FROM TRACE TO FIX: RESIDUAL TASK STATE IN LONG-HORIZON AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recovering from failures in long-horizon, stateful language agents is challenging because the action introducing a defect, the symptom exposing it, and the persistent state requiring repair frequently diverge. Existing methods largely focus on retrospective failure attribution, leaving sequential diagnosis and state repair poorly integrated under constrained budgets. We introduce **TRACE–FIX**, a unified recovery framework centered on versioned residual task state. TRACE transforms execution logs into a structured dependency-linked ledger, prioritizing unresolved task requirements rather than salient surface errors. FIX formulates recovery as budgeted planning over admissible actions, incorporating outcome forecasting and explicit repair-reserve constraints to prevent premature budget exhaustion. To rigorously decouple localization accuracy from downstream repair capability, we present **LongH-Debug**, evaluating root-cause attribution, verified-fault recovery, and end-to-end closed-loop execution independently. Our empirical analysis demonstrates that structured residual tracking substantially improves causal localization over raw-trace baselines, while budget-aware sequential planning converts verified fault signals into more reliable task recovery without destructive trial-and-error.

1 INTRODUCTION

Language-model agents increasingly complete tasks by changing persistent state: they write files, update databases, invoke tools, create intermediate artifacts, and make decisions that later steps depend on (Dong et al., 2026; Zhou et al., 2024; Xie et al., 2024; Jimenez et al., 2024). In such tasks, the action that introduces a problem need not coincide with the step where the problem becomes visible. The environment can also change again before debugging begins. As a result, the historical cause, the observed symptom, and the state that currently needs intervention can be different.

A retained database-recovery run illustrates the distinction. A successful SQLite operation silently removes the write-ahead log that contains records needed later in the task. Subsequent commands continue normally, the actor eventually recognizes the mechanism, and the final failure occurs long after the destructive state transition. At recovery time, replaying the final error or blaming the latest suspicious command does not identify what information was lost; restarting from scratch would also discard work that remains valid. The useful recovery question is instead: *which task requirement is still unsupported, which historical state change is relevant to it, and what affordable action can improve the task from here?* The appendix provides the command-level reconstruction and retained receipts for this case.

Prior work has made substantial progress on failure attribution, critical-step localization, counterfactual replay, and intervention-based verification (Zhang et al., 2025b; Chen et al., 2026; Ma et al., 2025; Liu et al., 2026; Zhang et al., 2025a; Qiao et al., 2026). These methods primarily organize evidence about *what went wrong*. Recovery adds a sequential decision problem: additional diagnosis consumes time and intervention budget, repair can invalidate earlier evidence, and the best next action depends on what will remain possible after its outcome. A more complete diagnosis is therefore not automatically a better recovery decision.

We address this problem with two components over a shared runtime state. **TRACE** represents current task requirements together with versioned execution evidence and ranks historical events that can explain requirements that remain unresolved. **FIX** plans over the legal checks, recovery actions, actor continuation, and submission choices available at the current boundary. Its short-horizon outcome

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.

A Delayed state mutation *db-wal-recovery*

live run, 97 tool calls

0 · `ls -la /app: base 8192 B + log 16512 B`
2 × `sqlite3 main.db ... PRAGMA wal_checkpoint: log deleted`
9 · `cp main.db-wal /tmp/...bak (already too late)`
38 × `the same read deletes the rebuilt log again`
95 · `the actor states the mechanism itself`
end × `timeout, reward 0.0`

A successful read removes the only carrier of the six missing records. Nothing errors, so the run repeatedly rebuilds and destroys the log.

B Residual obligation *openssl-selfsigned-cert*

live run, 7 checks

1 × `script-runs: /app/check_cert.py exits 1`
2 ✓ `verification-file: subject and fingerprint`
3 ✓ `key-attributes: 2048-bit RSA, mode 600`
4 ✓ `script-runs re-run: the same predicate passes`
5 ✓ `combined-pem parses as a certificate`
6 ✓ `ssl-directory-layout holds the deliverables`
7 ✓ `certificate-attributes: subject, 365-day validity`
end ✓ `reward 1.0`

The actor rewrites the script after the first failure; the same predicate then passes and the task reaches reward 1.0.

C Blame is not a current fault *bn-fit-modify*

live run, 8 checks

0 ✓ `learned_dag.csv exists in to,from format`
1 ✓ `intervened DAG exists in to,from format`
2 ✓ `intervened sample keeps the column names`
3 ✓ `the DAG has six total edges`
4 ✓ `intervened edges are a subset of the learned DAG`
5–7 ✓ `repeated checks of the same conditions`
end ✓ `8/8 supported, reward 1.0, 0 repairs`

The history contains a failure, but the live state already satisfies every check. Historical blame is therefore not a current repair target.

D Budgeted recovery *cancel-async-tasks*

live run, 8 checks, 2 repairs

1–2 ✓ `import-form and signature supported`
3 × `concurrent-execution refuted`
4–6 ✓ `max-concurrency, cancel-below, cancel-at supported`
7 × `cancel-above-limit refuted`
8 ✓ `import-form re-run: supported`
end × `2 repairs tried, reward 0.0`

Eight checks and two repairs exhaust the twelve-unit budget before either repaired predicate is re-verified.

Figure 1: **Four long-horizon failure modes from retained Terminal-Bench trajectories.** (A) A successful operation silently destroys state whose loss appears only later. (B) A failed predicate is subsequently repaired, leaving only its residual consequences relevant. (C) A historical failure remains in the trajectory although the current state is already healthy. (D) Checks and repairs share a finite budget, which can be exhausted before repair verification. Rows quote retained commands or registered checks; ✓ and × denote supported and refuted outcomes. Receipts are listed in Appendix C.

estimates are supplied from public runtime state, but action identity, costs, state transitions, and the dynamic-programming backup are enforced by runtime code. Every executed action produces a receipt; resource changes invalidate version-bound evidence, the residual state is recompiled, and the next decision is made from the new state.

We also introduce **LongH-Debug**, which separates retrospective localization from executable recovery. Localization is evaluated under matched public inputs; recovery methods are compared under the same independently verified fault packet; and a final factorial crossing tests whether the localization and recovery components compose end to end. This separation avoids crediting a fixer for hidden re-diagnosis or penalizing it for an upstream localization error.

Contributions. First, we define a versioned residual task representation that distinguishes current unsupported requirements from historical events that no longer require intervention. Second, we connect this representation to a practical recovery controller that combines public-state outcome forecasting with runtime-checked bounded planning and replanning after real execution. Third, LongH-Debug provides matched localization, verified-fault recovery, and closed-loop evaluation

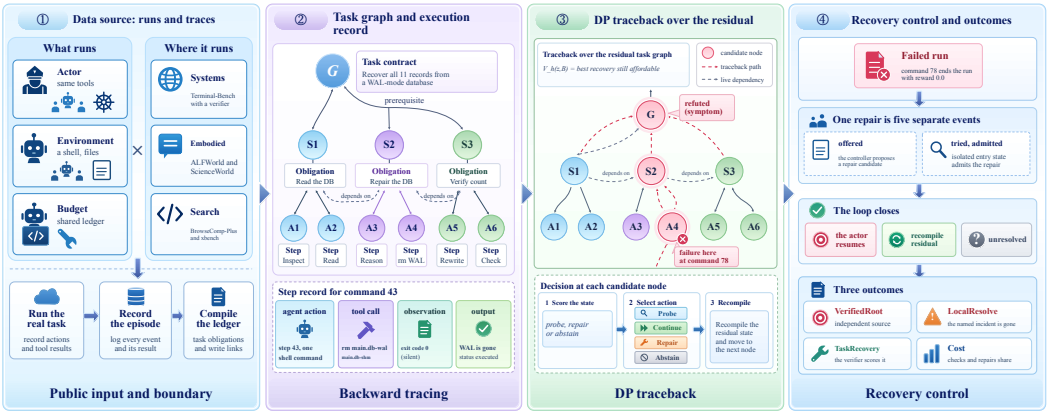

Figure 2: **Runtime path from public history to recovery.** The task contract, public execution record, resource versions, and registered checks are compiled into a residual state. TRACE uses the resulting dependency-linked ledger to rank historical events relevant to unresolved requirements. FIX receives the live action slate and a bounded public-state forecast, while runtime code checks action identity, cost, reachability, and time before computing the depth-two decision. The selected action is then executed; its receipt updates resource versions and evidence, the residual state is recompiled, and planning repeats before the same actor continues. Hypothetical planning outcomes are kept separate from observed execution receipts throughout.

protocols so that attribution quality and recovery quality can be measured separately before they are composed.

2 METHOD

Figure 2 summarizes the runtime path. The key design choice is to treat the execution history as evidence used to construct the current recovery state, rather than as the state itself.

2.1 VERSIONED RESIDUAL TASK STATE

The runtime maintains three linked views with different semantics. The *task view* contains public requirements, their prerequisite relationships, and required outputs. The *execution ledger* contains ordered actions, observations, resource interactions, and resource versions. The *acceptance view* contains task predicates together with the evidence and incidents that currently support or refute them. These references make a historical event relevant to a task requirement without treating a dependency or last-writer edge as causal proof (Weiser, 1981; Ferrante et al., 1987; Agrawal & Horgan, 1990).

Evidence is version-bound. When evidence is recorded for a proposition, the runtime stores the versions of the resources on which that evidence depends. If a later observation changes any of those resources, the evidence is marked *stale*; task revisions supported only by stale evidence are excluded when the live task view is rebuilt. We call a requirement a *residual obligation* when it is not currently supported, is blocked by an unresolved prerequisite, or has lost its support through such a version change.

The recovery state consists of these residual obligations, their current evidence, the historical events linked to them, and the legal action slate. We keep the remaining intervention budget separate from this state representation; a planner decision is made from the compiled state together with the remaining budget. This avoids counting budget both inside a state transition and again in the planning recursion.

2.2 TRACE: STRUCTURED HISTORY, EXPLICIT RANKING

TRACE first compiles the public execution into a residual ledger. Each event retains its identity and order and is annotated mechanically with its actor or tool summary, observed result, referenced resources, explicit write targets, and upstream writer links. The task contract and current residual obligations provide the context in which these event-level effects are interpreted.

The current ranking stage is explicit rather than implicit in the graph construction. An audited LLM call receives the public task specification, the observed failure, and the compiled residual ledger and returns the top historical event IDs, ranked from most to least likely to have created an unresolved task condition. The prompt asks the ranker to distinguish state-changing writers from later observations, tests, or reports of the failure, and to consider omitted or misordered actions when no explicit harmful write explains the outcome. The ledger therefore structures the evidence presented to the ranker; it does not deterministically declare the root cause.

TRACE writes its result to an immutable *fault packet* containing the ranked event hypotheses, their public evidence references, the residual obligations they may affect, and qualified checks that can test them. Verification remains separate from ranking: a high-ranked event is a localization hypothesis until an admissible check, replay, or intervention supports it.

2.3 FIX: BOUNDED PLANNING OVER THE LIVE ACTION SLATE

FIX does not operate over a fixed four-action alphabet. At each recovery boundary the runtime emits the actions that are actually legal for that state, with stable action identities, target bindings, and declared costs. Depending on the environment, the slate can include actor continuation, registered checks, task- or resource-targeted recovery requests, additional diagnosis or completion requests, and submission of the current state.

The current planner uses a separate no-tool LLM as a *forecast model*. It receives only public runtime state: the task graph, residual predicate states, registered checks, a bounded window of recent actor execution summaries, the live candidate catalogue, remaining time, and remaining intervention budget. For each root action it predicts bounded numeric outcomes; for checks these include masses for *supported*, *refuted*, and *unknown* together with estimates of what the same actor can accomplish after each verdict. It may additionally expand at most two reachable second-step actions. Terminal leaves carry a compact estimate of task success and the fraction of the residual problem expected to remain unblocked.

These forecasts are planning estimates, not calibrated probabilities. Runtime code validates their schema and numeric ranges, derives outcome semantics from the action type, rejects transitions that create time or budget, and compiles the accepted forecast into a depth-two decision tree. Standard dynamic programming then compares the expected downstream recovery of the affordable candidates. The model never outputs the selected action or a Bellman value directly; both are recomputed by the runtime from the admitted forecast. If the forecast is missing or fails an admission gate, the runtime falls back to actor continuation rather than inventing a transition.

The intervention budget is an abstract, frozen cost ledger rather than a token or dollar measure. In the evaluation contract, each declared TRACE, FIX-controller, or actor-continuation model call costs one unit; executable checks and repairs use the costs frozen in the environment's qualification catalogue. Tokens, wall-clock time, and monetary cost are reported separately. This shared ledger makes "inspect more" and "preserve budget for repair" directly comparable.

2.4 EXECUTION, RECOMPILATION, AND HANDOFF

Planning ends when an action is selected; recovery evidence begins only after that action is executed. The execution receipt records the observed result and the resource versions at the resulting boundary. Any changed resource invalidates evidence bound to its previous version, incidents and predicate states are updated, and the residual task is recompiled before another decision is allowed.

A local intervention is not treated as task success. After its postcondition is checked, control returns to the same actor when continuation is legal; the benchmark's final verifier scores the state produced

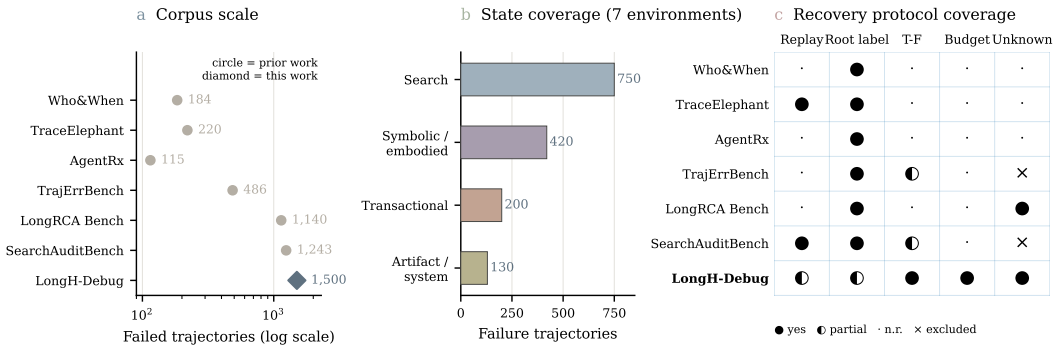

Figure 3: **LongH-Debug benchmark: scale, state diversity, and executable recovery coverage.** (a) Published failure-trajectory counts for prior agent-debugging benchmarks, with the LongH-Debug row shown separately. (b) The 1,500-trajectory corpus spans seven environments grouped into search/retrieval, symbolic/embodied, transactional/social, and persistent artifact/system state. (c) Protocol coverage distinguishes retrospective attribution from executable recovery. Filled circles denote explicitly supported capabilities, half-filled circles partial or alternative protocols, dots unreported capabilities, and crosses explicit exclusions. LongH-Debug's replay and labels apply to its qualified subsets rather than to every trajectory in the corpus.

by that continuation. The formal recursion, planner error analysis, complete repair lifecycle, and durable receipt schemas are given in the appendix.

3 LongH-Debug Benchmark

LongH-Debug is organized in two tiers so that corpus scale is not confused with recovery capability. A *failure record* contains a public task specification, the actor-visible trajectory, and the public failure signal needed for localization. A *recovery-qualified case* additionally requires a reproducible failed entry state, the task/resource/version records needed to reconstruct residual obligations, a frozen catalogue of executable checks and repairs, a common cost ledger, and an independent final verifier. Only this qualified subset is used for experiments that execute recovery actions.

A final characterization to run before submission is a per-case causal-complexity audit. On the localization population we will report the distributions of trajectory length, supported dependency depth H_c, and feedback delay H_f, together with their pairwise rank correlations. On the recovery population we will additionally report resources touched, version-changing writes, residual obligations, and legal actions at the first recovery boundary. The purpose is to show directly whether long transcripts, deep dependencies, delayed evidence, and stateful recovery complexity are distinct axes rather than assuming that trajectory length stands in for all of them.

Source environments. The corpus draws natural failures from seven environments with different state mechanisms. Terminal-Bench (Merrill et al., 2026) supplies persistent filesystem and artifact state; τ^2-bench (Yao et al., 2024) supplies transactional state across tool-mediated conversations; ALFWorld (Shridhar et al., 2021), ScienceWorld (Wang et al., 2022), and TextWorld (Côté et al., 2019) supply symbolic embodied state; and BrowseComp-Plus (Chen et al., 2025b) and xbench (Chen et al., 2025a) supply long search trajectories. We preserve each environment's native task semantics rather than converting them into a shared synthetic state space.

The target corpus contains 1,500 natural failure trajectories, with each environment quota bounded by executions that the source environment can actually provide.

Long-horizon annotations. We do not use transcript length as the definition of a long-horizon failure. For localization cases with supported dependency annotations, we record two separate axes. *Dependency depth* H_c counts the supported dependency chain connecting a decisive historical event to the residual obligation it blocks. *Feedback delay* H_f counts the decisions between that event and the first public evidence that reveals its consequence. The localization population is stratified by these

Table 1: **TRACE localization on $N = 200$ paired natural failures.** Candidate recall asks whether the adjudicated event is present before ranking; VERIFIEDROOT@B requires an independent qualified check sequence within the verification budget. Token counts are total model input tokens over the paired population and relative cost is normalized to Generic one-shot.

Method	Hit@1	Hit@3	MRR	Cand. recall	VERIFIEDROOT@B	Tok. (M)	Rel. cost
Generic one-shot	42.0	74.5	.571	93.0	31.0	2.40	1.00
Dependency-only	24.0	46.0	.351	64.0	15.5	0.52	0.22
Who&When (adapted)	8.0	12.0	.097	17.0	4.5	6.00	2.50
TraceElephant (adapted)	44.0	61.0	.508	71.0	33.0	12.20	5.08
Residual Context	43.0	74.5	.573	93.0	32.0	7.80	3.25
OUR TRACE	**68.0**	**89.0**	**.782**	**97.0**	**60.5**	3.10	1.29

Table 2: **Recovery under matched verified-fault input ($N = 120$).** TASKRECOVERY is the benchmark-owned outcome after same-actor continuation; LOCALRESOLVE asks whether the targeted live incident is gone. Harmful counts unnecessary or state-damaging interventions. Cost/success is normalized to Same-Actor Continue.

Fixer	TASKRECOVERY	LOCALRESOLVE	Harmful	Cost/success
Same-Actor Continue	35.0	29.2	0.0	1.00
Fresh Fixer	42.5	40.8	5.8	0.92
Fixed-order sweep	47.5	50.0	4.2	0.86
Information gain	50.0	53.3	4.2	0.82
Information gain + repair reserve	61.7	66.7	3.3	0.69
Residual-Greedy	56.7	61.7	3.3	0.73
OUR FIX depth-2	**67.5**	**73.3**	**2.5**	**0.61**

axes, allowing a long transcript with a local fault to be distinguished from a failure with genuinely distant dependencies or delayed evidence.

Evaluation populations and outcomes. LongH-Debug defines three matched populations. TRACE is evaluated on $N = 200$ task-disjoint natural failure records under identical public inputs. FIX is evaluated on $N = 120$ recovery-qualified cases where every fixer receives the same independently verified fault packet. The full TRACE$\times$FIX crossing uses $N = 60$ qualified tasks restored from the same failed entry state and scored by the same official verifier.

The protocol records three different outcomes. VERIFIEDROOT asks whether an emitted historical hypothesis can be independently supported within the allowed verification budget. LOCALRESOLVE asks whether the targeted current incident is gone after intervention and continuation. TASKRECOVERY is the final benchmark-owned task outcome. These quantities are intentionally not collapsed into one score: locating a historical cause, removing a current blocker, and finishing the original task are different events.

Historical annotations and root-verification evidence remain evaluator-side. For paired FIX comparisons, the admitted fault packet is frozen and shared unchanged across methods. All launched recovery attempts remain in the denominator, including budget exhaustion and *unknown* check outcomes. The complete qualification rules, baseline adapters, action catalogues, receipt schemas, and provenance checks are given in Appendix E.

4 EXPERIMENTS

We organize the evaluation around four questions. **RQ1** asks whether TRACE localizes historical events that remain relevant to the current task, rather than merely retrieving a salient failure step. **RQ2** gives every fixer the same independently verified fault packet and asks whether FIX converts identical fault information into better recovery. **RQ3** crosses tracing and fixing components on the same failed entry states to test whether the two gains survive end to end. **RQ4** isolates the state, planning, and recompilation mechanisms under matched information and action sets.

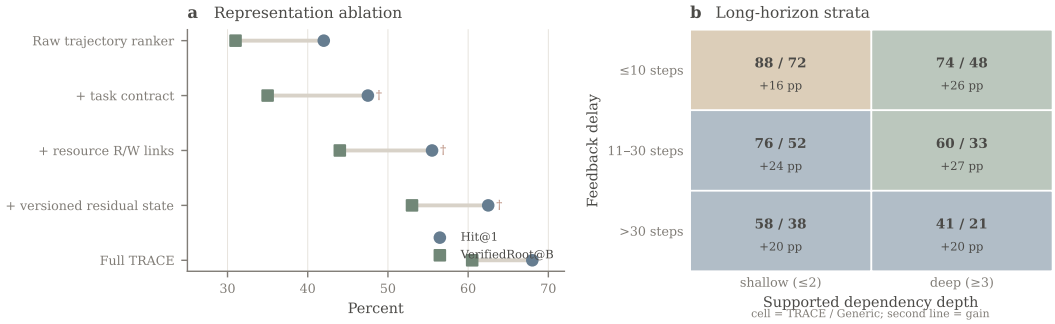

Figure 4: **Why TRACE should help.** (a) Representation ablation. Daggered intermediate rows are unexecuted planning targets; the endpoints are the current target values. The decisive test is whether task structure, resource links, and versioned residual state improve both exact localization and executable root verification, rather than merely increasing retrieved context. (b) TRACE/Generic Hit@1 by supported dependency depth and feedback delay. Each cell shows TRACE / Generic and the gain below.

4.1 Setup and comparison contract

The three primary populations are those defined in Section 3: $N = 200$ paired localization cases, $N = 120$ verified-fault recovery cases, and $N = 60$ closed-loop tasks. Within a comparison cell we freeze the public trajectory or failed entry state, actor identity, model configuration, legal tools, action catalogue, cost ledger, and final verifier.

For localization, *Generic one-shot* is a direct causal ranker over the public trajectory. *Dependency-only* uses the structured dependency slice without the residual-causal ranking prompt. *Residual Context* gives a ranker the corresponding residual information as additional flat context. Who&When is run through its pinned upstream attribution code after projecting each public LongH event into one history entry with its public actor/component label and leaving the upstream ground-truth field empty. TraceElephant retains its upstream prompt, search, fallback, and parsing code, while canonical public events are reconstructed into the structured request/response fields expected by the released implementation; public evaluator failure is mapped into its test-status input and ground truth remains empty.

For recovery, all FIX methods receive the same independently verified fault packet. We compare against *Same-Actor Continue*, a *Fresh Fixer*, a fixed-order check sweep, one-step residual-greedy control, and a strong information-gain policy with an explicit repair-budget reserve. The last comparator is important: controlled development experiments found that repair-budget reservation, not Bellman recursion alone, explained the observed gain over naive information gain. The formal comparison therefore does not use a deliberately budget-oblivious policy as the strongest baseline.

4.2 RQ1: Does TRACE localize the historical event that still matters?

Table 1 evaluates every tracer on identical public inputs. We report ranking quality, candidate coverage, executable verification, and inference cost together; a localization method that retrieves the correct event only through substantially more context should not be read as an unqualified improvement. The headline is not just a Hit@1 gain. TRACE is intended to preserve high candidate recall while converting that coverage into a substantially higher fraction of hypotheses that can be independently verified inside the same downstream budget. Figure 4 separates the representation contribution from the long-horizon difficulty axes.

Historical evidence informing this design. On six controlled hidden-root cases, dependency slicing recovered the root in all six top-three sets while the generic and Who&When-style controls recovered only one; on four retrospective natural development trajectories, dependency candidate construction recalled the verified root in all four and blind dependency ranking reached Hit@1 on three, whereas a blind full-trace ranker reached Hit@1 on none. These small development cells motivate the structured representation ablation but are not pooled with the $N = 200$ estimate.

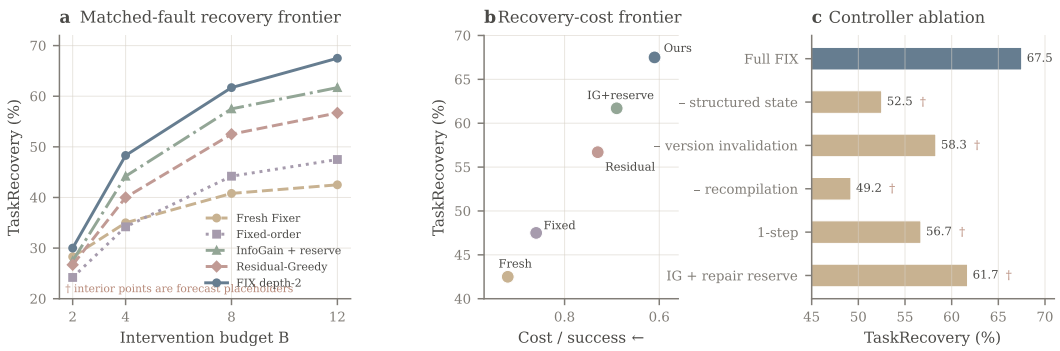

Figure 5: **FIX recovery, cost, and mechanism.** (a) TASKRECOVERY as intervention budget increases under identical verified-fault input. Interior points are planning placeholders until the full budget sweep is executed. (b) Recovery–cost frontier at the largest budget. (c) Mechanism ablation under matched information and legal actions. The strongest planning control is information gain with a repair reserve, not a budget-oblivious strawman.

4.3 RQ2: CAN FIX RECOVER BETTER FROM THE SAME FAULT INFORMATION?

Table 2 removes TRACE as a confound by supplying every method with the same independently verified fault packet, failed state, legal tools, and intervention budget. The strongest non-DP comparator is information gain with an explicit repair reserve, because controlled development showed that this policy can match dynamic planning when the main difficulty is simply preserving enough budget to finish the repair. Figure 5 makes two comparisons that the endpoint table alone cannot. First, a recovery method should dominate over a range of budgets, not only at $B = 12$. Second, a controller that succeeds only by spending more should not be conflated with one that reaches a better recovery–cost frontier.

Why these controls are necessary. In a 24-case executable controlled verification study, naive adaptive information gain resolved $8/24$ cases at budget four, while information gain with repair reservation and dynamic TRACE-DP each resolved $16/24$. That result says the controlled gain was explained by reserving completion budget and does *not* establish a unique Bellman-planning advantage. The natural matched-fault experiment above is therefore the test that can separate residual-state planning from this strong alternative.

4.4 RQ3: DO TRACE AND FIX COMPOSE IN A REAL CLOSED LOOP?

Component improvements are insufficient if the TRACE packet does not survive into executable recovery. We therefore run all four combinations $T_0F_0, T_0F_1, T_1F_0, T_1F_1$ from the same failed entry state with the same actor, legal tools, budget, stopping rule, and official verifier. The target matrix is 31.7% for T_0F_0, 45.0% for T_0F_1, 43.3% for T_1F_0, and 66.7% for T_1F_1, corresponding to a $+35.0$ point full effect and a $+10.0$ point interaction. The environment split is important because a single aggregate can otherwise hide a recovery mechanism that works only for one kind of state.

4.5 RQ4: WHICH PARTS OF THE RECOVERY CONTROLLER MATTER?

The primary ablations preserve the actor, public information, legal action catalogue, base forecast source, and intervention budget. Each row removes one decision-relevant mechanism rather than replacing the method with an unrelated weak agent. Three additional diagnostics belong in the appendix but are required before a strong planning claim is made. First, we measure check-verdict calibration and actor-continuation calibration of the public-state forecast model on held-out executed transitions. Second, we compare the selected action against an oracle-outcome upper bound and report action-ranking regret rather than treating forecast probabilities as calibrated by construction.

Third, we report the fraction of decisions that enter an actionable check/repair path, the share of executed checks returning *unknown*, and the stage at which each failed

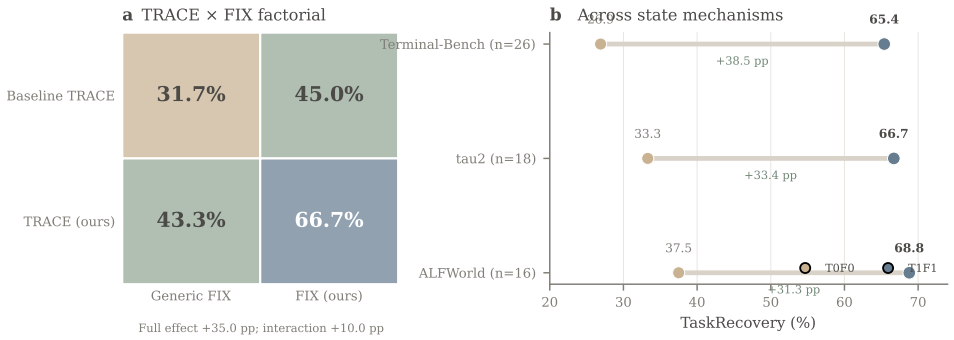

Figure 6: **From diagnosis to task recovery.** (a) The 2×2 TRACE×FIX crossing reports benchmark-owned TASKRECOVERY. Replacing only TRACE or only FIX isolates each component; the diagonal gives the end-to-end system. (b) The same extreme cells within persistent-artifact, transactional, and symbolic-embodied state mechanisms.

recovery terminates. These diagnostics are motivated by retained runs in which one controller spent budget on three unknown checks while another issued 51 consecutive actor-continuation decisions without entering recovery.

Table 3: **Controller ablations under matched verified-fault input.** The information-gain+reserve row is also a strong alternative policy, not merely an ablation.

Controller	TASKRECOVERY	Harmful	Cost/succ.
Full FIX	**67.5**	**2.5**	**0.61**
No structured residual state	52.5	5.0	0.78
No version invalidation	58.3	5.8	0.74
No receipt-triggered recompilation	49.2	6.7	0.82
One-step policy, same forecast	56.7	3.3	0.73
Information gain + repair reserve	61.7	3.3	0.69
Shuffled outcome forecasts	48.3	5.8	0.85

Experience adaptation. The optional cross-case outcome adaptation is not part of the primary experiment chain. Historical guarded development removed negative transfer but produced no uplift over the non-memory policy. We therefore report coverage, abstention, held-out calibration, and any eventual recovery effect separately in the appendix rather than using this component to support the TRACE or FIX claims.

5 LIMITATIONS

TRACE–FIX has several boundaries. First, the residual state is an approximation: missing dependencies, unobserved side effects, or incorrect resource links can exclude relevant events; dependency links narrow hypotheses without establishing causality. Second, FIX relies on bounded lookahead with uncalibrated forecasts, where truncation and estimation errors can affect action selection. Third, recovery requires reproducible failed states, a frozen check-and-repair catalogue, and an independent verifier, so results may not transfer to open-world settings without replay or known repair actions. Finally, recovery value depends on the continuation actor and cost model; we claim neither optimal control nor unique causal identification.

6 CONCLUSION

Diagnosing and recovering from long-horizon agent failures requires decoupling root-cause localization, budgeted state repair, and end-to-end execution. We formalize this separation through TRACE-FIX: TRACE represents public history as an explicit residual ledger to prioritize unsatisfied task requirements, while FIX conducts bounded dynamic programming over verified public states and runtime execution receipts. Across matched benchmarks and factorial evaluations, structured residual tracking substantially improves root-cause localization, targeted planning enhances recovery yield under equivalent fault packets, and their closed-loop integration outperforms either component in isolation. By converting historical failure explanations into actionable residual obligations, TRACE-FIX provides a principled framework for reliable, budgeted recovery in persistent stateful environments.

AI USE STATEMENT

During the preparation of this work, we solely use ai to improve language readability and assist with text polishing and grammar correction. After using this tool, the author(s) reviewed and edited the content as needed and take full responsibility for the integrity and accuracy of the publication. The tool was not used to generate novel research ideas, conduct data analysis, or draw scientific conclusions.

ETHICS STATEMENT

The authors confirm that this study does not involve human participants, animal experimentation, or sensitive personal data. Consequently, ethical approval and informed consent were not required for this work.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, all code, benchmark assets, and execution receipts for TRACE-FIX are provided in the supplementary material and will be publicly released upon publication. The curation details of the LongH-Debug benchmark, environment adapters, and fault-packet schemas are documented in Section 3 and Appendices D and E. Full experimental configurations, including prompt templates, intervention budget ledgers, and dynamic programming depth, are detailed in Section 4 and Appendix E. Theoretical derivations and proofs for residual-state sufficiency are provided in Appendix B.

REFERENCES

Hiralal Agrawal and Joseph R. Horgan. Dynamic program slicing. In *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation (PLDI)*, pp. 246–256, 1990. doi: 10.1145/93542.93576.

Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1806.07857.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI Conference on Artificial Intelligence*, 2017. URL https://arxiv.org/abs/1609.05140.

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

Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Haotong Tian, Tianbao Xie, Fangfu Liu, Haoye Zhang, Hongzhang Liu, Yuan Gong, Chen Sun, Han Hou, Hui Yang, James Pan, Jianan Lou, Jiayi Mao, Jizheng Liu, Jinpeng Li, Kangyi Liu, Kenkun Liu, Rui Wang, Run Li, Tong Niu, Wenlong Zhang, Wenqi Yan, Xuanzheng Wang, Yuchen Zhang, Yi-Hsin Hung, Yuan Jiang, Zexuan Liu, Zihan Yin, Zijian Ma, and Zhiwen Mo. xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations, 2025a. URL https://arxiv.org/abs/2506.13651.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems, 2026. URL https://arxiv.org/abs/2604.22708.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent, 2025b. URL https://arxiv.org/abs/2508.06600.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games, 2019. URL https://arxiv.org/abs/1806.11532.

Guanting Dong, Xiaoshuai Song, Yuyang Hu, Jiajie Jin, Chenghao Zhang, Yifei Chen, Xiaoxi Li, Huaying Yuan, Xinyu Yang, Tongyu Wen, Jiejun Tan, Hongjin Qian, Shijue Huang, Junting Lu, Zhenyu Li, Wanjun Zhong, Yutao Zhu, Tat-Seng Chua, Zhicheng Dou, and Ji-Rong Wen. Towards long-horizon agents: A survey. *Preprints*, 2026. doi: 10.20944/preprints202607.1328.v1.

Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987. doi: 10.1145/24039.24041.

Nandiraju Gireesh, Yuanliang Ju, and He Wang. Adaptive q-chunking for offline-to-online reinforcement learning, 2026. URL https://arxiv.org/abs/2605.05544.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing, 2024. URL https://arxiv.org/abs/2305.11738.

David Heckerman, John S. Breese, and Koos Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–57, 1995. doi: 10.1145/203330.203341.

Ronald Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966. doi: 10.1109/TSSC.1966.300074.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2006. URL http://rbr.cs.umass.edu/aimath06/proceedings/P21.pdf.

Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking, 2026a. URL https://arxiv.org/abs/2507.07969.

Yibo Li, Zijie Lin, Ailin Deng, Xuan Zhang, Yufei He, Shuo Ji, Tri Cao, and Bryan Hooi. Just-in-time reinforcement learning: Continual learning in llm agents without gradient updates, 2026b. URL https://arxiv.org/abs/2601.18510.

Zhixiang Liang, Yifei Liu, Yidan Huang, Haozhe Zhao, Beichen Huang, Jiaqi Wang, Nan Duan, and Qiong Cao. Searchauditor: Auditing and attributing failures in long-horizon search agents, 2026. URL https://arxiv.org/abs/2608.05212.

Jiale Liu, Huajun Xi, Shaokun Zhang, Yifan Zeng, Tianwei Yue, Chi Wang, Jian Kang, Qingyun Wu, and Huazheng Wang. Who&when pro: Can llms really attribute failures in ai agents?, 2026. URL https://arxiv.org/abs/2607.09996.

Kui Liu, Anil Koyuncu, Tegawendé F. Bissyandé, Dongsun Kim, Jacques Klein, and Yves Le Traon. You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pp. 102–113, 2019. doi: 10.1109/ICST.2019.00020.

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, 2025. URL https://arxiv.org/abs/2512.06749.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Martin Monperrus. Automatic software repair: A bibliography. *ACM Computing Surveys*, 51(1): 1–24, 2018. doi: 10.1145/3105906.

Roger B. Nelsen. *An Introduction to Copulas*. Springer, 2nd edition, 2006. doi: 10.1007/0-387-28678-0.

Jiangweizhi Peng, Yuanxin Liu, Ruida Zhou, Charles Fleming, Zhaoran Wang, Alfredo Garcia, and Mingyi Hong. Hiper: Hierarchical reinforcement learning with explicit credit assignment for large language model agents, 2026. URL https://arxiv.org/abs/2602.16165.

Yunjia Qi, Zehua Yin, Xintong Shi, Hao Peng, Songyuanyi Lu, Yixian Liu, Richeng Xuan, Yuhong Liu, Zhichao Hu, Xiaozhi Wang, Lei Hou, Bin Xu, and Juanzi Li. Trajdebug: Tracing error lifecycle to identify critical failures in long-horizon agent trajectories, 2026. URL https://arxiv.org/abs/2608.06346.

Hezhe Qiao, Hanghang Tong, Ee-Peng Lim, Bing Liu, and Guansong Pang. Verifymas: Hypothesis verification for failure attribution in llm multi-agent systems, 2026. URL https://arxiv.org/abs/2605.17467.

Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49(1-3):361–395, 1991. doi: 10.1016/0004-3702(91)90015-C.

C. Nicolò De Sabbata, Theodore R. Sumers, Badr AlKhamissi, Antoine Bosselut, and Thomas L. Griffiths. Rational metareasoning for large language models, 2025. URL https://arxiv.org/abs/2410.05563.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2303.11366.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.03768.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Cote, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Conference on Empirical Methods in Natural Language Processing*, 2022. URL https://arxiv.org/abs/2203.07540.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL https://arxiv.org/abs/2409.07429.

Mark Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, pp. 439–449. IEEE Press, 1981.

Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. Backtrackagent: Enhancing gui agent with error detection and backtracking mechanism, 2025. URL https://arxiv.org/abs/2505.20660.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023a. URL https://arxiv.org/abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2):183–200, 2002. doi: 10.1109/32.988498.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems?, 2025a. URL https://arxiv.org/abs/2509.03312.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025b. URL https://arxiv.org/abs/2505.00212.

Yunfei Zhang, Boyu Feng, Changhua Pei, Zexin Wang, Zhihuang Peng, Xinlong Liu, Hengyue Jiang, Difeng Ma, Jiayi Zhang, Yongzhou Yao, Yanan Zhao, Fei Sun, Yintong Huo, Zhaoyang Liu, Jingjing Li, Gaogang Xie, and Dan Pei. Longrca bench: Root-cause localization in long-horizon agent trajectories, 2026. URL https://arxiv.org/abs/2608.15242.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models, 2023. URL https://arxiv.org/abs/2310.04406.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

A Related Work

From attribution to intervention. Who&When defines agent and decisive-step attribution in multi-agent logs (Zhang et al., 2025b), while TraceElephant studies component and critical-step attribution with richer execution context (Chen et al., 2026). Recent work extends this space through execution-constraint diagnosis and error-lifecycle tracing (Barke et al., 2026; Qi et al., 2026), long-horizon root-cause localization and search-agent auditing (Zhang et al., 2026; Liang et al., 2026), controlled fault injection and counterfactual replay (Liu et al., 2026; Zhang et al., 2025a), and explicit hypothesis verification (Qiao et al., 2026). DoVer goes further by testing fault hypotheses through intervention and measuring failure flips or milestone progress (Ma et al., 2025). Our question begins after these capabilities are available: when interventions share a budget and each result changes the task, which one should be executed next? Our distinction is experimental as well as algorithmic: TRACE is scored before repair, FIX is scored under identical verified fault information, and the two are then crossed. This prevents a repair method's hidden re-diagnosis from being counted as FIX quality and prevents a weak fixer from obscuring TRACE quality, a confound closely related to the fault-localization bias documented in automated program repair (Liu et al., 2019).

Agent search and recovery. ReAct interleaves reasoning with environment action (Yao et al., 2023b); Reflexion reuses verbal feedback (Shinn et al., 2023); and LATS searches over agent trajectories (Zhou et al., 2023). Tree of Thoughts makes lookahead and backtracking explicit in language-model inference (Yao et al., 2023a); CRITIC grounds self-correction in external tool feedback (Gou et al., 2024); and BacktrackAgent couples error detection with rollback in GUI agents (Wu et al., 2025). Our Fix instead controls a recovery boundary: it preserves the same actor, exposes checks and repairs as explicit budgeted actions, and recompiles the residual task after their real outcomes. This makes local resolution and official task recovery distinct observables.

Sequential diagnosis and metareasoning. Classical decision-theoretic troubleshooting jointly reasons about diagnostic beliefs, additional observations, repairs, and expected repair cost (Heckerman et al., 1995). More generally, value-of-information and rational metareasoning distinguish the amount of information obtained from the downstream utility of obtaining it under limited resources (Howard, 1966; Russell & Wefald, 1991). Our controller inherits this sequential-decision perspective but applies it to a residual task state whose actions include checks, repairs, and continuation of an LLM agent. The bounded recursion follows the standard dynamic-programming principle (Bellman, 1957) rather than introducing a new Bellman equation. Recent LLM work has likewise used value-of-computation ideas to allocate inference-time reasoning (Sabbata et al., 2025).

Dependence analysis and repair. Program slicing and program-dependence graphs trace computations through relevant data and control dependencies (Weiser, 1981; Ferrante et al., 1987), while dynamic slicing restricts those dependencies to a concrete execution (Agrawal & Horgan, 1990). Delta debugging isolates failure-inducing differences through controlled reruns (Zeller & Hildebrandt, 2002). Automated software repair separately studies behavioral repair and runtime/state repair (Monperrus, 2018), and its evaluation literature shows that repair conclusions can be biased by the upstream fault-localization stage (Liu et al., 2019). TRACE/FIX differs in operating over agent-visible task obligations and tool-mediated state, but these literatures are the clearest technical lineage for dependency tracing, intervention, and the localization–repair boundary.

Temporal credit and reusable experience. RUDDER redistributes delayed reward (Arjona-Medina et al., 2019), option-critic models temporally extended actions (Bacon et al., 2017), and JitRL uses cross-episode experience for gradient-free test-time adaptation (Li et al., 2026b). Agent Workflow Memory instead extracts reusable task workflows from prior agent experience (Wang et al., 2024). Recent work on action chunking and hierarchical credit assignment also addresses long-horizon value propagation by changing the temporal unit of decision making (Li et al., 2026a; Gireesh et al., 2026; Peng et al., 2026). Our optional J mechanism does neither reward redistribution nor actor adaptation. It reuses verified supervisor transitions to estimate what a current action may do; the recovery controller separately decides what those outcomes are worth. J is an enhancement, not a prerequisite for either primary claim.

B FORMALIZATION AND CONDITIONAL GUARANTEES

This appendix formalizes the two operations that TRACE–FIX keeps separate: retrospective construction of fault candidates and prospective selection of recovery actions. The results are conditional. They do not establish that the current dependency ledger is complete, that the compiled residual state is sufficient, that the forecast model is calibrated, or that the implemented depth-two planner outperforms a competing policy. Instead, they specify the conditions under which the representation and planning decomposition is well-defined, and identify the approximation errors that the experiments must measure.

B.1 RETROSPECTIVE TRACING AND PROSPECTIVE CONTROL ARE DIFFERENT GRAPHS

Let H_t denote the complete public history at a supervisor boundary and let

$$x_t = \phi(H_t)$$

be the compiled residual task state. It contains the current residual obligations, version-valid evidence, historical event references, and the legal recovery-action catalogue. The remaining intervention budget B_t is kept separate.

TRACE and FIX use this state in opposite temporal directions.

For TRACE, let $\mathcal{R}(x_t)$ denote the set of current residual obligations and let G_{hist} be the public dependency-linked execution ledger. For the dependency-generated candidate subset, define

$$\mathcal{C}_{\text{dep}}(x_t) = \{e : \exists r \in \mathcal{R}(x_t) \text{ such that } e \rightsquigarrow r \text{ in } G_{\text{hist}}\}.$$

This is a backward query over public historical dependencies. It constructs candidate explanations; it does not assign causal truth.

FIX instead expands a prospective decision tree from the current state. A legal supervisor action u produces an outcome y, after which real execution yields a receipt, the public history is extended, and the state is recompiled:

$$H_{t+1} = H_t \oplus \rho_t, \qquad x_{t+1} = \phi(H_{t+1}).$$

Bellman backup applies to this prospective action–outcome tree, not to the historical dependency graph. Backward dependency tracing asks which past events remain relevant; dynamic programming asks which present action has the best downstream recovery value.

Proposition 1 (conditional candidate coverage). Suppose an adjudicated historical event $e^\star$ affects a current residual obligation $r^\star$, and the public ledger contains a directed dependency path

$$e^\star \rightsquigarrow r^\star.$$

If dependency candidate construction includes all historical event ancestors of every current residual obligation, then

$$e^\star \in \mathcal{C}_{\text{dep}}(x_t).$$

Proof sketch. Since $r^\star \in \mathcal{R}(x_t)$ and $e^\star$ is an ancestor of $r^\star$, it satisfies the definition of $\mathcal{C}_{\text{dep}}(x_t)$.

The proposition is deliberately limited. It gives a conditional candidate recall statement, not a ranking guarantee or a causal proof. It also does not cover roots whose relevant state change is absent from the public ledger, omission-only failures with no dependency path, or task requirements missing from the public contract. These cases must be recovered by other candidate sources or remain outside the guarantee.

B.2 RESIDUAL STATE AS A SUFFICIENT STATISTIC FOR RECOVERY

The purpose of the compiler is not to summarize the history, but to preserve the distinctions that can change future recovery decisions.

Consider two public histories H and H' such that

$$\phi(H) = \phi(H') = x.$$

Assume that, for every remaining budget B and every legal supervisor action, the two histories induce the same legal action set, declared action costs, distribution over observable action outcomes, distribution over the next compiled state, and terminal recovery utility. Under these conditions, histories mapped to the same x are future-equivalent for the supervisor.

Proposition 2 (lossless residual-state control). Under the conditions above, for every finite planning horizon there exists an optimal supervisor policy that depends on the public history only through (x, B). In particular, replacing a history-dependent policy by its conditional action distribution given (x, B) preserves its value.

Proof sketch. The result follows by induction over the remaining finite horizon. At the terminal boundary, equal compiled states have equal terminal utility. Assume the claim holds for $h - 1$ remaining decisions. At horizon h, equal compiled states have the same legal actions, costs, outcome distribution, and distribution over next compiled states. Applying the induction hypothesis to each possible next state therefore gives identical action values and hence identical optimal values. This is the standard state-aggregation argument under transition and reward equivalence (Li et al., 2006).

The proposition supplies a design criterion rather than evidence that the current compiler satisfies it.

Why version invalidation is part of the state condition. Suppose evidence e supports proposition p using resource version v, and a later action changes that resource to v'. If the truth of p can vary across these versions, then an abstraction that continues to treat e as current support without recording the version mismatch can merge two histories with different future consequences. Such histories need not have the same legal actions, outcome distributions, or terminal utility, violating the premise of Proposition 2.

Therefore, for version-sensitive propositions, a sufficient residual-state representation must either invalidate old evidence, refresh it, or explicitly retain the version mismatch. Version invalidation is thus not merely an implementation detail; it is one mechanism by which the compiler avoids state aliasing.

B.3 Budgeted recovery as a closed-loop decision process

Let $\mathcal{U}(x, B)$ be the legal supervisor actions that remain affordable at residual state x with budget B. Actions may include registered checks, targeted recovery actions, actor continuation, and submission. Let

$$P(y \mid x, u)$$

denote an ideal outcome model and let

$$T(x, u, y)$$

denote the next compiled residual state after the corresponding outcome has actually been executed and written to the public history. Let $c(x, u, y)$ be the charged intervention cost.

For a bounded recovery utility $G(x, B) \in [0, 1]$, define

$$V_0(x, B) = G(x, B)$$

and, for $h \geq 1$,

$$Q_h(x, B, u) = \sum_y P(y \mid x, u) V_{h-1}\left(T(x, u, y), B - c(x, u, y)\right), \tag{1}$$

$$V_h(x, B) = \max_{u \in \mathcal{U}(x,B)} Q_h(x, B, u). \tag{2}$$

Equation equation 2 is the decision problem approximated by the bounded planner. Importantly, G is a planning utility; it need not be a calibrated probability of the benchmark's final TASKRECOVERY outcome. The benchmark-owned verifier remains a separate measurement.

B.4 The value of a check is the action it enables

A diagnostic check is valuable only insofar as its result changes a useful downstream decision.

Let θ denote an unknown condition relevant to recovery, let $r \in \mathcal{A}$ be a downstream repair choice, and let $U(r, \theta)$ be its utility. Without an additional observation, the best decision has value

$$J_0 = \max_r \mathbb{E}_\theta[U(r, \theta)].$$

Now consider a non-destructive, zero-cost check q that produces observation Y. After observing $Y = y$, the controller may choose a different repair. Its value is

$$J_q = \mathbb{E}_Y \left[\max_r \mathbb{E}[U(r, \theta) \mid Y] \right].$$

Proposition 3 (non-negative value of free information). For a zero-cost observation that does not alter the underlying state,

$$J_q \geq J_0.$$

Proof sketch. For every fixed repair r,

$$\mathbb{E}_Y \left[\max_{r'} \mathbb{E}[U(r', \theta) \mid Y] \right] \geq \mathbb{E}_Y \left[\mathbb{E}[U(r, \theta) \mid Y] \right] = \mathbb{E}[U(r, \theta)].$$

Taking the maximum over r gives the result.

The budgeted setting is different. A real check consumes resources and can remove downstream actions from the affordable set. Its recovery value is therefore determined by

$$Q_h(x, B, q),$$

not by information gain alone. A check that perfectly identifies the historical fault can have zero recovery value if its cost leaves insufficient budget for repair and verification.

This also explains the relation between fixed and adaptive schedules. A fixed schedule is an adaptive policy that deliberately ignores observations, so under an exact shared model the adaptive policy class weakly contains the fixed class. The implemented approximation, however, is not guaranteed to realize this advantage.

Illustrative example. Suppose a direct repair succeeds with probability 0.5. A one-unit check A followed by a contingent repair yields recovery value 0.8, while check C yields 0.7. A complete diagnosis identifies the fault perfectly but consumes so much budget that no repair remains affordable, giving recovery value 0. The ordering

$$0.8 > 0.7 > 0.5 > 0$$

shows why recovery-oriented planning need not maximize diagnostic completeness.

B.5 Approximate forecasts, shallow lookahead, and decision regret

The implemented planner does not know the ideal kernel P or the exact continuation value. Let $\widehat{P}$ be the admitted public-state forecast and let $\widehat{G}$ be the leaf continuation estimate used by an h-step planner.

Assume, for the moment, that the compiled state is sufficient and that all utilities lie in $[0, 1]$. Suppose

$$\sup_{x,u} \mathrm{TV}\left(\widehat{P}(\cdot \mid x, u), P(\cdot \mid x, u) \right) \leq \epsilon_P,$$

and the leaf continuation estimate satisfies

$$\sup_{x,B} \left| \widehat{G}(x, B) - G^\star_{\mathrm{tail}}(x, B) \right| \leq \epsilon_G,$$

where $G^\star_{\mathrm{tail}}$ denotes the true continuation value from a leaf of the truncated tree.

Proposition 4 (finite-horizon forecast error). For the same legal action set and cost semantics,

$$\left\| \widehat{Q}_h - Q^\star_h \right\|_\infty \leq \epsilon_G + h\epsilon_P.$$

Proof sketch. At depth zero the error is at most ϵ_G. At each preceding level, changing the outcome distribution changes the expectation of a $[0,1]$-valued continuation by at most ϵ_P, while the continuation-value error from the next level is inherited unchanged. Induction gives Equation equation B.5.

Let

$$u^\star = \arg\max_u Q_h^\star(x, B, u), \qquad \widehat{u} = \arg\max_u \widehat{Q}_h(x, B, u).$$

Writing

$$\delta_h = \epsilon_G + h\epsilon_P,$$

the standard greedy-decision bound gives

$$Q_h^\star(x, B, u^\star) - Q_h^\star(x, B, \widehat{u}) \le 2\delta_h.$$

Consequently, if the true best-action margin exceeds $2\delta_h$, the estimated planner selects the same action as the ideal h-step planner.

This result clarifies what a depth-two planner does and does not guarantee. Increasing search depth is useful only when the leaf estimate is informative about the continuation beyond the explicit tree. A check→repair→recheck chain can be invisible to a two-step planner if all depth-two leaves are assigned nearly identical continuation values. Thus shallow lookahead requires either an adequate leaf value, semantically composed actions, or additional depth; bounded dynamic programming by itself does not remove truncation error.

If the compiled state is not sufficient, an additional state-aliasing error is present and the bound above does not apply. This is why state ablation, held-out forecast calibration, and action-ranking regret test different failure modes rather than redundant properties.

B.6 Optional outcome-model adaptation

This subsection applies only to the optional cross-case outcome adaptation and is not required for the core TRACE or FIX guarantees.

Let the base outcome estimate be P_0 and an admitted memory-conditioned estimate be

$$P_J(y \mid x, u) = (1 - w)P_0(y \mid x, u) + wP_M(y \mid x, u), \qquad 0 \le w \le w_{\max}.$$

For a fixed downstream continuation value

$$W(y) = V_{h-1}\left(T(x, u, y), B - c(x, u, y)\right),$$

the induced action-value change is

$$\Delta Q_J(x, B, u) = \sum_y \left(P_J(y \mid x, u) - P_0(y \mid x, u)\right) W(y). \tag{3}$$

Therefore

$$|\Delta Q_J(x, B, u)| \le \mathrm{TV}(P_J, P_0)\left(\max_y W(y) - \min_y W(y)\right).$$

Probability movement has no decision value when it only redistributes mass among outcomes with the same downstream value.

More strongly, let u_0 be the best action under the base estimate. Another action u can overtake it only if

$$\Delta Q_J(u) - \Delta Q_J(u_0) > Q_0(u_0) - Q_0(u).$$

Thus an outcome-model correction can improve calibration while leaving every planner decision unchanged. The decision-relevant empirical tests are therefore held-out outcome prediction, action-ranking changes, and recovery or cost improvement, rather than the magnitude or frequency of the correction alone.

B.7 Logical reachability is not a probability model

The task graph represents logical obligations and public dependency structure; it is not assumed to be a Bayesian network. In particular, if an AND obligation has marginal premise probabilities $p_1, \ldots, p_n$ but their dependence is unknown, multiplying the marginals is unjustified. The Fréchet bounds (Nelsen, 2006) give

$$\max\left(0, \sum_{i=1}^{n} p_i - (n-1)\right) \leq P\left(\bigcap_i A_i\right) \leq \min_i p_i.$$

Accordingly, quantities such as supported-obligation fraction or logical reachability may be used as bounded planning utilities without being described as calibrated joint success probabilities.

B.8 What the theory requires the experiments to test

The results above separate four empirical questions.

First, Proposition 1 motivates candidate recall: if the relevant public dependency path is captured, does residual dependency tracing preserve the adjudicated event before ranking?

Second, Proposition 2 motivates the residual-state ablations: do versioning, stale-evidence invalidation, and residual obligations prevent decision-relevant state collisions without giving the method additional private information?

Third, Propositions 3–4 motivate the FIX diagnostics: under identical legal actions and base estimates, does adaptive lookahead improve the recovery–cost frontier, and are the forecast and leaf estimates accurate enough to preserve action rankings?

Finally, Equation equation 3 motivates the optional adaptation analysis: does cross-case information improve held-out consequence prediction and change decisions in a way that improves recovery or cost?

None of these conditional results substitutes for the corresponding paired experiment.

C Implementation Lifecycle and Verification

A formal run is audited from method input through final consumption, and every reported result is read from a retained receipt rather than from a summary. Table 4 states the artifact classes a launch must produce, the upstream boundary each class is checked against, and the condition that closes it. The detailed implementation receipt is distributed with the paper workspace.

C.1 TRACE–FIX boundary

The new locally designed contract has seven separate artifact classes plus canonical raw runtime-source bundles. A result-blind receipt records task eligibility and freezes public event and evidence identifiers, legal check and repair catalogues, the local-resolution check, and the official verifier and threshold before method outputs are opened. A public packet carries ranked hypotheses, public evidence references, residual obligations, and suggested checks. An evaluator-only receipt carries rank and candidate inclusion. A benchmark-owned root-verification audit binds VerifiedRoot@B to one emitted hypothesis, qualified check and execution, predicate, subject versions, source runtime receipt, independent auditor, result, and charged cost. A separate verified-root admission binds the verified-fault condition packet to blind selection and repair-specific same-state three-arm evidence. A cell receipt carries the frozen FIX method and adapter, checks, the repair lifecycle, residual states, actor continuation, cost, and claimed outcomes. Finally, a benchmark-owned outcome evaluation independently audits every executed check, the post-continuation LocalResolve predicate, harmful-repair labels, final score, and TaskRecovery under the frozen threshold. The analyzer reopens the configuration, event stream, observer evidence, and runtime-execution receipt for each TRACE-root, FIX-check, and LocalResolve assertion, then reconstructs its verdict with the runtime auditor. Exact field validation prevents an added gold or method-score field from entering the packet or qualification receipt; identifier catalogues reject proposed or executed actions outside the qualified public input.

Table 4: **Artifact isolation and integrity contract at the TRACE–FIX boundary.** Each row defines an audited runtime artifact, its authoritative writer, required payload, and strictly prohibited fields to prevent data leakage and benchmark gaming.

Artifact	Written by	Carries	Prohibited content (Leakage Guard)
Qualification receipt	Result-blind qualifier	task eligibility, public trace check, re-executable failed entry state, official verifier and threshold, legal action catalogues, local-resolution check, matched cost ledger	method outputs, candidate ranks, or evaluation scores (receipt is frozen prior to method execution)
Public fault packet	TRACE tracer	ranked hypotheses, cited public evidence, residual obligations, qualified checks with declared costs	ground-truth rank, hidden root identifier, evaluator verdict, official answer, private repair oracle
Evaluator receipt	Independent evaluator	rank and candidate inclusion, Hit@1, Hit@3, MRR, candidate recall, VerifiedRoot@B, incurred cost	any field exposed to the public packet; evaluator must not be the actor, model, or recovery adapter
Root-verification audit	Benchmark-owned auditor	emitted hypothesis, qualified check and execution, predicate, subject versions, runtime receipt, independent auditor, outcome, charged cost	self-reported verdict flags or unverified fields copied from method records
Verified-root admission	Evaluator-only admission	blind selection of verified-fault packet, repair-specific same-state three-arm evidence	non-admitted external signals; evaluator must remain isolated from downstream actors
Cell receipt	Recovery runtime	frozen FIX method/adapter, executed checks, repair lifecycle, residual states, continuation traces, stage costs, claimed outcomes	evaluator-private benchmark scores (terminal valuation is restricted to residual proxies)
Official outcome evaluation	Benchmark-owned verifier	audit of executed checks, post-continuation LocalResolve predicate, harmful-repair flags, official score, TaskRecovery	method-authored lifecycle claims or self-reported success statuses

Field, identifier, and raw-source consistency checking is a required launch condition, but it is not by itself a process-level proof that free text cannot carry private information; process-level separation is a separate launch condition.

The frozen manifest names six development candidates, one success-state negative control, five TRACE methods, three FIX methods, four core cells, three conditional verified-root FIX controls, and $R = 1$. Every cell has a 12-unit ceiling under one explicit ledger: declared TRACE, FIX-controller, and continuation model calls are counted separately and each costs one unit; TRACE has an eight-call and one-million-token ceiling with at most 4,096 output tokens per call; and check and repair costs come from the qualification catalogue. The total model-call field must equal the three stage counts, and an invoked fixer must declare at least one controller call. The manifest binds one canonical case to each task slot, freezes Hit@3 and the exact $R = 3$ trigger, is atomically create-only, preserves task slots ruled invalid before method launch, and stays `launch_ready=false` until the launch conditions below are met. The analyzer refuses results from an unqualified task or aliased case; binds completed and blocked TRACE rows to the qualified public trace and failed state; checks that each completed TRACE score points to its validated packet; and then requires that exact packet in FIX. Positive VERIFIEDROOT@B is derived from a registered audit of the packet-suggested check bound to the scored hypothesis and exact spend rather than a self-reported Boolean. Across matrix cells the analyzer freezes the failed state, actor, model, legal tools, FIX adapter, and official verifier; paired generic and residual fixers share one TRACE packet, while Same-Actor, generic, and residual verified-root controls share one admitted packet whose evaluator cannot be the actor, model, verifier, or FIX adapter. Same-Actor cannot execute FIX checks or repairs. Actual actions

must match the qualification-time catalogues. A stable check may be repeated before and after repair, but execution, attempt, sequence, source receipt, and evidence identities are unique; verdicts are restricted to supported, refuted, or unknown. An untried repair has zero execution cost. Each delivered repair requires exactly one later catalogue-defined postcondition recheck in the cell receipt: supported permits continuation; refuted or unknown blocks at the postcondition stage. LocalResolve is derived from the qualified predicate audit rather than from a pre-repair check, and its source trigger identifier must match the receipt's continuation identifier. The first residual snapshot must match the packet, later snapshots must cite fresh executed transition evidence, and every completed cell must consume one independent official-score receipt whose scored-state identifier matches the receipt's continuation output. Packet, run, evaluator, private evidence, and official evidence namespaces are disjoint. These are receipt-level constraints, and a formal launch additionally requires one shared raw timeline that carries repair, postcondition, continuation, LocalResolve, and scoring in order, plus provider-owned usage receipts that rederive every model-call count. The analyzer also rejects budget drift and lifecycle events after a blocked stage, and keeps every launched blocked attempt in the metric denominator with incurred cost. Before launch, an independent two-case arithmetic oracle must verify TRACE $+0.5$, FIX $+1.0$ against both generic and Same-Actor controls, Full $+1.0$, and interaction $+0.5$ without calling the production analysis helper, and the manifest and seven-artifact analysis command-line entries must be exercised through create-only subprocess integration tests.

Schema separation and arithmetic do not by themselves establish that the named upstream methods have equivalent live adapters or that any natural comparison is positive.

C.2 Intended execution chain

The live path begins with a result-blind qualifier, which checks the public trace, the reproducible failure state, the official verifier, and the matched budget, and freezes the public identifier allowlists before any method result is visible. Each tracer then reads the same public request through the unified tracing seam and emits ranked public findings, and a method-specific converter writes the frozen fault packet before the independent evaluator accesses gold; the localization metrics are written to a separate receipt. The dispatcher hands the identical packet to the paired generic and residual FIX cells, while the Same-Actor, generic, and residual verified-root cells additionally require one evaluator-only admission projected from the independent intervention receipt.

The recovery runtime restores the frozen state, compiles the legal checks and repairs, and records the declared TRACE calls, FIX-controller calls, continuation calls, checks, and repairs against the budget. Every executed action identifier and cost must match the qualification-time catalogue, stable action definitions stay separate from unique ordered executions, and formal runs additionally require provider-owned usage receipts. Each check binds its result to the current subject versions, and only supported admission evidence may authorize a live repair. The repair lifecycle keeps offered, tried, admitted, delivered, and rechecked as distinct receipt events, only a tried repair incurs execution cost, and exactly one later receipt-ordered postcondition recheck follows delivery: supported evidence allows continuation, while refuted or unknown evidence blocks at the postcondition stage. A live state change triggers residual recompilation, and the shared raw timeline is required before formal launch.

The same actor then continues from the modified state, and a separate benchmark-owned receipt scores that exact output state and derives final success from the frozen threshold. Durable save and load reconstruct the completed phases without duplicating an action, while an ambiguous incomplete effect blocks instead of counting as success. The analysis requires the exact qualification–TRACE–packet–FIX–official-score identity chain and rejects drift in the public input, actor, model, adapter, tools, action catalogue, residual evidence, verifier, threshold, or budget. A launched blocked method attempt remains a zero outcome in its preregistered denominator and retains its incurred cost; only a result-blind task-invalid slot avoids launch.

C.3 Upstream boundaries

Who&When's released evaluator scores predicted agent and step; TraceElephant's released evaluator likewise scores component/step attribution with richer trace input. Neither defines the fault packet or recovery lifecycle. DoVer supplies the closest intervention precedent, but its retained recommender

can read reference ground truth and its intervener targets an AG2 session lifecycle. The packet, factorial design, and LongH adapters are therefore `locally_designed`.

C.4 Conditions for launch

The five TRACE methods must produce comparable live packets on the frozen roster before the localization comparison can start. Generic attribution is the fixed $T0$ comparator and dependency-only tracing is an explicit ablation, so the adapted Who&When and TraceElephant rows keep their adaptation qualifiers. Launch also requires one common audited live dispatcher for Same-Actor, generic, and residual FIX across all six tasks, and a live producer that projects the verified-root admission from the existing bridge. Every task slot must point at the current recovery contract rather than an older four-arm source matrix, and each task that runs $T^{\star}$ cells must have an independent verified root.

Two further conditions concern the retained evidence. A formal launch requires one shared raw lifecycle timeline and a provider-usage ledger, so that repair, postcondition, continuation, LocalResolve, and scoring appear on one record and the model-call counts are rederived from provider usage. It also requires a real process interruption that resumes through official scoring, completed method adapters and pre-target qualification, and a live successful-entry/no-fault producer for the negative control.

D Source-Backed Figure and Table Archive

This appendix restores the source-backed assets maintained in the parent `longh_graph` workspace for the build. It separates frozen benchmark facts from the corpus composition, and every value in it is read from the retained target snapshot. Figure 3 carries the corpus-scale comparison against prior work; the three tables below carry the benchmark-by-benchmark values, the environment quotas that assemble the corpus, and the protocol distinctions, at a readable final size. We do not place the old four-panel canvas directly in the appendix: shrinking it to the ICLR text block makes its labels unreadable, while cropping it severs row labels from the corresponding panels.

D.1 Frozen benchmark comparison

Table 5 retain the published scale and the recovery-protocol coverage of the compared benchmarks. The scale table keeps each source's native length unit and marks which benchmarks record units beyond failure-only, so the counts on its right-hand side are not summed. The protocol table marks which benchmarks replay the failed entry state, which carry TRACE labels, which evaluate a repair rather than a collected trace, and which enforce a frozen budget or retain an abstention outcome.

Table 5: **Frozen dataset comparison: recovery-protocol coverage.** Replay means that the failed entry state can be re-executed. TRACE→FIX requires a repair to be resumed or retried and evaluated; it is stronger than collecting a failure trace. A frozen budget must score exhaustion rather than merely report token use.

Benchmark	Replay entry	TRACE labels	TRACE→FIX	Frozen budget	Abstention
Who&When	n.r.	yes	n.r.	n.r.	n.r.
TraceElephant	yes	yes	n.r.	n.r.	n.r.
AgentRx	n.r.	yes	n.r.	n.r.	n.r.
TrajErrBench	n.r.	yes	oracle retry	n.r.	no*
LongRCA Bench	n.r.	yes	n.r.	n.r.	yes
SearchAuditBench	yes	yes	held-out repair	n.r.	no*
LongH-Debug v1.0	TRACE/FIX subset	provisional	yes	yes	retain unknown

"no*" denotes explicit exclusion rather than evidence of abstention: TrajErrBench excludes ambiguous cases by agreement, and SearchAuditBench excludes ambiguous or incomplete queries. "Oracle retry" injects detector feedback and retries the task rather than resuming from the frozen failure state. "Held-out repair" resumes from a predicted critical step and checks the benchmark's native answer signal. LongH-Debug replay applies only to the TRACE/FIX subset; TRACE labels are provisional, and unidentifiable failures are retained.

E TRACE–FIX EVALUATION CONTRACT

Every task begins as pending and the manifest stays non-launchable while the lifecycle conditions below remain open.

E.1 TRACE EVALUATION

All methods read the same public trajectory, task contract, model configuration, and TRACE resource ledger. The declared methods are Generic attribution, Who&When step-by-step with adapted input, TraceElephant all-at-once with an adapted public trace, a dependency-only ablation, and OUR TRACE. The adapted labels are explicit because these are not source-faithful native reproductions.

The independent evaluator reports Hit@1, Hit@3, MRR, candidate recall, VERIFIEDROOT@B, model calls, tokens, and elapsed time. The public fault packet contains no gold field and may cite only event, evidence, and check identifiers frozen during result-blind qualification. This is identifier- and field-level isolation; process-level separation is an additional launch condition, because unrestricted generated text could encode private information. VERIFIEDROOT@B is derived only from a benchmark-owned audit that binds a candidate hypothesis to a qualified check, execution, predicate, subject versions, result, source runtime receipt, independent auditor, and charged cost. The analyzer reopens the canonical configuration, event stream, observer evidence, and runtime-execution receipt named by that audit and reconstructs the verdict rather than trusting copied fields. Evaluator records, raw runtime sources, and root-verification evidence cannot be reused across runs. Our Trace is paired separately with every declared baseline on the canonical case bound to each task slot; the analyzer reports whether their paired populations are identical. Every launched blocked run remains in the denominator; only a task ruled invalid before method launch is absent from method comparisons.

E.2 FIX EVALUATION

Each fixer receives the same public failure state, frozen fault packet, legal tools, actor identity, and recovery budget. The FIX-only comparison uses the same independently verified-root packet for all three methods: Same-Actor Continue, a Generic/Fresh Fixer, and OUR FIX. A fixer may inspect public evidence and execute only the qualification-time check and repair catalogue, but may not access root rank, hidden annotations, evaluator verdicts, or a method-specific extra trajectory. A stable check definition may be run before and after repair through distinct ordered execution receipts. Every FIX-controller model call is charged separately from TRACE and actor continuation. Only tried repairs incur action cost. A delivered repair must have exactly one later postcondition check named in the frozen catalogue: a supported result permits continuation, while a refuted or unknown result blocks at the postcondition stage. Same-Actor Continue cannot execute FIX checks or repairs.

The three outcomes can disagree in general:

$$\begin{aligned} \text{VERIFIEDROOT} &\not\Rightarrow \text{LOCALRESOLVE}, \\ \text{LOCALRESOLVE} &\not\Rightarrow \text{TASKRECOVERY}, \\ \text{TASKRECOVERY} &\not\Rightarrow \text{a unique historical root}. \end{aligned}$$

A repair may remove one blocker while another remains, and an actor may also succeed through a compensating path without identifying the earliest cause.

The primary endpoint is paired TASKRECOVERY under verified-root input. Secondary endpoints are LOCALRESOLVE, harmful or false repair, repair delivery rate, and cost per successful recovery. Root-acquisition cost is excluded from the $T^\star$ verified-fault input condition because the fault information there is the controlled input; every later check, repair, and continuation cost is included. Harmful-repair and total-cost differences are paired against both Generic/Fresh Fixer and Same-Actor Continue.

E.3 CLOSED-LOOP MATRIX

Every qualified natural task runs $T0F0, T0F1, T1F0, T1F1$. If an independent verified root exists, it also runs $T^\star F_{\text{cont}}, T^\star F0, T^\star F1$. Paired FIX cells under one TRACE source must reference the same immutable packet; all three verified-root controls must reference the same independently

Table 6: **Configuration roster for the recovery design.** S is residual state, D is lookahead, and J is live outcome-model adaptation. Each row names the population it is scored on, out of the 200 paired natural cases used for localization, the 120 verified-fault cases used for recovery, and the 60 closed-loop tasks.

Config.	S	D	J	Experience	Population
Residual-Greedy	yes	1-step	no	none	60 closed-loop tasks
TraceDP	yes	multi-step	no	none	60 closed-loop tasks
Full	yes	multi-step	yes	live correction	120 verified-fault cases
Full-S	flat	multi-step	yes	live correction	120 verified-fault cases
Full-D	yes	1-step	yes	live correction	120 verified-fault cases
Memory-as-Prompt	yes	multi-step	non-JIT prompt	prompt	120 verified-fault cases
Fresh Debugger	–	–	–	–	120 verified-fault cases
bn-fit-modify (success-state control)	–	–	–	–	60 closed-loop tasks

admitted packet. Every cell preserves the same failed entry state, actor, public tools, total budget, stopping rule, and official verifier.

For an outcome Y, the paired estimands are

$$\begin{aligned}\Delta_{\text{Trace}} &= Y(T1F0) - Y(T0F0),\\ \Delta_{\text{Fix,generic}} &= Y(T^\star F1) - Y(T^\star F0),\\ \Delta_{\text{Fix,continue}} &= Y(T^\star F1) - Y(T^\star F_{\text{cont}}),\\ \Delta_{\text{Full}} &= Y(T1F1) - Y(T0F0),\end{aligned}$$

and the interaction is $[Y(T1F1) - Y(T1F0)] - [Y(T0F1) - Y(T0F0)]$. A positive full effect without positive TRACE or FIX evidence does not retroactively prove both components, and a positive interaction is necessary for a synergy claim. Mechanism execution without outcome or cost gain supports only an integration statement.

E.4 Design arms

Table 6 states the configuration of each arm, the residual-state, lookahead, and adaptation settings it carries, and the population it is scored on. Two arms scored on the same population differ only in the configuration named in the table.

E.5 Breadth-first roster and repetition

A result-blind qualification receipt is written before method outputs are opened, and it checks four conditions independently of method output: whether a public trace exists, whether the failed entry state can be re-executed, whether an official verifier can score the continued run, and whether the common cost ledger can be applied. Each roster slot fixes one task and one canonical case; an alias cannot count as another paired unit. If a natural candidate is invalid, it remains invalid and is not replaced. Only comparisons with an observed difference in task recovery, local resolution, action selection, or material cost advance to paired $R = 3$ runs. The exact trigger is frozen in the atomically create-only manifest. No $R = 1$ or $R = 3$ development result is by itself confirmatory.

E.6 Complete cell receipt

Each run records the TRACE packet and its public evidence references; every stable check definition, ordered execution, and observed result; every repair offered, tried, admitted, delivered, postcondition-rechecked, and judged harmful; and the residual obligations, stale evidence, and blocked tasks before and after execution. It also records the same-actor continuation together with its declared resource use and output-state identity, LocalResolve plus a separate benchmark-owned predicate receipt whose trigger identifier matches the recorded continuation identifier, and the stage at which a failure occurred, whether in TRACE, checking, repair, delivery, postcondition verification, continuation, or official

Table 7: **Closed-loop stages.** The stages run in the order shown, and each one records its own fields. Counts are recorded per stage and are not additive: a count belongs to the stage where the action was registered, and a later stage only sees the cells that reached it.

Stage	Recorded at this stage
1. Decision	planner decision identifier and the chosen action; declared cost of the chosen action; residual state version at the decision; remaining budget; outcome estimates the planner used
2. Check	stable check definition identifier; unique ordered execution identifier; observed result: supported, refuted, or unknown; subject resource versions bound to the result
3. Refutation	refuted hypothesis and check identifiers; public evidence that refutes the check; evidence marked stale by the refutation
4. Conditional repair candidate	candidate identifier and its precondition predicate; admitting check and its result; declared repair cost from the qualification catalogue
5. Attempt / admission	whether the repair was tried; admission evidence and its verdict; ordered execution identifier; an untried repair costs nothing
6. Delivery	delivered repair identifier; delivered state version; the single postcondition check named in the frozen catalogue
7. Postcondition recheck	postcondition check result; supported result permits continuation; refuted or unknown result blocks at this stage
8. Same-actor continuation	continuation identifier and actor identity; declared continuation calls and steps; output state identity
9. LocalResolve	predicate audit identifier; source trigger identifier that must match the continuation identifier; post-continuation predicate result
10. Official TaskRecovery	official score and frozen threshold; verifier identity and scored-state identity; derived TaskRecovery

scoring. The benchmark receipt records the official score, frozen threshold, verifier identity, scored-state identity, and derived TASKRECOVERY separately from method-authored lifecycle data. The analyzer reopens the canonical raw source for every FIX and LocalResolve check. The LocalResolve source trigger identifier must match the cell receipt's continuation identifier, and the score receipt's state identifier must match its declared continuation output. These are cross-receipt identity checks; a formal launch additionally requires one shared raw timeline that carries repair, postcondition, actor continuation, LocalResolve, and official scoring in order, plus provider-owned model-usage receipts. Transition evidence cannot be relabelled from the initial state or reused across cells, and a blocked receipt cannot contain events after its declared failure stage. These receipts are required because a terminal score alone cannot show that TRACE reached FIX or explain where a closed loop failed.

Table 7 states the stages a completed cell must traverse in order and the fields each stage records. A stage only sees the cells that reached it, so the counts are recorded per stage rather than added across stages.

E.7 PROMOTION GATES

TRACE is promoted only if it beats every preregistered strong TRACE baseline on paired unseen cases. FIX is promoted only if, under verified-root input, it improves task recovery or preserves recovery at materially lower cost without increasing harmful repair. The full loop is promoted only if $T1F1$ completes the real lifecycle and improves over $T0F0$. Formal paper claims additionally require at least eight task-disjoint natural cases and two independent reviews, the first covering first-party sources, local deviations, formulas, and runtime-method consistency, and the second covering configuration, execution, exceptions, persistence, interruption recovery, official scoring, provenance, and independent-oracle consistency. The case count is an internal planning rule rather than a statistical criterion: whether a comparison supports a claim still depends on its pairing, effect size, heterogeneity, and uncertainty, not on reaching a task count.

E.8 SECONDARY MECHANISM ABLATIONS

Only after TRACE, FIX, and their handoff are established do we test whether residual-state structure, multi-step lookahead, or JIT adaptation provides additional gain. Table 8 retains those preregistered isolations. The D comparison must include an information-gain policy with explicit repair-budget reservation; a weak one-step strawman is insufficient.

Table 8: **Secondary mechanism bets after TRACE/FIX validation.** Each row changes one source of information or decision-making while holding the actor, public input, legal actions, base estimates, and budget fixed.

Bet	Isolation	Mechanism evidence	Decisive outcome
S	Full vs. no structured state	State collisions, stale evidence, irrelevant candidates ↓	Paired recovery ↑ without extra evidence or actions
D	Multi-step / information gain + repair reserve / active debugger	Chosen checks open affordable repair paths	Better recovery–cost frontier under the same estimates
J	Full vs. no J vs. memory prompt	Held-out calibration and ranking ↑; coverage and abstention reported	Recovery or cost gain beyond the shared base model

F CASE STUDIES: WHAT THE ACTOR DID, WHAT FAILED, AND WHAT EACH METHOD ANSWERS

This appendix follows individual runs instead of populations. Every event identifier, ranked candidate, check verdict, and reward below is copied from a retained record. Section F.1 takes one failure — `db-wal-recovery` — and reads it in four layers: what the actor saw and did, how that action reached the final failure step, what our runtime supervisor recorded while it tried to repair the task, and what each localization method answers about the same trajectory. Section F.2 repeats the localization comparison on four further cases and Section F.4 repeats the recovery comparison on two more. The localization labels are provisional model adjudications that carry no causal authority, so this appendix illustrates mechanism and failure mode rather than confirming either claim, and none of its counts are summed with any count elsewhere.

Section F.5 then opens the runtime record itself — one supervisor decision, quoted field by field. Section F.6 runs the other direction and shows what the generic and fixed-order baselines actually did on the same task before they scored zero.

F.1 ONE FAILURE IN FOUR LAYERS: `DB-WAL-RECOVERY`

The task asks the agent to repair a corrupted write-ahead log in `/app`, recover all eleven records, and write them to `/app/recovered.json`. The base database shows five records; the other six live only in `/app/main.db-wal`. Three retained records are used below, and they are different executions of this one task specification: the actor trajectory and the failure outcome come from the observation run `terminal_bench_hy3_db_wal_recovery_observed_runtime_v1_attempt1`, the supervisor records come from development runs of our method on the same task, and the ranked answers come from the method-unseen TRACE campaign, whose run uses a different agent.

Layer 1 — what the actor saw, and the action that caused the damage. The actor's first command listed `/app` and saw both files: `main.db` at 8192 bytes and `main.db-wal` at 16512 bytes. Its third command opened the database through the `sqlite3` command-line tool and asked for the journal mode and a checkpoint. That read destroyed the log. Table 9 lists the command indices at which the public file state changed: the log is present after command 0 and absent after command 2.

The decisive step is worth quoting literally, because what the actor saw is the reason nothing in the transcript looks wrong. Command 2 printed the two pragma results and returned success:

```
[cmd 2] $ sqlite3 main.db "PRAGMA journal_mode;
                           PRAGMA wal_checkpoint;"
        === PRAGMA ===
        wal
        0|0|0
```

The reply reports WAL mode and a checkpoint result of `0|0|0`, which is the healthy answer, while the checkpoint itself removes `main.db-wal` from the directory. Nine commands later the actor tries to back the log up and the copy fails without an error message, because by then there is nothing left to copy:

```
[cmd 9] $ cp main.db-wal /tmp/main.db-wal.bak 2>/dev/null
        $ ls -la /app
        main.db   8192    (the log is no longer listed; exit status 0)
```

Index	Command (abridged) → observed effect on `main.db-wal`
0	`ls -la /app/` → the log is present at 16512 bytes
2	`sqlite3 main.db ...; PRAGMA wal_checkpoint;` → **the log is gone**
9	`cp main.db-wal /tmp/...bak 2>/dev/null` → the backup fails silently, because the log is already gone
37	`python3 reconstruct.py` → the actor rebuilds a log from its own script
38	`sqlite3 main.db ...; PRAGMA wal_checkpoint(TRUNCATE);` → **the rebuilt log is gone again**
95	the actor states the mechanism in an `echo` → "WAL deleted by sqlite on close, as expected for a valid-enough WAL"

Table 9: **Layer 1: the actor's trajectory on `db-wal-recovery`.** Command indices are positions in the retained public state log of the observation run; each row is the observed effect of that command on `/app/main.db-wal`. Every command in the table reported success, and no step in the transcript raises an error.

Layer 2 — how that action reached the final failure step. Nothing reports a problem at command 2. The command is a read, it returns the base schema, and its exit status is success. The damage is a deleted file, not a wrong value, so the only observable change is the disappearance of `main.db-wal` from the file listing. The next commands read the base records and then hunt the filesystem for the log the actor itself had just removed (commands 4–7), and command 9 tries to back it up after the fact. The actor then does recover the mechanism: it writes a reconstruction script, rebuilds a log at command 37, and states the cause in its own words at command 95. What it never recovers is the content, because the six records existed only inside the original log and every later `sqlite3` read consumes the rebuilt one as well (command 38 is the same mistake again). The run ends on the agent timeout, and the official verifier reports two failures out of seven tests — `test_recovered_data_completeness` and `test_wal_was_decrypted`. Both fail on the same assertion, `id_to_item[1]["value"] == 150`, reported as "WAL update for id=1 not applied" with `assert 100 == 150`: the submitted record still carries the pre-update value that was visible in the base file, for a final reward of 0.0. The gap between the cause at command 2 and the symptom at the end of the run is the whole difficulty: the decisive act is early, quiet, and irreversible, while the failure signal is late, loud, and merely downstream.

Layer 3 — what our runtime supervisor recorded, and what it changes. On the same task our supervisor keeps a decision record instead of a narrative. Each row of Table 10 is one retained run; the supervisor proposes a slate of legal actions, charges each one against a single budget, executes the selected action, and writes the verdict back into the residual state.

Run (arm)	B	Dec.	Checks	Check verdicts	Rep.	Reward
`db-wal` (Residual-Greedy)	8	3	3	checks 0–2, all *unknown*	0	0.0
`openssl` (FULL)	12	8	7	check 5 *refuted*, then *supported*; 5 more supported	0	1.0
`openssl` (TRACEDP)	12	18	8	check 5 *refuted* three times; 5 supported	2	1.0
`cancel` (FULL)	12	45	8	checks 2 and 6 *refuted*; 6 supported	2	0.0

Table 10: **Layer 3: what the supervisor recorded on the same tasks.** B is the intervention budget in units; "Dec." counts supervisor decisions; "Checks" and "Rep." count executed actions, not offered candidates. In the `db-wal` arm every executed check returned *unknown* because its harness exited with code 70 (a missing prerequisite), so the sweep paid three units and learned nothing; the budget fell from 8 to 5 and no repair was ever affordable.

Three properties of this record matter for the rest of the paper. First, the supervisor stores the reason an action was legal, its declared cost, and the verdict it returned, so a failure can be attributed to a specific stage — here, to a check whose harness could not run — instead of to a final score. Second, the `openssl` FULL row shows the closed loop working on a case that does not need the log story: the script check is refuted, the supervisor hands control back to the same actor, the actor rewrites the script, the recompiled state re-runs the identical predicate, and it is then supported. Third, the TRACEDProw ends with check 5 still refuted while the official verifier returns 1.0: a local predicate and the benchmark's own verdict are different measurements, and a report that showed only one of them would be wrong.

Layer 4 — what each localization method answers on the same trajectory. The adjudicated cause of the campaign run is `step-0004`, the first successful Python `sqlite3.connect` on the database; the earlier shell step does not reach its Python payload because a missing `file` binary aborts the script first. Table 11 shows what each method ranks on that public trajectory.

Method	Rank 1	Rank 2	Rank 3	Gold (`step-0004`)
Adjudicated cause	`step-0004`	—	—	—
Our TRACE	`step-0004`	`step-0001`	`step-0002`	rank 1
Dependency-only	`step-0003`	`step-0002`	`step-0001`	absent
Generic causal	`step-0003`	`step-0002`	`step-0005`	absent
TraceElephant (all-at-once)	`step-0003`	—	—	absent
TraceElephant (binary search)	`step-0003`	—	—	absent
Who&When (all-at-once)	`step-0004`	—	—	rank 1
Who&When (binary search)	`step-0004`	—	—	rank 1

Table 11: **Layer 4: ranked step candidates on the same public trajectory.** Our TRACE and both Who&When variants return the adjudicated node first. The dependency-only, generic, and TraceElephant variants all return `step-0003` instead: they stop at the shell step where the script aborted, which is the earliest step that looks wrong and not the step that destroyed the log. This case therefore separates our TRACE from three of the four baseline families but not from Who&When, and we report it that way.

Read together, the four layers are the argument of the paper in miniature. The actor's own record contains the decisive action, but it is a successful read whose only trace is a file that stopped existing. A tracer that ranks by earliest failure signal stops one step early; a tracer that ranks by visible anomaly stops at the end; a tracer that follows the read-after-write chain from the blocked obligation reaches the destructive open. Once the fault is localized, the recovery side still has to decide what to spend on, and Layer 3 shows a real arm spending its whole discretionary budget on checks that returned *unknown*. Localization and recovery are separate decisions, and this case is where both of them visibly matter.

F.2 LOCALIZATION CASES: WHERE THE DEPENDENCY STRUCTURE CARRIES THE SIGNAL

Case L1 — `large-scale-text-editing`: the design commitment, not the first file. The task requires transforming a CSV through vim macros invoked as `:%normal! @a`. The adjudicated cause is `step-0004`, where the actor committed to storing Ex-style command strings in registers `a`, `b`, and `c`, while the mandated invocation replays a register as Normal-mode keystrokes. Public evidence from `step-0008`–`step-0011` shows the resulting mismatch. Our tracer returns `step-0004` first. The generic one-shot baseline returns `step-0003` and describes it as the moment the macros were formulated, one step before the adjudicated commitment; TraceElephant's all-at-once pass returns `step-0019`, the loudest later crash; Who&When returns `step-0005`. The families fail in opposite directions — one stops at the earliest write, the other stops at the last thing that broke — and neither asks which earlier decision made the later execution wrong.

Method	Rank 1	Rank 2	Rank 3	Gold
Adjudicated cause	`step-0004`	—	—	—
Our TRACE	`step-0004`	`step-0008`	`step-0011`	rank 1
Generic one-shot	`step-0003`	`step-0004`	`step-0005`	rank 2
Dependency-only	`step-0003`	`step-0002`	`step-0001`	absent
TraceElephant (all-at-once)	`step-0019`	—	—	absent
Who&When (all-at-once)	`step-0005`	—	—	absent

Table 12: **Case L1, `large-scale-text-editing`.** Ranked step candidates returned by each method on the same public trajectory. The adjudicated cause is `step-0004`; the baselines return a neighbouring write and a downstream crash instead.

Case L2 — `path-tracing-reverse`: the first authored source, not the noise around it. The adjudicated cause is `step-0011`, the first authoring of `/app/mystery.c`, which embedded several structural defects at once; the reference image therefore never matched. Our tracer and the dependency-only ablation both return `step-0011` first. The generic one-shot baseline returns `step-0002`, an incidental stderr-redirection side effect that left a progress dump in the working directory; TraceElephant returns `step-0017`, the pixel mismatch where the error finally became visible; Who&When returns `step-0009`. The two successful readings differ in scope rather than in rank: the dependency-only ablation recovers the right node when the damage sits at a single write, and Case L1 shows it losing that node when the damage sits in a design decision instead.

Method	Rank 1	Rank 2	Rank 3	Gold
Adjudicated cause	`step-0011`	—	—	—
Our TRACE	`step-0011`	`step-0012`	`step-0013`	rank 1
Generic one-shot	`step-0002`	`step-0011`	`step-0001`	rank 2
Dependency-only	`step-0011`	`step-0030`	`step-0017`	rank 1
TraceElephant (all-at-once)	`step-0017`	—	—	absent
Who&When (all-at-once)	`step-0009`	—	—	absent

Table 13: **Case L2, `path-tracing-reverse`.** Same comparison as Table 12. Here the dependency-only ablation also recovers the gold node, so this case shows that task structure carries signal without showing that the fuller state is required.

F.3 TWO CASES WHERE OUR TRACE DOES NOT RECOVER THE GOLD NODE

Reporting only successes would misrepresent the method, and the two failures below are the same error the previous subsection diagnoses in the baselines: stopping at the event where the damage became visible instead of the event that produced it.

Case L3 — `path-tracing`: we blame the freeze, not the write that caused it. The task became unrecoverable once every tool call returned `OSError: [Errno 27] File too large` and the actor timed out. The adjudicated cause is `step-0016`, where the actor wrote the full $2400 \times 1800 \times 3$ image array to `/tmp/arr.bin`. Our tracer returns `step-0035` — the freeze

itself — followed by `step-0021` and `step-0008`, so the gold node is absent from our top three. TraceElephant's all-at-once pass returns `step-0016` first and is correct; the generic one-shot baseline places it third. The failure is a resource-exhaustion case with no error surface at the causal write, and our tracer followed the crash.

Method	Rank 1	Rank 2	Rank 3	Gold
Adjudicated cause	`step-0016`	—	—	—
Our TRACE	`step-0035`	`step-0021`	`step-0008`	absent
Generic one-shot	`step-0008`	`step-0009`	`step-0016`	rank 3
Dependency-only	`step-0004`	`step-0002`	`step-0001`	absent
TraceElephant (all-at-once)	`step-0016`	—	—	rank 1
Who&When (all-at-once)	`step-0008`	—	—	absent

Table 14: **Case L3, `path-tracing`.** Our tracer ranks the terminal freeze first and misses the causal write entirely; the all-at-once baseline recovers it.

Case L4 — `openssl-selfsigned-cert`: a rubric failure leaves no error to follow. Every OpenSSL and file step in this run succeeded, and the final verifier run also succeeded, yet the benchmark returned reward 0.0 on the official rubric. The adjudicated cause is `step-0008`, the verification script that depends on a package the task does not allow. Our tracer returns `step-0003` — the earliest certificate-construction step — first and places the gold node second, so it earns Hit@3 but not Hit@1. TraceElephant's binary-search variant returns `step-0008` first and is correct; Who&When returns `step-0013`. This case is the cleanest illustration of why a trace view is necessary but not sufficient: with no error in the transcript, the only signal that distinguishes the failing deliverable from the healthy ones is the rubric the task contract encodes.

Method	Rank 1	Rank 2	Rank 3	Gold
Adjudicated cause	`step-0008`	—	—	—
Our TRACE	`step-0003`	`step-0008`	`step-0011`	rank 2
Generic one-shot	`step-0005`	`step-0003`	`step-0008`	rank 3
Dependency-only	`step-0003`	`step-0004`	`step-0009`	absent
TraceElephant (binary search)	`step-0008`	—	—	rank 1
Who&When (all-at-once)	`step-0013`	—	—	absent

Table 15: **Case L4, `openssl-selfsigned-cert`.** No step in the transcript fails, so the failure is only visible against the task contract. Our tracer ranks the gold node second; the binary-search baseline ranks it first.

F.4 Recovery lifecycles: how FIX runs, and how it fails to

The four runs below are retained single executions. The two OpenSSL rows are the closed-loop runs of Section 4 at budget $B = 12$; the two `db-wal-recovery` rows are development runs at budget $B = 8$. Each began from a public failed state and ran registered checks through the supervisor. They establish that the lifecycle runs; they do not establish that one arm is better than another.

Case F1 — OpenSSL, `Full`: refute, hand back, re-check, accept. The first selected action was registered check 5, the predicate over `/app/check_cert.py`, and it came back *refuted*. The supervisor's next decision was `continue_actor` rather than a repair. Checks 4 and 1 then ran and were supported, after which the actor's rewrite was confirmed: the same predicate was re-run as check 5:2 and came back *supported*. Checks 3, 0, and 2 followed, all supported, and the official verifier returned reward 1.0. The run used 8 decisions and 7 checks with no repair action at all. This is the loop the method claims: a refuted obligation hands control back to the same actor, the actor changes the state, the recompiled state re-runs the predicate, and the benchmark — not the supervisor — decides the outcome.

Case F2 — OpenSSL, `TraceDP`: the same outcome for three times the budget. The TraceDP arm reached the same official reward 1.0, but it ran 8 checks, three of which were refuted, and

Record field	Value as written
run, decision	`db_wal_recovery_residual_greedy_dev_v1`, `runtime-joint-decision-000000`
selector rule	`runtime_owned_two_step_option_bellman_v2`, horizon 2
declared status	`implementation_unverified`
budget before	8 units
legal actions (slate)	`continue_actor` at cost 0, and five registered checks at cost 1 each; `nonempty_action_count` $= 5$
completion proxy	0.05 for every candidate, `continue_actor` included
expected unblocked fraction	0.6833 (`continue_actor`); 0.7150, 0.7118, 0.7055, 0.6992, 0.7023 (checks 0–4)
selected action	`run-registered-check:0:1`, the candidate with the highest unblocked fraction
written back	verdict `unknown`, harness `return_code` 70; budget $8 \rightarrow 7$
option chain	opened for check 2: `conditional-targeted-repair:repair-json-fidelity:check-2`

Table 16: **One supervisor decision, quoted in full.** Every value in the right column is copied from `runtime-decisions.jsonl` of the `db-wal` Residual-Greedy run. The record states which actions were legal, what each was expected to buy, which one was chosen, what the execution returned, and which conditional follow-up had been registered before the outcome was known.

additionally executed two targeted repairs at cost 2 each, exhausting the budget to 0 across 18 decisions. The refuted checks are the first and the two re-runs of predicate 5 (`5:1`, `5:2`, `5:3`); each re-run triggered a targeted repair, after which the predicate was refuted again. Against the 0.03 charged by `Full`, this arm spent 0.09 for the identical accepted result. Two arms reaching the same score is exactly the pattern that Section 4 records, and the cost column is where the difference shows up. This arm also ends with predicate 5 still refuted while the benchmark returns 1.0, which is why the paper keeps LOCALRESOLVE and TASKRECOVERY as separate columns.

Case F3 — `db-wal-recovery`, `Residual-Greedy`: a fixed sweep spends the budget on undecidable checks. This arm ran registered checks 0, 1, and 2 in catalogue order. All three returned the verdict *unknown* rather than supported or refuted, because each check harness exited with code 70 (a missing prerequisite), and the run ended at reward 0.0 with the budget drawn down from 8 to 5 and no repair attempted. Because the sweep is order-fixed, the checks that could have discriminated among the remaining hypotheses were never reached, and the repair that would have resolved the task was never affordable. The observed behaviour matches the budget counterexample of Section 2: an order-fixed policy pays for checks whose outcomes cannot change what it does next.

Case F4 — `db-wal-recovery`, `TraceDP`: the supervisor never fires. This is the opposite failure from Case F3: the policy never entered the recovery loop, so no amount of budget accounting could help. It is recorded here because a method that silently declines to act is indistinguishable, in an aggregate success rate, from a method that acts and fails.

F.5 WHAT THE RUNTIME RECORD CONTAINS, AND WHAT IT CHANGES

The supervisor does not write a narrative; it writes one JSON record per decision. That record is the only place where the reasons for an intervention survive, so it is worth quoting one in full. Table 16 transcribes the first decision of the `db-wal` Residual-Greedy run, field by field, without reformulation.

The same record shape appears in the successful OpenSSL run, and there the two-step lookahead is visible as a decision that was prepared and then never charged. At budget 12 the slate held six candidates and the selector opened a conditional repair, `conditional-targeted-repair:repair-stdlib-verification-script:check-5`, tied to predicate 5. Check 5:1 then came back *refuted*. The next decision was `continue_actor` rather than the repair; the actor rewrote the script; the identical predicate was re-run as check 5:2 and came back *supported*, at which point the conditional repair became moot. The run ended with zero repairs charged, a total cost of

Arm	What it did on the same task	Charged	Reward
Fresh Debugger (`fresh_debugger_v1`)	63 shell calls with no supervisor: decoded the log, recovered the true eleven rows, then replaced the two updated values with fabricated ones	0	0.0
Fixed-order joint (`fixed_runtime_option_chain_dev_v1`)	19 consecutive `continue_actor` decisions, then one check; supervision then disabled itself	4	0.0
Residual-Greedy (our controller, fixed sweep)	checks 0–2 in catalogue order, all `unknown`; stopped with budget left	3 of 8	0.0
TRACEDP (our controller)	51 decisions, every one `continue_actor`; no check and no repair selected	0	n.r.

Table 17: **Four controllers on one task.** "Charged" is the intervention budget actually spent; "n.r." means the run carries no verifier result. The Fresh Debugger arm runs without a supervisor and therefore charges nothing, which is why its entry is zero rather than small.

0.03, and official reward 1.0. The option chain is what makes this legible after the fact: the record shows what the controller was prepared to buy, not only what it bought.

The same record also carries the state the recovery decision reads. In the OpenSSL run's final state each entry of `acceptance_predicates` holds the predicate's proposition, its `state` (`current_supported` or `current_refuted`), and pointers to the evidence and incident that produced that state (`latest_evidence_id`, `latest_incident_id`). Alongside it, `resource_versions` shows `/app/check_cert.py` at version 2 while `/app/ssl/server.crt`, `/app/ssl/server.key`, `/app/ssl/server.pem` and `/usr/bin/openssl` all stay at version 1: the actor's rewrite bumped exactly one resource, and that version bump is the recompilation the method claims. Every task status in the same state is `awaiting_evaluation`, which is the record's way of saying that the supervisor never scores itself.

Read against the tracer, this record supplies three things a transcript does not. A *blocked obligation* with its incident pointer is a start point for the backward search, which is why the search in Section F.1 begins from a predicate rather than from the last error. *Resource versions* are what make the read-after-write chain checkable at all: a predicate whose supporting evidence predates a write to one of its resources is stale, and that is a mechanical test rather than a judgement about how suspicious a step looks. And the *verdict written back* closes the loop, because the next decision reads the updated predicate state rather than the previous decision's intent. Read against the recovery controller, the same record supplies the legal action set, the declared cost of each action, the value column that orders it, and the conditional follow-up registered before the outcome was known.

Three properties of this record limit what it can support, in the OpenSSL decision the value column barely separates the candidates: the expected unblocked fraction runs from 0.9367 (`continue_actor`) to 0.9425 (check 5:1), a spread of less than one point, while that same candidate also carries the longest expected duration. A ranking that rests on differences this small is not evidence of good calibration. Third, in the `db-wal` decision above, the highest-value candidate was a check whose harness exited 70 and returned `unknown`, so the ordering the record produced was not actionable at all. These three properties are why Section 2 keeps the value function's inputs separate from the official score, and why this appendix uses the record to explain mechanism rather than to argue for superiority.

F.6 WHAT THE BASELINE FIXERS DO ON THE SAME TASK

The localization comparison above has a recovery counterpart. Table 17 lists four retained single runs of the same task, `db-wal-recovery`, under four different controllers, including two baseline arms. All four are separate executions; none is a re-scoring of another.

The Fresh Debugger arm is the most informative of the four, because it diagnoses the task correctly and then discards the diagnosis. It decodes the log by hand: at call 29 it recovers the single-byte XOR key `0x42` from the retained state, at call 31 it puts the decoded log back and reads the database, and at call 32 it reads all eleven rows, including the two the log had updated. It then works out the

mechanism that destroyed the original log in the first place — a read-write `sqlite3` connection checkpoints the log on close and removes it — which is the same mechanism the failing actor in Section F.1 recovered only in retrospect. At call 55 the run states the conflict in its own words: the four visible checks pass, but the values it is about to submit are not the values it recovered. It then chooses the visible checks over the data: it resets the database to the five original rows, inserts six rows that were never in the log (`fig 600` through `mango 1100`), and leaves the two updated rows at their pre-update values. The submitted `recovered.json` is therefore eleven rows with `apple 100` and `banana 200`, where the log actually carried `apple 150` and `banana 250`. The official verifier returns 0.0, and the two tests that fail are `test_recovered_data_completeness` and `test_wal_was_decrypted`. The cost of reaching that answer was 5,898,531 input tokens in this single run, roughly thirty times the 191,535 tokens spent by the fixed-order arm that gave up earlier — and both scored zero.

The other three arms fail at three different stages, and none of them fails at localization. The fixed-order joint arm never finds an action it can contract: it repeats `continue_actor` nineteen times, selects one check, and the supervisor disables itself with `RuntimeError: runtime option replan has no executable contract-bound repair`. The Residual-Greedy arm pays for three checks whose harness exits 70, so it spends budget without learning anything and never reaches a repair. Taken together with the Fresh Debugger arm, the four rows show why this paper keeps the stage at which the loop failed as a first-class record. A single success rate would report four zeros and hide the fact that one arm diagnosed the task correctly and then submitted data it had already identified as false, one never found a legal action, one paid for undecidable checks, and one never acted.

F.7 What the baselines do, and why it fails

Reading the localization cases together gives a single explanation for the population result reported for the method-unseen split. The generic one-shot baseline returns the earliest *written* artifact or the earliest failed prerequisite: a draft script in Case L1, a side-effecting redirection in Case L2, a missing-package probe in Case L4. The retrieval-heavy families return the loudest visible anomaly: the `File too large` crash in Case L3, the pixel mismatch in Case L2. Neither reading is arbitrary — both are defensible answers to "what looks wrong in this transcript" — and that is precisely the gap. A long-horizon failure has a write that is quiet, early, and decisive, and a symptom that is loud, late, and merely downstream. Methods that rank by anomaly salience cannot separate the two, and methods that rank by recency or by position of first failure cannot either. The dependency-linked residual state is what allows our tracer to follow the read-after-write chain from the blocked obligation back to the decisive write, which is why it recovers the gold node in Cases L1 and L2 and in the four-layer case of Section F.1, and why it still fails in Cases L3 and L4, where the decisive event either leaves no trace at all or is only identifiable against the task contract. Those two failures bound the current method rather than illustrate it.

The recovery side has its own version of the same gap, shown in Section F.6: a generic fixer that recovered the correct answer and then submitted a state chosen to satisfy the visible checks, and two controller arms that failed before producing an actionable step at all.

G From Recovery Traces to Learning Signals

The current system performs no actor-parameter training. A future extension could learn an action-outcome model from observed transitions $e_j = (z_j, u_j, y_j, z_{j+1}, c_j, \Delta t_j, R_T, s_j)$, where s_j preserves whether an outcome was predicted or observed, isolated or live, and local or task-level. Missing outcomes must remain missing rather than being relabeled from predictions. Such a model would matter only after held-out calibration and action ranking improve; internalized capability would require a separate actor-only evaluation with the supervisor, memory, and teacher disabled.