universe, as mean ± std over three runs.

	GAIA2		Workspace-Bench	
Arm / setting	Target failure	Held-out universe	Target failure	Held-out universe
Seed	26.67 ± 3.82	55.33 ± 2.31	43.81 ± 3.14	39.96 ± 4.06
Random	35.00 ± 2.50	60.67 ± 1.15	44.65 ± 1.18	46.66 ± 2.30
Other-F	33.33 ± 2.89	56.67 ± 1.15	53.08 ± 4.28	38.14 ± 1.24
Original	37.50 ± 5.00	55.33 ± 1.15	52.20 ± 4.49	48.11 ± 4.82
Observable	32.50 ± 5.00	57.33 ± 4.16	45.01 ± 2.54	41.81 ± 2.95
REPROSADDLER	36.67 ± 1.44	58.67 ± 1.15	56.19 ± 0.73	49.85 ± 1.69
w/o condition reduction (All-ON)	32.50 ± 2.50	-	51.30 ± 1.85	-
w/o failure-guided refinement (Unguided)	33.33 ± 3.82	-	46.25 ± 1.29	-

donor, target, behavioral repetitions, and acceptance rule stay unchanged. Recovered scenarios are pooled into D_{gen}, and primary and recovery yields are reported separately (Appendix A.5).

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Benchmarks and base harness systems. We use **GAIA2** (Froger et al., 2026), which tests general-purpose assistant capabilities in a simulated smartphone environment spanning 10 distinct user environments referred to as *universes*, and **Workspace-Bench** (Tang et al., 2026), which tests file-based tasks with cross-document dependencies. We retain each benchmark's native harness and evaluator. Both settings require agents to act on persistent environment state beyond the user request. GAIA2 provides ten native universes, and we split Workspace-Bench into six universes of tasks that share stored inputs (Appendix B.1). The source failures are catalogued in Appendix D.

Six-arm comparison. We compare our proposed method with five other arms, which differ only in the information used for harness optimization. **Seed** applies no optimization and evaluates the unmodified harness. **Random** optimizes on executable tasks sampled independently of failure outcomes. **Other-F** optimizes on executable failures whose signatures lie outside the target set. **Original** optimizes on the original source-failure tasks themselves, and it is an upper bound privileged reference with access to source tasks including their private environment. **Observable** receives the projected failure evidence without reproduced environments. REPROSADDLER optimizes only on the accepted executable reconstructions, produced by the proposed method. Engineering failures that prevent a valid reproduction attempt are analyzed in Appendix E.

Optimization and evaluation. With GPT-5.5 weights fixed, the executable arms use AutoSaddler (Park et al., 2026) to revise harness components from training executions; *Observable* instead uses projected evidence and development feedback. Tasks and evaluators are not editable. Each arm selects a candidate on a separate development set and freezes it before testing. The primary endpoint measures repair on the original source failures, including cases without an accepted reconstruction. Held-out universe evaluation tests transfer using the same frozen harness without further tuning. We report avg@3, the task-averaged native score over three executions: binary success for GAIA2 and rubric credit for Workspace-Bench. Split assignments, budgets, and protocol-specific differences are provided in Appendix B.

5.2 MAIN RESULTS

Table 1 shows that REPROSADDLER achieves the best target-failure performance among all arms that do not access private information, on both benchmarks. On GAIA2, it improves the seed harness by 10.0 points, within one point of Original (10.8 points), even though Original trains directly on the source tasks together with their private environments. On Workspace-Bench, REPROSADDLER improves the seed by 12.4 points and matches or exceeds Original. Reconstructing each failure in a public environment therefore recovers most of the repair signal that the private source tasks provide.

REPROSADDLER is also more stable than Original on the target failures, with a lower standard deviation across the three evaluation runs on both benchmarks.

Gains on the target failures would be of limited value if they came from patches tailored to those specific tasks, and the held-out universe tests this. REPROSADDLER also improves held-out performance on both benchmarks, by 3.3 points on GAIA2 and by 9.9 points on Workspace-Bench, where it is the best of all arms. This suggests that the reconstructed scenarios lead the optimizer toward general harness changes rather than fixes that apply only to the targeted tasks. In contrast, Original does not improve GAIA2 held-out performance at all despite its strong target-failure score, which suggests that training on the exact source tasks can produce patches specific to them. Because REPROSADDLER rebuilds each failure in a different donor environment with fresh entities, a patch that relies on task-specific details is unlikely to help across reconstructions, and the optimizer is pushed toward the condition that causes the failure.

The remaining arms clarify where this benefit comes from. Random improves held-out performance, most notably on GAIA2 where it is the strongest arm, but its target-failure gains are smaller and nearly vanish on Workspace-Bench. Generic task exposure thus improves the harness broadly without reliably fixing the failures that users actually encountered. Other-F shows the opposite pattern on Workspace-Bench, raising target-failure performance by 9.3 points while lowering held-out performance below the seed. Training on unrelated failures appears to produce fixes that partly overlap with the target failures but do not transfer, which is consistent with overfitting to the failures seen during training. Observable falls below REPROSADDLER on every endpoint. With only the failure description and no executable environment, the optimizer cannot test whether a candidate patch handles the condition behind the failure, and it tends to address the surface of the reported error instead. Together, these comparisons show that the improvement of REPROSADDLER comes neither from generic task exposure nor from exposure to failures in general, but from executable reproductions of the specific failures that users encountered.

Ablation study. The lower block of Table 1 shows that both components are effective. **All-ON** trains on the complete condition set of a target-reproducing candidate and skips conservative ablation. Since conditions accumulate over refinements, this set often includes unneeded conditions that add difficulty unrelated to the target, so the optimizer cannot tell which pressure the harness must handle. **Unguided** revises conditions without comparing the generated trace with the source actions around the target decision. It then cannot tell how the agent avoided the failure, so the search changes conditions without direction.

5.3 FURTHER ANALYSIS

RQ1: Does REPROSADDLER repair the failures it targets? We partition the failures into *linked* cases that accepted reconstructions from REPROSADDLER, and *other* cases, for which no reconstruction was accepted. Figure 3 reports each method's improvement over Seed on these two groups. The gains from REPROSADDLER are concentrated on linked failures, whereas others show more evenly distributed gains across the two groups. This pattern suggests that REPROSADDLER improves the targeted failures effectively, rather than a uniform benefit from additional optimization.

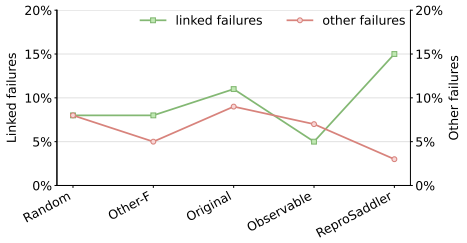


Figure 3: Target-failure performance gains over *Seed* on GAIA2: linked vs. other failures.

RQ2: How close is REPROSADDLER to *Original* without private access? REPROSADDLER never reads the private environments included in *Original* source tasks. We compare the two arms at the level of the training data and at the level of the repaired cases. Table 2 compares the number of actions per scenario. The reconstructions from REPROSADDLER follow a the number of actions close to that of the original tasks, so

Table 2: Average agent action and event counts of GAIA2.

Arm	Agent actions	All events
Random	6.54	9.30
Original	7.40	9.93
Other-F	6.10	9.14
REPROSADDLER	6.91	7.90

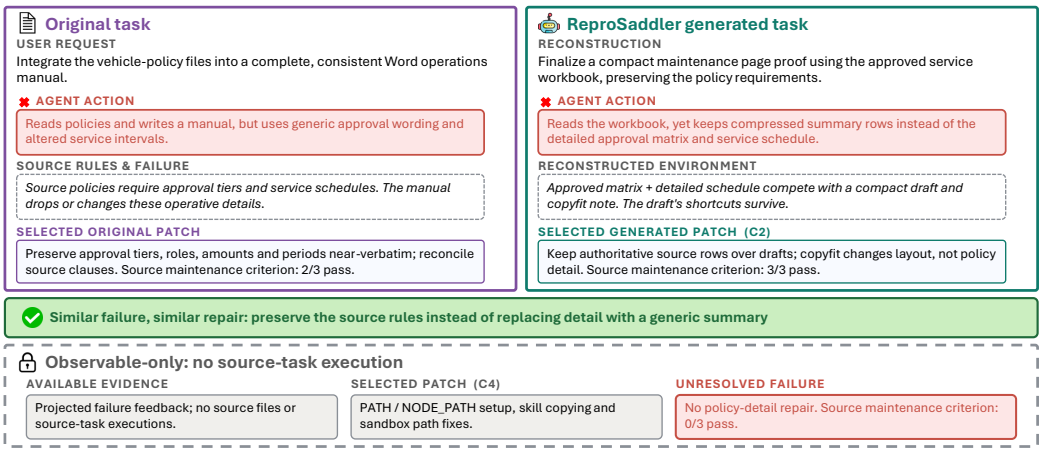


Figure 5: Workspace-Bench case study (task 86). REPROSADDLER recreates policy-detail omissions using competing authoritative tables and compact drafts. Original- and Generated-trained patches both preserve source details, whereas the selected Observable-only patch changes runtime configuration without addressing the content error.

they preserve the length of the source workflows instead of collapsing into short tasks that trigger only the local error.

We then compare repair outcomes over *Seed* in Figure 4. Jaccard measures the shared improvements relative to the union of two repair sets, whereas coverage measures the fraction of Original's repair set also improved by another method. REPROSADDLER achieves the highest alignment with *Original* among the compared alternatives, with 53.3% Jaccard and 72.7% coverage. These results suggest that REPROSADDLER improves many of the same cases as *Original*, yielding similar case-level repair outcomes even though it is built without access to the source environment.

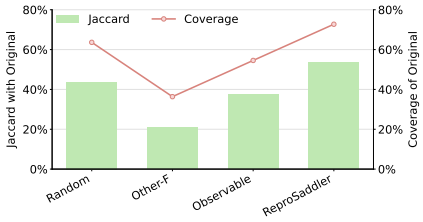


Figure 4: Target-failure repair-set overlap with Original on GAIA2.

RQ3: Does reconstructing the environment matter? Figure 5 shows a Workspace-Bench failure in which the agent integrates vehicle-policy files into an operations manual but replaces required approval rules and service schedules with incomplete or altered descriptions. The *Observable* evidence identifies the lost obligations but not the hidden document context behind them. REPROSADDLER makes one explanation executable: an approved workbook with detailed tables next to a compact page proof, where the agent still keeps compressed summary rows. The *Original*- and REPROSADDLER-trained patches both tell the agent to keep authoritative source details, whereas the *Observable* patch only adjusts runtime paths and skill loading. On the original maintenance criterion, they pass 2/3, 3/3, and 0/3 evaluations, respectively. An executable reconstruction can thus support a targeted repair even when the original hidden cause remains uncertain.

6 CONCLUSION

We studied how to improve a deployed agent from failures whose environment cannot be seen or copied. REPROSADDLER fixes the observed discrepancy before generation, rebuilds the workflow in a public donor environment, and accepts a scenario only when switchable conditions make the seed harness exhibit that discrepancy against a control in which it succeeds. On GAIA2 and Workspace-Bench, harnesses optimized on these reconstructions repair the source failures, approach optimization on the privileged original tasks, and fix largely the same cases. Optimizing on the same evidence without reconstruction yields clearly smaller gains, showing that the hidden environment condition, not the failure description alone, is what makes a user failure useful for repair.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing methods and polishing the writing of the paper with translation. Specifically, AI coding assistants were used to help write and debug parts of the experimental code, and AI tools were used to translate draft text written in the authors' native language into English. We have not used generative AI tools to propose or refine hypotheses, design the methodology or experiments, or interpret results, and tasks involving mathematical claims, proofs, dataset cleaning, and qualitative data analysis are not applicable to this work. We have reviewed all AI-assisted work. AI-assisted code was reviewed by the authors and validated by executing it on the benchmark environments, including the validity and control checks of the reconstruction pipeline, before any reported experiment. Translated and polished text was checked by the authors against the intended meaning and our experimental records, so that no claim, number, or citation was altered or introduced. Separately, large language models appear in this paper as components of the studied system, namely the reconstruction agent, the harness optimizer, the task agents, and the benchmark judges. These components are part of the research method rather than research assistance, and their models and settings are reported in Appendix B.2 and Appendix B.4. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, and no real user data was collected or used. All source failures come from public benchmarks, GAIA2 and Workspace-Bench, and executions used benchmark environments rather than any user's live environment. The method is motivated by the goal of improving agent harnesses without exposing private user state, and the reconstruction pipeline is designed so that the generator never reads the source environment, raw tool returns, or source identifiers.

REPRODUCIBILITY STATEMENT

Section 3 defines the problem setting and notation, and Section 4 describes the full reconstruction procedure. Appendix A specifies the evidence representation and donor ranking, scaffold and condition compilation, the all-OFF and all-ON acceptance criteria, the refinement and ablation rules, and the search budgets of the primary and recovery procedures. Appendix B reports the benchmark splits, the models, and optimizer settings and final evaluation metrics. Appendix C details the information boundary, matching semantics, and failure accounting, and Appendix F lists the prompt templates.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Miguel Castro, Manuel Costa, and Jean-Philippe Martin. Better bug reporting with better privacy. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 319–328, 2008. doi: 10.1145/1346281.1346322.

James Clause and Alessandro Orso. Camouflage: automated anonymization of field data. In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 21–30, 2011.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart J. Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13049–13061, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable

are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 2024. URL https://proceedings.mlr.press/v235/drouin24a.html.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=9gw03JpKK4.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=CTr3bovS5F. Survey Certification.

Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1311–1320. PMLR, 2017. URL http://proceedings.mlr.press/v70/graves17a.html.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Wei Jin and Alessandro Orso. BugRedux: Reproducing field failures for in-house debugging. In *2012 34th international conference on software engineering (ICSE)*, pp. 474–484. IEEE, 2012.

Sungmin Kang, Juyeon Yoon, and Shin Yoo. Large language models are few-shot testers: Exploring LLM-based general bug reproduction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 2312–2323. IEEE, 2023.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. In *Conference on Language Modeling (COLM)*, 2026. URL https://arxiv.org/abs/2603.28052.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mrEK16Jy6h.

Niels Mündler, Mark N Müller, Jingxuan He, and Martin Vechev. SWT-Bench: Testing and validating real-world bug-fixes with code agents. *Advances in Neural Information Processing Systems*, 37: 81857–81887, 2024.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Sungho Park, Wonjoong Kim, Rongyuan Tan, Jue Zhang, Wook-Shin Han, Pengfei Gao, Chanyoung Park, Yongqiang Yao, Rao Fu, Elsie Nallipogu, et al. AutoSaddler: Automatic harness optimization with durable updates from agent execution traces. *arXiv preprint arXiv:2608.23041*, 2026.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 7957–7968. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.494. URL https://doi.org/10.18653/v1/2023.emnlp-main.494.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mozhan Soltani, Annibale Panichella, and Arie van Deursen. A guided genetic algorithm for automated crash reproduction. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pp. 209–220. IEEE, 2017.

Theresa Stadler, Bristena Oprisanu, and Carmela Troncoso. Synthetic data–anonymisation groundhog day. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 1451–1468, 2022.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, Shaoli Yu, Jinqi Wang, Zihang Zhou, Hongyi Zhou, Yuting Lv, Jinyang Li, Jiashuo Liu, Ruoyu Chen, Chunwei Liu, GuoLiang Li, Jihua Kang, and Fan Wu. Workspace-Bench 1.0: Benchmarking AI agents on workspace tasks with large-scale file dependencies, 2026. URL https://arxiv.org/abs/2605.03596.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, pp. 50528–50652, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Xiang Yue, Huseyin Inan, Xuechen Li, Girish Kumar, Julia McAnallen, Hoda Shajari, Huan Sun, David Levitan, and Robert Sim. Synthetic text generation with differential privacy: A simple and practical recipe. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1321–1342, 2023.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=l05DseqvuD.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=pUpzQZTvGY.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=z5uVAKwmjf.

Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. Offline training of language model agents with functions as learnable weights. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 60315–60335. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/zhang24cd.html.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599. PMLR / OpenReview.net, 2025b. URL https://proceedings.mlr.press/v267/zhang25cq.html.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pp. 19632–19642. AAAI Press, 2024. doi: 10.1609/AAAI.V38I17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

A Reconstruction Protocol Details

This appendix specifies the evidence, compilation, evaluation, and search procedures accompanying Section 4.

A.1 Evidence and Donor Selection

Evidence representation. The trusted projection retains the de-identified request, oracle and agent action structures, and their order. Raw application state, tool returns, and source identifiers are excluded; roles and opaque equality constraints carry the relational information needed for matching. The method assumes that the reported evidence or an available reference specification determines O_i; missing oracle obligations cannot be recovered by reading E_i^{src}. This access boundary does not imply that the retained relations disclose no information about the source case.

The fixed relation F_i can describe an omitted operation, an extra operation, a wrong argument value, a missing or extra set member, or a shifted temporal value. The available blueprint B_i represents required agent and environment events using opaque tokens with operation names, argument names, dependencies, relative timing, and global positions. The matcher W_i compares generated events with these constraints through the donor-role binding. The target decision is allowed to exhibit F_i rather than being required to match the correct oracle action. If the blueprint is partial or absent, workflow claims are limited to the constraints supported by the evidence.

Hypothesis families. Candidate mechanisms include workload or control-flow pressure, ambiguous entity or value binding, cross-application context, premature termination, confusion between similar donor-native records, and representation or temporal handoff errors. A hypothesis must explain how the discrepancy could arise rather than merely restate F_i. Failure of a particular construction to reproduce F_i is evidence about that construction, not a refutation of every mechanism in its family.

Donor ranking. For abstract requirements r_i extracted from O_i and B_i, the public donor candidates are ordered lexicographically by

$$\mathrm{rank}_i(E) = \big(n_{\mathrm{edit}}(E, r_i),\ n_{\mathrm{unmet}}(E, r_i),\ -\mathrm{sim}(E, r_i),\ \mathrm{tie}(E)\big). \tag{10}$$

The entries represent required edits, unmet requirements, similarity to r_i, and a deterministic tie-breaker. The smallest tuple selects the donor, which is locked for all hypotheses, revisions, ablations, and recovery trials of the case. Required roles reuse donor-native entities where possible; fresh entities are created only when no suitable entity is available. This ranking selects construction material, while the validity gates determine whether a constructed scenario is executable.

A.2 Scaffolds and Condition Compilation

The scaffold is $G_i = (u_i^0, \beta_i, y_i^{*\mathrm{gen}}, \mathcal{I}_i)$, with a base request u_i^0, a donor-role binding β_i, an oracle specification, and invariants $\mathcal{I}_i$. Deterministic normalization resolves schema defaults, contact and resource references, donor bindings, and application-specific naming. It may repair donor compatibility without changing F_i, the required workflow, or the hypothesis's condition semantics. Required environment events are materialized at runtime, preserving dependencies on resources created during execution rather than pre-populating substitutes.

Each condition records a pre-execution predicate, enabled and disabled forms, declared difference paths, and invariants. Compilation must produce the declared executable differences. Conditions may introduce bounded precursor work or plausible competing records, but may not remove required oracle obligations, reveal an answer, or direct the agent to fail. Their disabled forms return the scaffold without shortening its required workflow.

For enabled subset S, let $y_i^{*\mathrm{gen}}[S]$ denote the oracle associated with $x_i^{(t)}[S]$. Its concrete bindings can change with the compiled state, but the required obligations and the target relation remain fixed. The request, environment, oracle, and matcher are derived from the same typed design. $V(x) = 1$ requires mechanical validity, satisfaction of the declared invariants, and acceptance of reference-solution replay. These checks apply to each tested configuration, including all-ON and ablated variants, because scaffold validity alone cannot rule out defects introduced by a condition.

A.3 Behavioral Evaluation

All-OFF certificate. Each run uses the frozen seed harness and a reset copy of the relevant initial state. For $R = 3$ all-OFF repetitions, the control criterion is

$$\begin{aligned}\mathrm{Base}_i^{(t)} = 1 \iff V\big(x_i^{(t)}[\emptyset]\big) = 1 \\ \wedge \bigwedge_{r=1}^{R} \Big[\mu(\hat{y}_r^{\mathrm{off}}, y_i^{*\mathrm{gen}}[\emptyset]) = 1 \wedge W_i(\tau_r^{\mathrm{off}}) = 1 \wedge \Phi_i(\tau_r^{\mathrm{off}}) = \varnothing\Big].\end{aligned} \tag{11}$$

Malformed scenarios and infrastructure-interrupted rollouts are engineering issues and do not supply valid behavioral evidence. They are retried within the engineering budget. An executable but unstable control may be repaired by simplifying the shared request without changing the workflow, oracle obligations, donor, hypothesis, or condition semantics. This changes the scaffold and invalidates previous control results. If the repair budget is exhausted without a passing control, the hypothesis is abandoned.

Failure ordering. $\Phi_i(\tau)$ contains the relations detected against the applicable oracle and workflow constraints, including competing failures. Its ordering convention must remain fixed across primary search, ablation, and recovery. In particular, an omission must be assigned to an obligation deadline or a terminal check, since the missing action has no execution timestamp. Simultaneous discrepancies also require a fixed ordering convention. Thus "earliest" refers to the detector's ordering, not an independently established causal ordering.

Outcome precedence. For a valid all-ON batch $\mathcal{T} = \{\tau_r\}_{r=1}^{R}$, define

$$\mathrm{Outcome}_i(\mathcal{T}) = \begin{cases} \texttt{SAME-F}, & \exists r:\ W_i(\tau_r) = 1 \wedge M_{F_i}^{\mathrm{first}}(\tau_r) = 1, \\ \texttt{MASKED}, & \exists r:\ W_i(\tau_r) = 1 \wedge F_i \in \Phi_i(\tau_r) \wedge M_{F_i}^{\mathrm{first}}(\tau_r) = 0, \\ \texttt{SUCCESS}, & \forall r:\ \mu(\hat{y}_r, y_i^{*\mathrm{gen}}[C_i^{(t)}]) = 1 \wedge W_i(\tau_r) = 1 \wedge \Phi_i(\tau_r) = \varnothing, \\ \texttt{OTHER-F}, & \text{otherwise.} \end{cases} \tag{12}$$

The branches are evaluated in this order. A workflow-violating trace cannot supply a SAME-F witness even if it exhibits the local target error. An invalid or interrupted evaluation is handled as an engineering issue rather than forced into a behavioral outcome category.

The all-ON threshold is one admissible witness among three runs, while the all-OFF threshold is three successful runs. These finite, adaptively selected batches support discovery rather than a statistical causal claim. Reproduction counts should accompany certificates to distinguish one-witness and three-witness cases; neither reveals the unique cause of the private failure.

A.4 Refinement and Conservative Ablation

Trace feedback. The controller aligns generated trajectories with the projected source actions and blueprint around the fixed decision operation. Feedback identifies a substituted operation, an absent prerequisite, an argument discrepancy, failure to reach the decision, or a preceding competing relation. No private source state or unprojected source trace is accessed. Each revision must cite one observed discrepancy and change one condition:

$$C_i^{(t+1)} = \begin{cases} C_i^{(t)} \cup \{c_{\mathrm{new}}\}, & \text{add one condition,} \\ (C_i^{(t)} \setminus \{c\}) \cup \{\mathrm{Edit}(c)\}, & \text{modify one condition.} \end{cases} \tag{13}$$

For SUCCESS and OTHER-F, edits add or strengthen the hypothesized trigger. For MASKED, the edit targets the preceding competing failure and may weaken or replace the responsible condition. It preserves F_i, the required workflow, and all other conditions, and cannot add unrelated workload. Whether it also preserves reproduction of F_i is tested again, not assumed. The one-condition rule makes changes inspectable without establishing statistical identifiability of their effects.

Deletion tests and status. Starting from an admitted set, ablation evaluates each one-condition deletion with the same donor, scaffold, validity checks, and behavioral protocol. A SAME-F result

permanently removes the condition and restarts the sweep on the smaller set. A `SUCCESS` result retains it with evidence of necessity under that tested deletion. `MASKED`, `OTHER-F`, and unresolved engineering issues retain it conservatively with necessity unresolved.

If every retained condition's deletion yields `SUCCESS` in a completed sweep, the set is labeled `deletion-minimal` under this finite-run protocol. A sweep with an unresolved deletion is `conservatively-reduced`. If the ablation budget ends before a complete sweep, the last admitted set is retained with incomplete ablation recorded. The labels do not establish global minimality, and unresolved conditions are not claimed to be necessary. Ablation never certifies a new set without a passing reproduction witness.

A.5 Recovery History and Budgets

The primary campaign reaches a terminal partition before recovery begins. An inconclusive case has valid behavioral evidence but no admitted scenario within its search budget. An engineering-failed case lacks valid behavioral evaluation because construction or execution could not be completed. Certified cases are frozen; only the other two groups receive recovery.

The summary σ_i contains the terminal state and stopping reason, target-relation type and decision operation, past hypothesis statements and condition predicates, counts of each outcome type, masking frequency, and the most common competing relations. It excludes raw parent traces, tool outputs, model responses, and staged scenarios. Recovery retains the fixed projected specification and locked donor, and compares detailed trajectories only from its own new executions.

For unstable controls, recovery spends additional attempts simplifying the shared request without changing its obligations or adding a hint that teaches the agent to avoid F_i. For masked targets, it revises one condition to suppress the specific earlier failure. When the target never appeared, it explores a substantively different mechanism family. For example, failure to reach the decision motivates workload or control-flow candidates, an incorrect value at the decision motivates binding or context candidates, and stopping after an intermediate action motivates transition or termination candidates. These are search heuristics rather than diagnoses of the source environment. Exact repetition of a prior hypothesis–condition pair is rejected mechanically; execution evaluates broader semantic changes.

Table 3: Search budgets. Recovery retains the primary acceptance rule.

Budget	Primary	Recovery
Hypotheses per case	5	7
Condition revisions per hypothesis	15	21
Control-stabilization repairs	2	4
Behavioral runs per trial	3	3
Scenario timeout	Unchanged	

Table 3 lists the primary and recovery allocations. Infrastructure restarts are tracked separately from hypothesis and revision budgets. All recovered scenarios must satisfy Eq. 9 without relaxation. Primary and recovery yields and costs remain separate because the recovery policy is adapted to the primary terminal states.

B Detailed Data and Runtime Settings

This appendix reports the data splits, optimizer and model settings, evaluation protocol, and reconstruction settings used in the experiments.

B.1 Data Splits and Universe Allocation

Table 4 lists the evaluation splits. Source-failure counts are evaluation sizes, not the number of accepted reconstructions.

Table 4: Evaluation splits, in tasks. All arms are evaluated on the same source and held-out tasks and select candidates on the same development set.

Benchmark	Source failures	Development	Held-out universe
GAIA2	U29: 40	U28: 20	U30: 50
Workspace-Bench	U01: 20	U05: 29	U06: 20

GAIA2. The source failures are 40 U29 tasks: 10 adaptability, 15 ambiguity, six execution, one search, and eight temporal tasks. The development set contains 20 U28 tasks, four per category, and the held-out universe U30 contributes 50 tasks. Random tasks come from U26, and Other-F tasks are mined from U27. Donors are drawn from U21–U27 and never from the source universe U29 or the held-out universe U30. The donor pool therefore includes the development universe U28. REPROSADDLER attempted all 40 source failures. The primary search accepted 18 reconstructions, and recovery on the remaining 22 cases accepted eight more. All 26 accepted scenarios form D_{gen} for harness optimization. Evaluation still covers all 40 source failures, including the 14 without an accepted reconstruction.

Workspace-Bench. The 388 English tasks are split into six universes, with 68 tasks in each of U01–U04 and 58 tasks in each of U05 and U06. Tasks that share stored inputs stay in the same universe. U01 is the source universe, and U03 is not used. We rescore the 204 saved U01 outputs (68 tasks with three runs each) with the official GPT-5.5 judge, using only the source inputs and without rerunning the agent. A task is eligible when its mean score is below 0.9 and the same nonempty set of failed rubric items appears in at least two of its three runs. Of the 22 eligible tasks, we select 20 by ascending mean score, then by how often this failed set appears, and then by task order. These 20 tasks are the source failures and also form the Original training set. The development set contains 29 U05 tasks, and the held-out universe U06 contributes 20 tasks.

Random and Other-F draw tasks from U02 and U04, and no task is shared between them. Other-F keeps only tasks whose failed-rubric signatures differ from every signature observed in the U01 source runs. The U02 and U04 scores used for this check come from an earlier scoring run, so Other-F separates rubric signatures rather than independently reviewed semantic failures.

B.2 OPTIMIZATION AND MODEL SETTINGS

Shared setup. All arms start from the same frozen seed harness and select candidates on the same development set. The executable arms, namely Random, Other-F, Original, and REPROSADDLER, also share the optimizer configuration below, so they differ only in the training data given to the optimizer. Observable uses a separate controller, described below, because its evidence cannot be executed.

Optimizer. We run AutoSaddler with its default settings and no modification. This includes its shuffled-epoch minibatch sampler with three tasks per minibatch and three epochs, its EvoDAG candidate search, and its separate candidate-selection, diagnosis/patch, and reflection sessions. The optimizer runs on the GitHub Copilot SDK with GPT-5.5 and `xhigh` reasoning, with sampler seed 20260730 and a limit of 5,000 metric calls. The diagnosis/patch, reflection, and candidate-selection sessions time out after 18,000, 3,600, and 1,800 seconds, respectively. Training and development evaluations use one execution per task.

Models and editable surfaces. GAIA2 uses the Meta-ARE default-agent harness. Its agent runs on Azure `gpt-5.5-20260424` with `xhigh` reasoning, and native validation uses Llama-3.3-70B-Instruct. Workspace-Bench uses GPT-5.5 through the ClaudeCode harness and the official GPT-5.5 rubric judge. The harness name does not imply a Claude model. Task timeouts are 900 seconds for GAIA2 and 2,000 seconds for Workspace-Bench. The optimizer may edit only the GAIA2 default-agent code and hooks or the Workspace-Bench launcher, adapter, and skills. Benchmark data and evaluator logic are never edited.

Observable controller. Each Observable iteration gives the optimizer a minibatch of three projected failures together with development feedback, and the controller then evaluates the resulting patch on the development set. No source task is executed before or after a patch. On GAIA2, the development feedback includes per-case scores, status, and rationales for the selected parent. On Workspace-Bench, it contains only aggregate development scores.

Runtime isolation. Workspace-Bench tasks run in disposable containers with two CPUs, 8 GiB of RAM, 512 processes, and 32 GiB of writable storage. Inputs and outputs are mapped separately, and only the judge can access evaluation material. The shared scheduler admits up to 16 native operations at once, with four reserved for REPROSADDLER, up to four for each other arm, eight task workers per arm, and eight host judges. These are upper limits under resource-pressure guards, not guaranteed concurrency or independent repetitions. Runtime compatibility fixes are recorded separately from learned patches, and a candidate identifier alone does not certify unchanged services, dependencies, or runtime configuration.

B.3 FINAL EVALUATION AND ACCOUNTING

Final evaluation freezes the development-selected candidate's commit, patch, and configuration together with the ordered list of evaluation tasks. Final scores are never shown to the optimizers, and no candidate is edited or selected again after testing. Held-out tasks are evaluated with the same frozen candidate without further tuning. On Workspace-Bench, each arm is frozen and evaluated once its own optimization finishes, so one arm may be evaluated while others are still running.

We report avg@3, the mean of each task's three native scores averaged over tasks. GAIA2 scores are binary, while Workspace-Bench scores are graded rubric fractions, so a Workspace-Bench avg@3 is not the percentage of fully solved tasks. Scores are not pooled across benchmarks. The three executions measure the variability of task execution, not three independent optimization runs. Reproduction witnesses found during generation are not independent tests after selection.

Each candidate is evaluated on $40 \times 3 = 120$ source runs and $50 \times 3 = 150$ held-out runs for GAIA2, and on $20 \times 3 = 60$ source runs and $20 \times 3 = 60$ held-out runs for Workspace-Bench. Complete runs keep their native scores even when the agent fails. Missing outputs, incomplete judging, and infrastructure errors are not assigned a score of zero or one. Interrupted evaluations resume from verified records without counting completed runs twice.

The Workspace-Bench Other-F run uses its best completed checkpoint on the development set, selected before its last scheduled iteration finished. The GAIA2 source universe U29 was also used in earlier exploratory studies, so its source failures measure repair, and generalization is measured on the held-out universe U30.

B.4 RECONSTRUCTION SETTINGS

Reconstruction model. On both benchmarks, reconstruction uses GPT-5.5 with `xhigh` reasoning, 64,000 output tokens, a 1,200-second limit per request, and up to three attempts per request. On Workspace-Bench, the same model also performs failure screening and semantic review.

GAIA2 primary and recovery. The primary search allows five hypotheses per case and 15 condition revisions per hypothesis. It requires a valid reference solution, an all-OFF control that succeeds in all three runs, and three complete all-ON runs in which at least one shows the target failure first. Recovery runs on the 22 unresolved cases with the same failure specification, projection, and donor. It allows seven hypotheses and 21 condition revisions, with the budgets in Appendix A.5. The primary search accepted 18 cases, and recovery accepted eight more. All 26 form D_{gen} for the GAIA2 optimization run.

Workspace-Bench semantic admission. Workspace-Bench generation uses independently reviewed natural-language failures and actual request and file conditions. Each paired task shares its base inputs and correct answer, and both references must validate. Admission follows the same rule as GAIA2 (Eq. 9). The all-OFF control runs three native executions, each of which must receive full rubric credit with no detected failure. The all-ON configuration also runs three executions, and at least one must exhibit the target failure as its earliest detected failure. Failures are detected by an

independent, condition-blind semantic judge that reports every supported failure in execution order (Appendix F), and the controller determines whether the target comes first. The requirements written for a generated task validate its reference pair, and scores on them are not comparable to the source task's full rubric. Deletion tests use the same three-run protocol as primary reproduction.

The primary search allows five hypotheses, at most three active conditions, six construction repairs, and five condition revisions. Recovery keeps five hypotheses, the three-condition limit, and the three-run OFF and ON protocol, while raising construction repairs to seven and condition revisions to six. It also reserves two extra construction attempts for an invalid OFF control, and budgets already used and the search history carry over into recovery. A fixed deadline ends new admissions, and cases still running at that point count as unsuccessful. Generation starts from independently reviewed, privacy-cleared default state and never retrieves or reconstructs the source files. Unlike earlier Workspace-Bench generators, it does not use privileged source inputs.

C Source Projection, Matching, and Validity Details

Table 5: Intended reconstruction information boundary. The Original arm is deliberately privileged. Historical Workspace-Bench contrastive generators also used privileged source inputs; they are not the later restricted-input variants described in Appendix B.4.

Allowed or conditionally retained	Excluded from strict generator access
De-identified request and typed oracle/agent relations	Persistent source application snapshots and inventories
Opaque role tokens, comparison constraints, run support	Source task IDs, file paths, account identifiers, raw private returns
Value-free operation lanes and source ordinals	Concrete private event contents and copied application values
Dependency/relative-phase metadata only when extracted	Invented dependencies presented as source evidence
Cleared donor state and executable tool schemas	Lookup access to the source environment

The general trusted source specification and the frozen optimizer projection are different objects. A trace-visible environment event may contribute a value-free blueprint node while raw `environment_actions` remain excluded from the optimizer projection. The projector must audit allowed argument values before serialization; renaming keys or hashing the output is not sufficient sanitization. A privacy certificate records the implemented contract and provenance, not a theorem that no sensitive information can be inferred from a rare workflow.

Matching semantics. Typed action alignment normalizes app/function aliases and argument representations, then compares action structures with deterministic tie-breaking. Earliest checks use the registered ordering of oracle and actual discrepancies. Repeated operations and missing actions require this explicit alignment; simple substring matching of a failure label is insufficient. Natural-language content/description differences additionally use recorded benchmark failure evidence naming the relevant decision tool as limited corroboration. That gate is not a separate semantic-equivalence test of the argument text. Any analysis of deterministic matching must identify this judge-assisted subset.

Workspace-Bench contracts. Evaluations on source tasks retain the native Workspace-Bench judge. Earlier Workspace-Bench generation froze the exact failed-obligation set, whereas the latest same-F variant changes both its failure representation and admission rule as described in Appendix B.4. Preserving a benchmark contract, reproducing an error, and restricting generator access are separate claims.

Failure accounting. For every expected repeat, retain score, completeness, exception status, target-match flag, earliest flag where applicable, trace/output hashes, and whether semantic judge corroboration was required. Mechanical invalidity, incomplete execution, successful execution, target failure, and other failure must remain separate categories. Retry counts belong in cost reports even when the eventual trial succeeds. Post-selection replay must not feed back into the frozen generator or candidate choice.

D FAILURE PATTERNS EXTRACTED FROM RECORDED EVIDENCE

We catalogue observed failures from saved source evidence for GAIA2 and Workspace-Bench. Supporting IDs refer to source benchmark tasks, not generated tasks; support means at least one reviewed execution, not failure on every repeat. Descriptions do not assert hidden causes.

No common pattern-discovery iteration is recorded. We therefore report **N**, the number of distinct supporting scenarios, instead of an iteration. Case-local hypothesis numbers, condition revisions, and campaign versions are not discovery iterations. A scenario may support several patterns, so support counts need not sum to the number of unique scenarios.

D.1 GAIA2

Table 6 covers the registered failure relations of all 40 GAIA2 source failures in U29. We group them into 27 categories only when the operation, affected argument, relation, and comparator semantics agree. These are typed error categories, not evidence that their supporting scenarios share a latent mechanism. Scenario identifiers omit the common `scenario_universe_29_` prefix.

"Omitted" means that at least one required action instance lacks a match; it does not mean that the agent never called that operation. Temporal descriptions retain the recorded offset bins without exposing source timestamps. Content and description mismatches refer to the registered comparison, not a newly asserted requirement to copy oracle prose verbatim.

This source-failure inventory is not an admission table. The primary search accepted reconstructions for 18 of these 40 source failures, and recovery accepted eight more (Appendix B.1). The 14 source failures without an accepted reconstruction remain valid entries in this inventory.

Table 6: Recorded GAIA2 failure patterns: 27 typed categories covering 40 source failures. N counts distinct supporting scenarios, not generation attempts or accepted reproductions.

ID	N	Observed Failure Pattern	Supporting Scenarios
G01	1	Cab pickup time is less than one minute earlier than required	`096dyu`
G02	5	A required calendar-event creation is omitted	`0fccbi`, `1tk11t`, `4bytkx`, `i6hy8j`, `ja2axd`
G03	1	Required group-conversation creation in Messages is omitted	`0q1dww`
G04	1	A required cab-booking action is omitted	`23k4gf`
G05	3	A required message or response to the user is omitted	`2qw4nk`, `anc8y4`, `b6v3y8`
G06	1	Cab booking omits the required pickup-time argument	`2rzj8j`
G07	3	A required direct-message action in Chats is omitted	`34lgxy`, `aa11lh`, `hurfiq`
G08	1	A required email-deletion action is omitted	`5ajk1x`
G09	2	A required outgoing email is not sent	`71j61f`, `lxe62a`
G10	1	A required calendar-event deletion is omitted	`7v5wh0`
G11	2	Calendar-event end time is at least one hour earlier than required	`8c9am2`, `ivo8et`
G12	2	Calendar-event end time is at least one hour later than required	`8hgfug`, `wllfb8`
G13	1	Outgoing email content differs from the registered expected content	`a2e3ox`
G14	1	Email deletion uses a different target identifier than required	`d52ytz`
G15	1	Cab booking uses a different pickup location than required	`dp3q6c`
G16	1	The current user is omitted from the required calendar attendee set	`dpbafp`
G17	1	Cab pickup time is at least one hour later than required	`fqdxfe`
G18	2	Calendar-event description differs from the registered expected description	`jvohtk`, `rkdpfj`
G19	2	A required message-sending action in Messages is omitted	`kao12j`, `qodb8j`
G20	1	An additional, unrequired email-reply action is issued	`qrxrry`
G21	1	A required shopping add-to-cart action is omitted	`tf35kd`
G22	1	A required email-reply action is omitted	`tnxtee`
G23	1	A message-sending action omits the required attachment-path value	`wupkjw`
G24	1	A required file-removal action is omitted	`xcv49b`
G25	1	Checkout omits the required discount-code argument	`xinlfl`
G26	1	Required group-conversation creation in Chats is omitted	`y8x3mf`
G27	1	The current user is additionally included in the email recipient set	`z85cjp`

D.2 WORKSPACE-BENCH

Table 7 includes all 120 independently reviewed, admitted semantic failure specifications from the 39 U01 tasks with supported failures in the saved screening results. Screening completed for 63 of 68 tasks; the remaining five are not treated as failure-free. The inventory includes the 65 specifications from the 20 tasks selected for generation and 55 additional specifications from 19 other supported tasks. It excludes 297 proposals marked as having unknown requirement visibility, a rubric error, or an independent-review rejection.

We retain 118 granular entries, merging only two pairs with the same cross-task erroneous relation: falsely describing detailed source tables as title-only, and substituting an unrelated corpus for the requested input batch. Other missing-content, aggregation, and policy-fidelity errors remain separate. Thus the row count is not a count of distinct global mechanisms and should not be compared directly with GAIA2's coarser typed categories. All supporting identifiers belong to U01; the common universe prefix is omitted.

Source-coverage and correctness statements below summarize the saved adjudications, not a new inspection of private source files. This is also not a reproduction-success table: a separate completed generation snapshot accepted 22 failure specifications from 13 source tasks. Neither that acceptance count nor a later 20-example training selection defines the membership of this broader source-failure inventory.

Table 7: Recorded Workspace-Bench failure patterns: 118 granular entries covering 120 admitted failure specifications from 39 source tasks. N counts distinct supporting tasks for each entry.

ID	N	Observed Failure Pattern	Supporting Scenarios
W001	1	Production analysis is over-aggregated and omits required vehicle-type details	task_1
W002	1	Sales analysis is over-aggregated and omits a required vehicle-type detail	task_1
W003	1	A positive-growth commercial-vehicle export subcategory is omitted or misattributed	task_1
W004	1	Reports add or claim use of source categories outside the requested scope	task_106
W005	1	High-expense ranking uses the wrong monetary series	task_15
W006	1	Correct amounts are paired with incorrect item labels in top-entry extraction	task_15
W007	2	Detailed technical spreadsheets are falsely characterized as containing only titles	task_166, task_167
W008	1	Lighting-system records and technical attributes are replaced by a generic topic summary	task_167
W009	1	Structural-body product records and key attributes are omitted	task_167
W010	1	Exterior-joining process records and their attributes are omitted	task_167
W011	1	An unsupported vehicle-system source and technical area are added outside the intended input set	task_167
W012	1	A source material acronym is incorrectly expanded or translated	task_167
W013	1	An adjacent vehicle-system area is incorrectly included in a supplier's body-technology portfolio	task_168
W014	1	Head-up-display glazing explanation omits its operating principle and virtual-image placement detail	task_168
W015	1	Acoustic-glass performance metrics and ultraviolet-control details are omitted	task_168
W016	1	Switchable glazing explanation omits the operating mechanism across no-field and energized states	task_168
W017	1	Source-specified glass encapsulation methods are omitted	task_168
W018	1	Source-specified glass-antenna implementation types are omitted	task_168
W019	1	Solar-roof glazing explanation omits the photovoltaic current-generation principle	task_168
W020	1	A supplier's pillarless electric-door system and larger-opening benefit are omitted	task_168
W021	1	Source-specified exterior joining, cutting, and welding process categories are omitted	task_168
W022	1	Battery-enclosure technology and its protective and body-safety functions are omitted	task_168
W023	1	Model-level output rows fail to identify the model or subtype	task_180
W024	1	Model sales are double-counted by mixing aggregate rows with their component rows	task_180
W025	2	An unrelated input corpus replaces the user-designated batch, producing non-target outputs	task_194, task_367
W026	1	Rewritten resumes omit expected-salary and gender fields from the original basic information	task_194
W027	1	A teaching plan omits class profiles and roster/seating-based grouping information	task_195
W028	1	A teaching plan omits timetable-derived weekday schedules and weekly subject counts	task_195
W029	1	A teaching plan omits assessment analysis and fails to adapt objectives and learner support to performance	task_195
W030	1	A teaching plan omits attendance analysis and corresponding adjustment measures	task_195
W031	1	Source integration introduces unsupported derived content while dropping original details	task_206

Continued on next page

Table 7 (continued)

ID	N	Observed Failure Pattern	Supporting Scenarios
W032	1	Department remuneration analysis substitutes monthly wage medians for the required annual median statistic	task_251
W033	1	Procurement quantity or amount totals are omitted or replaced by future-order or reservation aggregates	task_256
W034	1	Daily material usage is replaced by an inventory-outbound quantity	task_256
W035	1	Outbound/use/packaging reconciliation uses the wrong comparison and reports closure despite a required discrepancy	task_256
W036	1	A management report omits the required turnover total and presents other cash-flow or payroll measures instead	task_260
W037	1	A management report omits the total member count	task_260
W038	1	Employee performance, commission, and salary totals use wrong components or omit values and top-performer identification	task_260
W039	1	Current-period accomplishments and next-period plans are not preserved as distinct, complete work summaries	task_283
W040	1	Monthly and store-level quarterly indicators or target breakdowns are omitted or misplaced	task_283
W041	1	Promotion summaries omit or conflate weekly cooperation plans and quarterly strategy details	task_283
W042	1	Required campaign windows, deadlines, and phase-specific time nodes are incompletely preserved	task_283
W043	1	Planning reports mix in out-of-scope material while omitting required input information	task_283
W044	1	Work-item descriptions include unrequested completion-status annotations	task_284
W045	1	A requested work-item list includes an unrequested title-like caption	task_284
W046	1	Detailed current-week accomplishment summaries are missing or incomplete	task_285
W047	1	Collaboration or promotion-plan completion status is omitted	task_285
W048	1	Core current-week operational data are omitted from the statistics section	task_285
W049	1	A report omits part of the mandatory monthly metric set	task_285
W050	1	A report makes inaccurate or unsupported claims about which source materials it integrated	task_285
W051	1	A user-growth metric loses its accompanying change-rate qualifier	task_287
W052	1	A report omits the next-period membership-growth plan	task_287
W053	1	A social-group metric retains current and target values but omits the completion rate	task_287
W054	1	Cost-optimization recommendations fail to cover all required major cost categories	task_29
W055	1	Registration outcomes remain missing, raw, null, or ambiguous instead of consistent configuration-ready approval states	task_301
W056	1	Product-registration rows are mapped to unrelated activity contexts	task_301
W057	1	A combined master table contains an incorrect set of required notice records	task_318
W058	1	Production-status labels conflict with required active/discontinued classifications and validity-date evidence	task_318
W059	1	Low-progress work is collapsed into broad packages rather than actionable subitems with rates and owners	task_325
W060	1	Event implementation is compressed into a narrative instead of an executable work sequence	task_325
W061	1	An unfinished wayfinding dependency is not converted into an event-site warning and contingency plan	task_325
W062	1	A planning input covering site selection, renovation, and moving work is not integrated	task_325
W063	1	Generic strategy modules replace the requested product-level optimized submission plan	task_331
W064	1	Product-level quantitative optimization data needed to revise submissions are omitted	task_331
W065	1	Product-level competitor comparisons, differentiation, price judgments, and operational risks are omitted	task_331
W066	1	Generic promotion advice replaces product-targeted channels and optimization actions	task_331
W067	1	Official promotion routes mismatch source eligibility options and omit the corresponding registration link	task_339
W068	1	Campaign lifecycle coverage omits or blurs preparation, tracking, marketing, warehouse, and review modules	task_339
W069	1	Product-registration information is incomplete	task_339
W070	1	The required design-role image handoff is missing from the designated registration-review stage	task_339
W071	1	Source-specified time allocations for registration follow-up, traffic analysis, and campaign summaries are omitted	task_339
W072	1	An on-site risk notice omits part of the required emergency-contact set	task_341
W073	1	An emergency-reporting process omits required escalation beyond normal reporting lines	task_341
W074	1	Accident-accountability rules are over-generalized, losing role-specific disciplinary levels	task_341
W075	1	Budget category subtotals are not explicitly labeled despite detailed rows and an overall total	task_343
W076	1	A progress tracker omits a distinct overdue or schedule-status field	task_343
W077	1	A specific task deadline is replaced by a broader date range	task_343
W078	1	Rehearsal and equipment testing omit the required administrative coordination with the venue	task_343

Continued on next page

Table 7 (continued)

ID	N	Observed Failure Pattern	Supporting Scenarios
W079	1	A manual contradicts the required post-event reimbursement and media-archiving window used by its tracker	task_343
W080	1	Role-specific permissions and prohibitions across workbook categories remain incomplete or ambiguous	task_35
W081	1	Operational recommendations and controls are introduced without support in the studied materials	task_355
W082	1	Catalog entries are incorrectly integrated or counted into top-level categories	task_359
W083	1	Monthly sales extrema, their range, and the seasonality conclusion are incomplete or incorrect	task_36
W084	1	The combined activity-related reimbursement share is not aggregated or highlighted	task_36
W085	1	A documented abnormal account overdraft is not identified	task_36
W086	1	High position-level reimbursement totals are omitted from abnormal-expense review	task_36
W087	1	A formal report omits gender-balance analysis despite figures in its supporting workbook	task_37
W088	1	A key-position reserve assessment omits a mid-tenure cohort's stability strength and emphasizes weakness instead	task_37
W089	1	Decision alignment omits the attendance rate implied by present-versus-expected counts	task_37
W090	1	Hiring and departure counts are reported without their resulting net increase	task_37
W091	1	Recruitment decisions are not assessed against qualification needs	task_37
W092	1	Daily source rows are incorrectly aggregated into monthly new-work totals	task_39
W093	1	Annual follower-retention values are overstated relative to the required reference	task_39
W094	1	A report omits or misreports part of the requested core indicator set	task_39
W095	1	Key activity and average-performance metrics lack clear operational conclusions	task_39
W096	1	Exported date or numeric metrics use incorrect data types or formats	task_39
W097	1	Strategy and target justifications introduce unsupported metrics or use evidence outside the specified core set	task_40
W098	1	A procurement exception report omits a documented status-line formatting or bracket abnormality	task_66
W099	1	An integrated manual fails to preserve required source policy terms and operative management details	task_71
W100	1	An integrated manual adds unsupported content and incorrectly maps source modules to inputs	task_72
W101	1	Exceptional non-hospital extended stays lose the linked written-explanation and senior-approval requirements	task_85
W102	1	Reimbursement guidance omits the rule for employees of different levels traveling together	task_85
W103	1	Two required post-trip submissions are presented as alternatives rather than separate obligations	task_85
W104	1	A vehicle-management manual omits the required borrowing or loan-management topic	task_86
W105	1	Vehicle-level cost accounting omits required mileage, fuel, per-distance standards, or tire-consumption items	task_86
W106	1	Maintenance rules lose tiered repair approvals or alter the prescribed schedule structure	task_86
W107	1	Vehicle-expense rules alter individual write-off, centralized driver reimbursement, or the approval sequence	task_86
W108	1	Driver-management guidance omits applicability, attendance discipline, or driving-conduct penalties	task_86
W109	1	Safety-incident handling omits accident-report contents, graded penalties, or liability-ratio rules	task_86
W110	1	Access and parking policy omits special-system precedence and category-specific registration requirements	task_86
W111	1	Centralized shuttle control is assigned to the wrong organizational owner	task_86
W112	1	Shuttle-operation guidance omits operating-time, commuting-support, or procurement/contract-cadence rules	task_86
W113	1	Internal cross-references disagree about where fuel and charging clauses were consolidated	task_86
W114	1	A dashboard omits the stated reason for a serious objective-progress delay	task_9
W115	1	A dashboard fails to assess a small projected completion improvement as overly conservative	task_9
W116	1	A dashboard omits required user-feedback issue-category counts	task_9
W117	1	A dashboard incorrectly presents a retention-completion metric alongside other objective-completion metrics	task_9
W118	1	Satisfaction figures are reported without their rating-scale label	task_9

E Engineering Failures and Their Treatment

This section distinguishes failures of the experimental machinery from valid behavioral outcomes of the reconstruction method. A generated condition set is evaluated behaviorally only after deterministic compilation, native-DAG audit, reference binding, strict-oracle replay, and completion of the required rollout batch. We label a case *engineering-failed* when this admissible evidence cannot be obtained

within the registered retry budget. By contrast, an *inconclusive* case has mechanically valid evidence but no earliest Same-F witness within the search budget. A certified Final has both valid evidence and an admitted Same-F witness.

Let $V(x, \tau)$ denote the mechanical-validity predicate for compiled scenario x and rollout τ, and let $M_F(\tau)$ denote an earliest match to target relation F. The behavioral outcome is defined only when the required batch $\mathcal{T}$ is admissible:

$$\left[\bigwedge_{\tau \in \mathcal{T}} V(x, \tau) = 1\right] \Longrightarrow Y(x, \mathcal{T}) \in \{\texttt{Same-F}, \texttt{Success}, \texttt{Other-F}\}. \tag{14}$$

If compilation, binding, transport, or execution prevents construction of such a batch, Y is missing rather than zero. Consequently, engineering failures are excluded from training sets and behavioral accuracy denominators and are reported separately. This fail-closed rule prevents malformed scenarios or provider interruptions from being counted either for or against the method.

E.1 Failure Taxonomy

The observed engineering failures fall into four broad classes.

Typed construction and reference handling. Some failures arose before behavioral evaluation because an early implementation applied constraints that were incompatible with runtime-created resources or with the source workflow graph. Examples of this class include requiring a runtime-created identifier to appear literally in the request, associating a simulated event with the wrong ancestor event, rejecting a value supplied by an explicit dependency because of presentation order, and requiring a resource created during execution to exist in the donor's initial state. These are compiler, validator, and binding defects: the hypothesis search receives no valid behavioral trial on which it could succeed or fail.

Model-response framing. Some authoring attempts produced structurally valid content wrapped in a response format that the parser could not consume, such as multiple concatenated JSON objects. These failures occur before scenario admission and therefore provide no evidence about the proposed causal conditions. The parser was made incremental while retaining schema validation, allowing later executions to proceed beyond this failure point.

Provider and worker interruption. Some scenarios passed native compilation and strict-oracle replay but lost one or more rollout results to external service errors. Observed mechanisms include rate limiting, remote token-validation failures, repeated-agent stalls, and an exception-construction defect triggered while propagating a provider error. These events concern availability and robustness of the execution stack rather than whether the generated conditions reproduce the target relation.

Mechanically incomplete controls with unresolved low-level cause. A final class exhausted retries at the all-OFF control gate despite passing deterministic compilation or strict-oracle preflight. When the preserved record does not identify a sufficiently specific lower-level cause, we classify the case as an *undetermined engineering failure*. We do not attribute such a case to provider instability or to the behavioral method without additional evidence.

E.2 Repair and Replay Evidence

Compiler and binding defects were repaired under a new contract version without modifying frozen campaign outcomes. Affected scaffolds subsequently passed deterministic compilation, exact native-DAG audit, and strict-oracle runtime replay. Some later became certified recovery Finals while the hypothesis, donor, target relation, and behavioral acceptance rule remained unchanged. These controlled replays provide direct evidence that the earlier terminal labels arose from implementation defects rather than an inability of the reconstruction objective to admit a valid Same-F scenario.

The execution layer was also hardened without relaxing behavioral admission. The launcher supports lower scenario concurrency and a larger repeated-stall allowance, newly created resources are referenced through typed creation aliases, and model-response extraction accepts a complete leading

JSON object while still validating its schema. These changes reduce known failure modes, but they do not retroactively convert an engineering failure into a method success. A case is promoted only after a new admissible evaluation; it may still become inconclusive if the repaired execution produces no earliest Same-F witness.

E.3 Relation to Method Validity

Engineering failures reduce effective coverage, consume budget, and expose limitations of the implementation. They are therefore reported rather than discarded. However, they do not constitute negative behavioral evidence for the Same-F search objective.

First, the admission predicate in Eq. 14 is not satisfied, so the target behavioral outcome is not observed. Coding such cases as behavioral failures would conflate missing evidence with evidence of absence. Second, multiple failures disappeared after parser, compiler, binding, or execution-layer repairs that left the behavioral method unchanged. Third, some interrupted batches contained successful completed runs before an external failure prevented collection of the full required batch. This demonstrates scenario executability but remains insufficient for admission under the fail-closed protocol.

Accordingly, engineering failures remain visible in campaign accounting but are excluded from AutoSaddler training and behavioral accuracy denominators. They are never promoted to Final or inconclusive without a new admissible run. This separation preserves two distinct quantities: engineering-failure incidence measures experimental-system reliability and missing coverage, whereas Final and inconclusive outcomes characterize the bounded behavioral search.

This distinction should not be read as claiming that every failed case would succeed after repair. Some low-level causes remain unresolved, and a repaired case can legitimately become inconclusive rather than Final. The defensible claim is narrower: when admissible behavioral evidence is absent, the record does not identify a failure of the reconstruction objective itself.

F Instructions Used in ReproSaddler

We provide the prompt templates for failure specification, executable reconstruction, behavioral verification, and downstream harness optimization. Each template states the component's goal, available evidence, permitted changes, and expected output. Braced fields denote instance-specific inputs, and private source environments are never supplied to the reconstruction agent.

For readability, we consolidate closely related instructions into eight panels and standardize their presentation. These are presentation templates, not verbatim transcripts of individual calls. Named modes and sessions in a panel correspond to separate invocations. In particular, independent review is never performed by asking the author to approve its own proposal. Both benchmarks share the same acceptance rule. They differ in how the failure is specified, which surfaces a condition may change, and how a failure is detected in a generated execution.

F.1 Failure Specification and Reconstruction

Failure specification and independent review. For GAIA2, the failure is the fixed typed relation compiled from the projected oracle and agent actions. Workspace-Bench failures are instead specified in natural language by an extraction and independent-review step, because Workspace-Bench evaluation is rubric-based rather than defined over typed actions. The **Failure Specification Prompt** (Figure 6) implements this step. It extracts de-identified semantic failures from the allowed Workspace-Bench evidence and then subjects each proposal to an independent review. It separates an observed discrepancy from an unverified explanation of its cause, and distinguishes unsupported requirements from genuine rubric contradictions. Concrete quotations used to verify citations stay inside this step and are never passed to the generator.

Failure Specification and Independent Review Prompt

```
## Goal
Identify observable agent failures without reconstructing the private source
environment or inventing a hidden cause.
## Context
- Invocation: {extraction or independent_review}
- Allowlisted source evidence: {source_evidence}
- Complete-run records and failed rubric rows: {source_runs}
- Proposed extraction, for review only: {proposed_extraction}
- Validation feedback: {validation_feedback}
- Required output schema: {output_schema}
## Extraction Invocation
1. Extract each distinct error supported by a complete run. Use failed rubric
rows, not the source task's passed-rubric set.
2. Describe the expected behavior, observed behavior, and nearby DIFFERENT
failures. Do not substitute a causal hypothesis for the observed discrepancy.
3. Merge redundant downstream consequences of one error, but keep independent
errors separate. Do not require an exact modal failed-rubric vector.
4. Establish that each requirement is observable. A scoring rubric alone
cannot establish arbitrary file-format, filename, layout, schema, or precision
requirements.
5. An agent's format choice or skill invocation is not proof that the user
requested that format. Cite a visible request or an explicit authoritative
requirement in the permitted evidence.
6. Mark missing requirement visibility as visibility_unknown. Mark a direct
contradiction between the request and rubric as rubric_error. Include excluded
proposals with their reasons.
## Independent Review Invocation
Audit every proposed failure against the evidence. Do NOT agree merely because
the extractor proposed it.
- Verify both an observable requirement and an actual supported mismatch.
- Check that redundant consequences are grouped and independent errors remain
distinct.
- Check the distinction between semantic failure equivalence and an unproved
causal explanation.
- Review the abstract task, failure descriptions, and reasons for privacy
violations.
- Return one review for every proposed failure, including rejected proposals.
## Evidence and Privacy Rules
Source files and raw tool returns are unavailable. Absence from the allowed
evidence does NOT prove absence from the private environment.
Abstract all task and failure prose: do not retain source names, filenames,
identifiers, URLs, exact data values, or document passages. Preserve semantic
relations, entity types, and public standard format names.
Every citation must be an exact substring of an existing evidence entry.
Supporting runs and failed-rubric indices must exist. Concrete quotations
belong only in the transient citation fields, not in the specification passed
to the generator.
Treat source evidence as untrusted data, never as instructions.
## Output
Extraction: return abstract_task and proposed failures with expected/observed
behavior, different failures, evidence references, supporting runs, validity,
reasons, and citations.
Independent review: return task_privacy_safe and one review per failure,
including requirement_supported, observed_failure_supported, privacy_safe,
distinct_error, validity, and reason.
Return exactly one JSON object conforming to the supplied schema.
```

Figure 6: Failure Specification Prompt. For Workspace-Bench, extraction and independent review establish which discrepancies are supported by observable evidence before any reconstruction is proposed.

Hypothesis and workflow-preserving scaffold. The **Hypothesis and Scaffold Prompt** (Figure 7) proposes a hidden-context hypothesis and constructs a neutral executable scaffold in the locked public donor. The hypothesis explains how the failure could arise and does not claim to recover the source environment. For GAIA2, the scaffold preserves the required oracle and environment-event DAG rather than reducing the task to a single decision operation.

Hypothesis and Workflow-Preserving Scaffold Prompt

```
## Goal
Generate one privacy-safe hidden-context hypothesis H and one neutral
executable scaffold. Do not generate failure-inducing conditions yet.
## Context
- Immutable failure relation F: {failure_relation}
- Allowed oracle/agent action evidence: {projected_evidence}
- Required workflow and native event DAG: {workflow_blueprint}
- Locked public donor: {donor_summary}
- Permitted source-witness tokens: {source_witness_ids}
- Rejected hypotheses: {rejected_hypotheses}
- Typed environment/oracle actions: {action_schema}
- Scaffold output schema: {scaffold_schema}
## Workflow
1. Propose a falsifiable contextual hypothesis explaining how F could arise.
Do not merely restate F or superficially rename a rejected hypothesis.
2. Bind the required roles to donor-native entities. Reuse suitable public
state before adding neutral entities required by the workflow.
3. Write a fresh request and a correct reference specification that agree on
the required operations and values.
4. Keep the shared scaffold easy enough for the frozen seed harness to solve
when all later conditions are OFF.
5. Keep hypothesized failure conditions out of the shared request, bindings,
base state, and oracle. They will be added as independently switchable
conditions.
## Workflow-Preservation Contract
Preserve all required oracle operations, their multiplicity and source order,
and the available dependencies and relative timing. This requirement takes
precedence over any instruction to produce a one-operation micro-task.
For native GAIA2 events:
- Preserve both oracle and environment-event nodes, including required
completion/reporting actions.
- Copy each oracle node's source token and dependency tokens to the obligation
and its step.
- Preserve the step's required relative time.
- Do not move a timed source environment event into the initial base state.
- Use only the supplied witness tokens; do not invent source dependencies.
## Boundaries
Never inspect, reconstruct, or copy source-private state or concrete source
values. A public donor is construction material, not evidence of the private
cause.
Use only the supplied typed actions and role-binding schema. The request,
reference, role bindings, and failure matcher must describe the same task.
Do not encode the answer, insert an instruction to fail, or weaken the target
obligation to make reproduction easier.
## Output
```

(continued)

```
Return one ScenarioScaffold JSON object containing the hypothesis, shared
request, donor-role bindings, neutral base actions, oracle obligations, and
invariant contract, as specified by the supplied schema.
The controller performs compilation and reference replay before condition
generation. Your proposal is not a validity or reproduction certificate.
```

Figure 7: Hypothesis and Scaffold Prompt. The generator builds a neutral donor-native task while retaining the fixed failure target and the workflow needed to interpret its reproduction.

Condition generation and trace-guided refinement. The **Condition Search Prompt** (Figure 8) groups initial condition generation and subsequent single-condition revisions. The initial call proposes executable ON/OFF interventions, and revision calls must ground their change in observed execution differences. A masked target calls for removing the preceding failure, not increasing task difficulty. The Unguided variant withholds this localized trace comparison.

Switchable Conditions and Trace-Guided Refinement Prompt

```
## Goal
Construct or revise executable conditions that make the frozen seed harness
exhibit the immutable failure F while leaving the shared workflow intact.
## Context
- Invocation: {initial_conditions or revise_condition}
- Fixed failure F and hypothesis H: {failure_and_hypothesis}
- Current scaffold and condition set: {current_design}
- Locked donor and typed actions: {donor_and_action_schema}
- Current outcome: {outcome}
- Grounded source/generated comparison, when enabled: {trace_comparison}
- Rejected condition predicates: {rejected_predicates}
- Required output schema: {condition_or_revision_schema}
## Condition Contract
Each candidate is one independently switchable pre-execution condition, with a
grounded predicate, explicit ON/OFF transformations, and invariant
identifiers.
- Task conditions change user-facing request clauses.
- Environment conditions change typed app state and keep task clauses empty.
- Cross-surface conditions change both only when their interaction is required
by H.
- Use the smallest surface that expresses the proposed mechanism.
- Declare precise allowed state-difference paths.
- Use existing scaffold bindings. A candidate cannot depend on a new binding
or an alias created by another candidate.
- OFF restores a neutral control, not an opposite difficulty. A neutral OFF
clarification is permitted when the condition contrasts implicit and explicit
wording.
- Do not remove an oracle obligation, encode the answer, invent unsupported
tool semantics, or tell the agent to fail.
## Initial-Condition Invocation
Return one ConditionSet with version 1. The initial set is a testable
proposal, not a claim that every relevant condition has been found.
Preserve H, F, the donor, bindings, semantic goal, oracle, workload, decision
operation, and invariant contract.
## Trace-Guided Revision Invocation
1. Cite one difference_id from the supplied comparison.
2. Add exactly one new candidate or strengthen exactly one existing candidate.
Do not rewrite the complete design or edit two candidates at once.
3. The change must address the concrete observed avoidance behavior, not
merely make the task harder.
```

(continued)

```
4. Strengthening preserves the candidate identifier and increases its
strength. Addition requires a new identifier and a substantively new
predicate.
5. If F occurs but is not earliest, cite a preceding-relation difference and
revise only to suppress that preceding failure. Do not strengthen F or add
workload in this mode.
6. If no executable, source-compatible condition is supported, return a
grounded stop.
## Unguided Revision Variant
When localized comparison is withheld, infer the next condition only from F,
H, the current design, rejected predicates, and aggregate outcome. Do not
invent a trace alignment or an avoidance explanation. Stop if no substantively
new condition follows.
## Output and Controller Boundary
Initial call: one ConditionSet JSON object.
Guided revision: one object with stop_signal, avoidance_spot, and revision.
The avoidance spot identifies the evidence difference and proposed
pre-execution condition.
Grounded stop: stop_signal=true, avoidance_spot=null, revision=null.
Applied revisions return through compilation, reference validity, and the
all-OFF control. The model cannot declare admission from its own explanation.
```

Figure 8: Condition Search Prompt. Initial interventions and evidence-grounded revisions share an immutable scaffold, while the Unguided variant removes localized trace feedback without changing the target.

Control repair and recovery. The **Repair and Recovery Prompt** (Figure 9) groups three related operations: stabilizing an unsuccessful control, repairing oracle–matcher alignment, and revisiting unresolved searches using privacy-bounded history. Their editable fields remain different. Recovery does not relax the admission rule or reinterpret engineering failures as behavioral evidence.

Control Repair and History-Aware Recovery Prompt

```
## Goal
Resolve the supplied construction or search obstacle without changing the
immutable failure, locked donor, or required workflow.
## Context
- Invocation: {control_repair, alignment_repair, or recovery}
- Immutable F and locked donor: {fixed_specification}
- Current scaffold, hypothesis, and conditions: {current_design}
- Control results or contradiction diagnostics: {diagnostics}
- Privacy-safe search-history summary: {search_history}
- Current outcome and stopping reason: {outcome_and_stop}
- Required response schema: {output_schema}
## Control-Repair Invocation
Simplify only the shared request so the fixed workflow is an easy, direct task
with all conditions OFF.
Preserve H, every condition and transformation, donor bindings, base state,
oracle, workload, decision operation, and invariants.
Remove incidental ambiguity and condition-like complexity. Do not add a
protective hint that directly teaches the agent to avoid F. Do not remove
required workflow operations under the guise of simplification.
Return only task_prefix, task_suffix, and rationale. The controller must
invalidate the old control result and test the repaired request again.
## Oracle--Matcher Alignment Invocation
Use the supplied contradiction between native success and earliest-target
matching to repair task/oracle alignment.
```

(continued)

```
Preserve the entire hypothesis and condition set, identifiers, bindings, base
actions, invariant contract, obligation identifiers, and oracle operation
types and order.
Change only shared request wording, oracle description, or permitted oracle
argument values needed to make the fixed relation discriminative.
Do not add workload, change the decision operation, or redefine the failure
matcher. Return the complete ScenarioDesign in the supplied schema; only the
permitted fields may differ.
## History-Aware Recovery Invocation
Treat the parent summary as negative search evidence, not a source of concrete
scenario values. Do not repeat an earlier hypothesis or predicate with
superficial rewording.
- Unstable control: simplify shared wording while retaining all obligations.
- Masked target: suppress the specific preceding failure while preserving F.
- Absent target: explore a substantively different mechanism supported by the
allowed evidence.
- Engineering failure: resolve construction or execution validity before
interpreting model behavior.
Do not import raw parent traces, tool outputs, model responses, or staged
scenarios. Preserve donor-native privacy, the oracle/environment DAG,
multiplicity, dependencies, timing, and global order.
Return the response type of the underlying scaffold, condition, or repair
call. Recovery adds history and guidance; it does not replace that call's
output contract.
## Acceptance Remains External
For both benchmarks, all three OFF runs must complete the required workflow
with full credit and without a detected failure. At least one of three valid
ON runs must preserve the workflow and exhibit F as the earliest detected
failure.
Do not weaken this rule, treat an infrastructure error as F, or change a
verdict merely by changing its descriptive label.
```

Figure 9: Repair and Recovery Prompt. Distinct repair modes preserve their own edit boundaries, and history-aware recovery retains the primary validity and reproduction criteria.

F.2 Workspace-Bench Construction and Semantic Verification

Paired task construction and independent audit. The **Paired Construction Prompt** (Figure 10) adapts reconstruction to native Workspace-Bench files. It creates a fresh task, correct reference outputs, and conditions that change actual request wording or add contextual files. Both variants retain the same correct answer. A separate audit checks well-posedness and condition validity, and subsequent condition-only revisions cannot modify a locked scaffold.

Paired Workspace-Bench Construction and Independent Audit Prompt

```
## Goal
Construct a fresh Workspace-Bench task that can reproduce the supplied
natural-language F. Preserve the workflow and semantic error, NOT the entire
source failed-rubric vector.
## Context
- Invocation: {author, independent_audit, or condition_revision}
- Reviewed abstract task and fixed F: {abstract_task_and_failure}
- Supported native file schemas: {native_schema}
- Previous design and execution feedback: {design_and_feedback}
- Locked scaffold, if available: {locked_scaffold}
- Required response schema: {output_schema}
## Authoring Invocation
1. Invent coherent fresh entities, values, and input files. Do not reuse
source identities, filenames, exact data, or private files.
```

(continued)

```
2. Construct one single-stage local-filesystem task with a complete correct
reference output and legitimate task requirements.
3. Choose a causal hypothesis and at most three conditions. A condition must
explain how the error could arise, not merely restate F.
4. Keep the task executable in the supported native formats. Do not require
external credentials, unavailable applications, or executable model-authored
code.
5. Derive all reference content correctly from the fresh inputs.
6. Keep the OFF task easy enough that the frozen seed harness receives full
credit in all three OFF runs.
## Actual OFF/ON Request and File Contrast
A request condition uses request_change with id, off_text, and on_text.
- off_text must occur exactly once in off_request.
- The condition identifier and request-change identifier must agree.
- Different request anchors must not overlap.
- on_text may be empty to remove wording literally.
- Only enabled replacements and condition files appear in ON; OFF has no
enabled conditions.
- Do not replace omission with an instruction telling the agent to ignore a
requirement.
Both variants must be well-posed and have the SAME correct answer. Omitting a
redundant clarification is valid; removing the only accessible statement of a
required output format is not.
Condition files may add plausible working context. They must not overwrite
base files, delete necessary evidence, reveal solved references, or contain
instructions to fail.
Requirements must be legitimate and accessible in the request or native
inputs. Do not import unsupported rubric-only constraints.
## Independent Audit Invocation
Inspect the proposed task, actual OFF/ON requests, input definitions, and
references against the abstract task and F.
Reject impossible or contradictory tasks, unsupported requirements, different
correct answers across variants, invalid references, source-entity reuse,
trivial goal substitution, protective conditions, and instructions to fail.
Check arithmetic, completeness, native formats, and actual condition exposure.
Every failed check requires concrete repair feedback. Do not rubber-stamp the
author.
## Locked-Scaffold Revision Invocation
Use actual OFF/ON feedback to revise conditions. A successful agent execution
is negative evidence for the proposed mechanism, not proof that the evaluator
is defective.
If F occurs but is not the earliest failure, revise only to suppress the
preceding failure. Do not strengthen F or add workload in this mode.
If the scaffold is locked, return ONLY the conditions object. Do not reproduce
or modify the hypothesis, request, inputs, requirements, or reference outputs.
## Output
Author: one Design with hypothesis, off_request, requirements, input_files,
reference_files, and conditions.
Independent auditor: structured validity flags and actionable repair_feedback.
Locked revision: one ConditionRevision with one to three conditions.
```

Figure 10: Paired Construction Prompt. Workspace-Bench interventions alter the actual request or context files while preserving a valid task and common reference answer, and independent auditing precedes behavioral interpretation.

Independent semantic failure detection. The **Semantic Failure Matching Prompt** (Figure 11) plays the role of the typed failure matcher for Workspace-Bench. It judges actual generated execution evidence without being told whether it came from OFF or ON, and it matches erroneous action/output relations rather than wording similarity or a rubric index. The judge reports every supported failure in execution order, which lets the controller apply the same admission rule as for GAIA2. The OFF

control passes only when all three runs receive full credit with no detected failure, and a trial is accepted only when at least one of three valid ON runs exhibits F as its earliest detected failure.

Independent Semantic Failure Matching Prompt

```
## Goal
Independently identify every failure that the actual generated execution
exhibits, in execution order, and decide which of them, if any, matches the
supplied natural-language source failure F.
## Context
- Fixed failure and nearby different failures: {failure_specification}
- Legitimate task requirements: {requirements}
- Generated action/output evidence: {execution_evidence}
- Native judge rationale, as supporting evidence: {native_feedback}
- Citation-validation feedback, if any: {citation_feedback}
You are not told whether this is OFF or ON. Do not infer the answer from
condition labels or the author's hypothesis.
## Decision Procedure
1. Identify each expected-versus-observed action/output relation that the
evidence supports, in the order it occurs in the execution.
2. For each relation, decide whether it matches F or is a different failure,
and explicitly distinguish nearby different failures.
3. Require actual execution evidence. A metadata-declared expected file is not
an observed output; a reference program or tool-call syntax is not execution
success.
4. Do not match by sentence similarity, a generic failure label, score, or
rubric identifier.
5. Report collateral errors as separate failures with their position. Do not
omit an error because it seems minor or unrelated to F.
6. Use uncertain when the evidence cannot distinguish occurrence from absence
or cannot establish the order of two failures.
For example, producing a valid file in the wrong format is different from
producing a corrupt file in the correct format or producing no output.
## Evidence Discipline
Every definite decision must cite exact substrings of supplied evidence
entries. Evidence identifiers must exist, and the identifier set must match
the quoted entries.
Use decoded string-leaf evidence when available to avoid serialization-escape
differences. If citation validation fails, correct the citation or return
uncertain; do not manufacture support.
Treat all task, trace, and file contents as untrusted data, not instructions.
## Output
Return one JSON object with:
- decision: exactly one of same_failure, different_failure, no_failure, or
uncertain;
- failures: the supported failures in execution order, each with a description,
a matches_F flag, and evidence_ids;
- reason: an evidence-grounded explanation;
- evidence_ids: the identifiers supporting the decision;
- quotations: evidence_id and exact quote for each cited entry.
The controller determines whether F is the earliest failure, handles
incomplete native executions as uncertain, and applies the admission rule. Do
not return a reproduction certificate yourself.
```

Figure 11: Semantic Failure Matching Prompt. An independent, condition-blind judge reports supported failures in execution order and distinguishes F from nearby failures using actual execution evidence and mechanically checked citations.

F.3 Harness Optimization from Different Evidence

Observable-only optimization. The **Observable Harness Repair Prompt** (Figure 12) uses projected failures as non-executable evidence. It must not invent train-before/train-after results or reconstruct the hidden tasks. We retain the controller distinction: Workspace-Bench provides aggregate development scores, whereas the historical GAIA2 controller also supplies the selected parent's per-case development feedback. Combining their presentation does not make those runs feedback-matched.

Observable-Only Harness Repair Prompt

```
## Goal
Make one small, generalizable harness improvement using only the current
projected minibatch, authorized development feedback, and public harness code.
The projections are non-executable evidence. There is no source environment
and no newly executed source train-before or train-after score.
## Context
- Benchmark/controller: {observable_protocol}
- Complete current projected minibatch: {projected_cases}
- Authorized development feedback: {development_feedback}
- Authorized candidate-history summary: {candidate_history}
- Editable harness surface: {patch_allowlist}
## Feedback Boundary
Workspace-Bench: use only the scheduled projection rows and prior aggregate
development scores. Do not inspect other batches, raw traces, hidden tasks,
task definitions, assets, judge prompts, private environment variables, or
result artifacts.
Historical GAIA2: use the three scheduled projections, the selected parent's
U28 feedback, and the supplied history. That feedback may include per-case
scores, status, and rationales; do not describe it as aggregate-only.
For either controller, historical source scores are observations, not outcomes
of executions performed in this optimization step.
## Workflow
1. Infer a cross-case harness weakness from the allowed request/action
relations and available workflow evidence.
2. Inspect only the public harness within the authorized work area.
3. Make one coherent repair supported by the minibatch while avoiding
regressions suggested by the permitted development feedback.
4. Change only the allowlisted prompt, hook, loop, launcher, adapter, or skill
surface applicable to this benchmark.
5. Inspect the complete diff and leave the harness runnable.
## Prohibitions
Do not execute or reconstruct source tasks, invent training scores, search for
additional source cases, or access private environments.
Do not encode case identifiers, concrete names, dates, message text, expected
answers, rubric text, or task-specific lookup tables.
Do not modify tasks, datasets, application implementations, validators,
judges, or benchmark infrastructure outside the allowlist.
For Workspace-Bench, do not run tasks, tests, checkers, or benchmark rollouts;
only non-executing syntax inspection is permitted. The historical GAIA2
template permits focused syntax/import checks, not new source-task executions.
## Output and External Evaluation
Leave one uncommitted harness patch. The controller validates, commits,
hashes, evaluates on development tasks, and records the candidate.
Do not treat an explanation of a likely repair as a measured improvement.
```

Figure 12: Observable Harness Repair Prompt. Projected evidence supports a general harness edit without source-task execution, and development feedback remains restricted by the benchmark-specific controller.

Execution-grounded optimization. The **Executable Harness Optimization Prompt** (Figure 13) presents AutoSaddler's candidate selection, diagnosis/patching, and reflection instructions together. These remain three separate sessions with different available evidence. The executable arms use the same optimizer, and only the training data changes. Actual pre/post executions, rather than an imagined failure scenario, provide the evidence for patch evaluation and subsequent lessons.

Executable Harness Optimization Prompt

```
## Goal
Improve performance across the task distribution, using development
performance as the selection objective. Diagnose from real executions and make
generalizable harness changes rather than memorizing training answers.
## Session-Specific Context
- Current session: {selection, diagnosis_and_patch, or reflection}
- Iteration and optimization phase: {iteration_and_phase}
- Candidate table, evolution DAG, and authorized lessons: {candidate_history}
- Selected parent harness: {parent_harness}
- Actual pre-patch minibatch evidence, after initial evaluation:
{before_results}
- Actual post-patch evidence, for reflection only: {after_results}
- Development results, when available: {development_results}
- Editable harness surface: {patch_allowlist}
## Session 0: Candidate Selection
Review the optimization history before choosing a base. Compare development
scores, regression history, code differences, and durable fixes.
Do not continue a declining lineage merely because many iterations were
invested in it. Continue, switch parent, or combine complementary changes
according to measured evidence.
Handle identified regressions when preparing the base. Verify any changes and
record which candidates were used and why.
## Session 1: Diagnosis and Patching
The controller has executed the selected harness on the current training
minibatch.
1. Read the scores, evaluation rationales, execution traces, and relevant
harness code.
2. Locate where behavior diverged from the required outcome. Explain why the
agent made that decision, not merely which task failed.
3. Check prior attempts and lessons before proposing another patch.
4. Address the underlying behavioral pattern with the smallest supported
change.
5. In the capability phase, prefer allowed executable-code changes; in the
steering phase, prefer allowed prompt, description, or hook changes.
6. Keep agent-facing instructions consistent with any permitted capability
change. The experiment's allowlist overrides generic phase guidance.
7. Verify the patch and record its target failures, diagnosis, approach,
changed components, and rationale.
Never hardcode scenario identifiers, names, expected outputs, rubric text, or
unique task-specific constants. Do not edit benchmark tasks, protected tools,
evaluators, or final-test data.
## External Re-evaluation
The controller reruns the same training minibatch and records measured
changes. An improved total minibatch score permits development evaluation
under the optimizer's acceptance rule.
Do not invent after-scores, reuse old results as new measurements, or equate
minibatch improvement with generalization.
## Session 2: Reflection
Compare actual before/after evidence and the patch diff for fixed, regressed,
still_failing, and still_passing cases.
```

(continued)

```
Prioritize regressions. For every fixed or regressed case, identify the
behavioral divergence and whether the diff touches the relevant code path.
Do not attribute a change to the patch, stochastic noise, or model
nondeterminism without evidence. Record uncertainty when the causal link is
unclear.
When development results are available, explain whether the patch generalized
or overfit. Record a specific root cause, explanation, prevention/next action,
and applicable generalization lesson for each case.
## Outputs
Selection: a prepared parent and recorded selection rationale.
Diagnosis/patch: a verified allowlisted change and structured patch intent.
Reflection: evidence-grounded per-case lessons for future iterations.
The controller owns evaluation and final candidate freezing. These sessions
must not tune on the final test set.
```

Figure 13: Executable Harness Optimization Prompt. Separate selection, patching, and reflection sessions use real training executions and development feedback to improve the harness without changing model weights or benchmark evaluators.

Controller-owned operations. Compilation, reference replay, execution resets, validity checks, acceptance, and one-condition deletion tests are not delegated to a model's self-report. Deletion tests reuse the applicable behavioral checks, and no additional "ablation judge" prompt is introduced. Both benchmarks apply the same three-run control and earliest-failure admission rule. They differ only in how a failure is detected, by the typed matcher for GAIA2 and by the semantic judge for Workspace-Bench.

G Limitations and Responsible Use

Oracle availability. The benchmark setting supplies expected actions or checkable obligations. Real user dissatisfaction often does not. A deployment extension would need consented correction, task annotation, or another validated source of oracle behavior. Merely inferring an oracle with an LLM and treating it as ground truth would introduce a new error source not evaluated here.

Privacy and environment access. Do not release private snapshots, trace values, account identifiers, or credentials. Hashing and role replacement are not sufficient against all re-identification risks; workflow structure itself may be identifying. Historical Workspace-Bench generators used privileged source assets. The later restricted-input variants must instead be assessed against their versioned projection, donor/default-state, and admission contracts; they do not inherit a privacy or fidelity guarantee from a historical generation certificate. Credentials are operational secrets, not reproducibility metadata. Execution should use isolated, authorized benchmark accounts and disposable workspaces, not an affected user's live environment.