TRACEACE: ANALYSIS, CLUSTERING, AND EXPLANATION OF FAILURES IN LLM-AGENT TRACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Debugging an LLM agent from its final answer is underdetermined: the answer does not reveal which span introduced an error, whether failures across traces share one fix, or whether a fluent root-cause narrative matches recorded execution. This paper presents **TRACEACE**, a collection-level workflow that turns heterogeneous observability traces into localized issues, interpretable same-fix clusters, evidence-constrained root-cause hypotheses, and an engineering report. Canonicalization preserves source span identifiers; taxonomy-conditioned scanners run alongside optional rule-based anchors and a structural channel; transparent signatures and a conservative merge form same-fix clusters, each investigated via one representative; and every hypothesis quotation is checked against the workspace before replay on secondary traces, escalating on disagreement. Across TRAIL and AgentErrorBench with eight LLM backbones and seven baselines, **TRACEACE** attains the highest joint accuracy on TRAIL GAIA under every backbone, with a 24.2-point mean gain over the strongest baseline, and on TRAIL SWE under seven of eight; on critical root-cause attribution it matches the strongest per-trace attributors. Ablations show the largest losses when specialized scanners are replaced by a unified one (up to 3.3 joint points) or cross-trace validation is removed (about 4 points of root accuracy). **TRACEACE** uses less than half the per-trace tokens of AgentDebug, and cluster-level investigation raises root accuracy over per-trace investigation on both AgentErrorBench environments (16.2 vs. 14.9 and 20.4 vs. 18.4). These are single-run point estimates; same-fix gold labels and interventional validation remain unavailable.

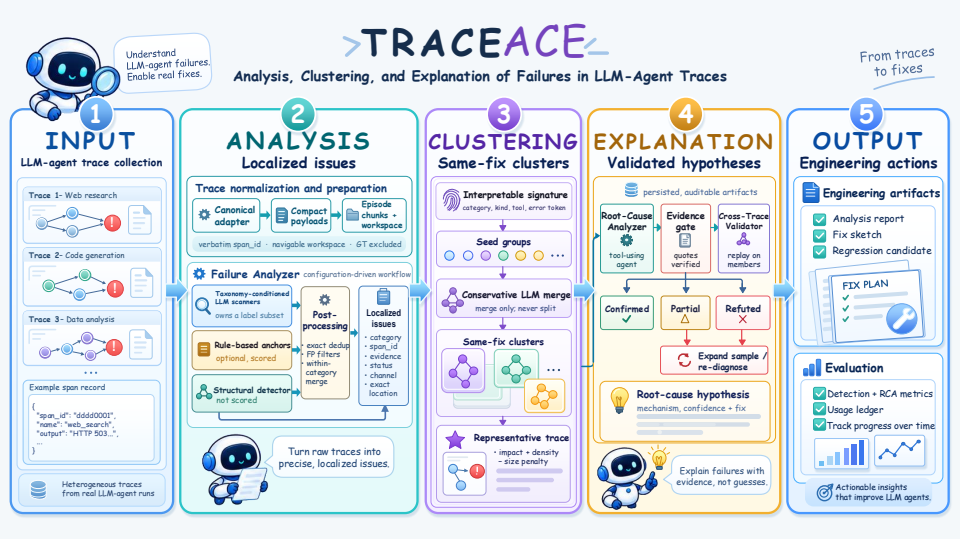


Figure 1: TRACEACE turns a collection of LLM-agent traces into localized issues, same-fix clusters, and validated root-cause hypotheses. The Failure Analyzer localizes failures per trace; clustering groups issues one fix would address; the Root-Cause Analyzer investigates one representative behind an evidence gate, and the Cross-Trace Validator replays its hypothesis on other members.

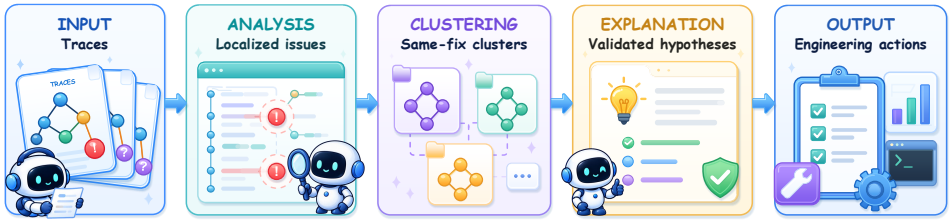
Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping 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.

1 INTRODUCTION

LLM agents (Yao et al., 2023; Schick et al., 2023; Sumers et al., 2023) interleave model calls, tool invocations, retrieval, memory, and orchestration over trajectories with hundreds of operations. Observability systems record an execution as a trace of nested spans, each carrying identity, parentage, timing, status, and attributes (OpenTelemetry Authors, 2026); AI-specific conventions add messages, model parameters, retrieved documents, and tool calls (OpenInference Authors, 2026). Visibility is not diagnosis: an engineer must still determine which event caused harm, where the failure first became actionable, and whether the same mechanism recurs elsewhere.

Existing benchmarks cover parts of this problem: TRAIL (Deshpande et al., 2025) provides a 21-leaf error taxonomy with exact-span annotations, TELBench (Wang et al., 2026) studies harmful-span localization in deep-research trajectories, and TraceElephant (Chen et al., 2026) measures the value of complete over partial observation. All evaluate one trajectory at a time, whereas operators must also answer a collection-level question: if a thousand traces fail, how many distinct fixes are needed? Per-trace labels do not reveal recurring mechanisms, identify representative cases, test whether an explanation transfers across traces, or yield a regression plan.

This paper calls the broader task *collection-level agent-trace debugging*. It takes a cohort of traces, possibly from different agent frameworks or telemetry schemas, and returns an auditable chain from traces to engineering actions:



Analysis must preserve exact source locations while finding small failures in long, repetitive traces; *clustering* must group incidents that admit the same intervention, even when their topics or error strings differ; *explanation* must cite recorded evidence and remain falsifiable when replayed against other cluster members.

TRACEACE realizes this chain with deterministic representations, specialized model calls, and persisted audit artifacts (Figure 1). It canonicalizes and segments heterogeneous traces, combines taxonomy-conditioned scanner proposals with deterministic detection channels, groups recurring failures through interpretable signatures, selects a representative for evidence-constrained investigation, and validates the resulting hypothesis on secondary traces before producing an engineering report.

This paper makes four contributions:

1. **Evidence-preserving analysis.** Adapters keep every source `span_id`; compaction, segmentation, and filesystem workspaces make long traces inspectable without exposing ground truth.
2. **Hybrid, auditable detection.** Taxonomy-conditioned scanners plus deterministic and structural channels; status-preserving filters and merge guards record how each issue arose.
3. **Interpretable collection-level diagnosis.** Transparent signatures seed same-fix groups, a conservative merge joins compatible ones, and an explicit score picks the representative trace.
4. **Falsifiable root-cause explanation.** Hypotheses cite resolvable spans and recoverable quotations; replay on secondary traces confirms, qualifies, or refutes them and can trigger re-diagnosis.

The evaluation compares **TRACEACE** with seven baselines across eight backbones, using TRAIL for multi-error detection and AgentErrorBench for critical root-cause attribution. **TRACEACE** leads joint accuracy on TRAIL GAIA under every backbone and on TRAIL SWE under seven of eight, matches the strongest attributors on AgentErrorBench, and improves root accuracy over per-trace investigation when clusters share one investigation (Section 4).

2 PROBLEM FORMULATION

Let $\mathcal{T} = \{T_i\}_{i=1}^{N}$ be a collection of agent traces and $\mathcal{Y}$ its issue vocabulary. Diagnosis proceeds in three stages: *analysis* maps traces to localized issues $\widehat{\mathcal{I}}$, *clustering* partitions them into same-fix groups $\widehat{\mathcal{C}}$, and *explanation* returns hypotheses with cross-trace validation records $\widehat{\mathcal{H}}$ (object definitions in Appendix A).

Span identity. A trace T_i is a timestamp-ordered span sequence and $\mathrm{ID}(T_i)$ the set of its verbatim source span identifiers. Every localized output must satisfy $\mathrm{loc}(I) \in \mathrm{ID}(T_i)$ under exact string equality, because telemetry providers use leading zeros, mixed case, and prefixes that re-numbering would silently invalidate; **TRACEACE** therefore carries the source identifier unchanged through adaptation, compaction, chunking, prediction, and scoring.

Analysis: taxonomy-constrained localized issues

An issue is a tuple of category, exact source span, short summary, supporting observation, and impact. The category must be a leaf of the workflow taxonomy $\mathcal{Y}_d$, and the observation must be recoverable from the trace at the cited span; ground truth is an input to evaluation, never to analysis. Several issues may legitimately share one span, since one event can violate more than one category.

Clustering: from recurring issues to same-fix groups

Topical similarity is not the target: two timeouts may need different fixes, while surface-different errors may vanish with one schema correction. Two issues belong together when a single intervention would address both. Clustering proposes such a partition and selects one representative trace per cluster by a transparent utility; because the relation is an operational target rather than a checkable equivalence, the partition's validity is tested by cross-trace validation, not assumed.

Explanation: grounded hypotheses and cross-trace validation

Investigating the representative $T_k^\star$ of cluster C_k yields a hypothesis H_k with a root cause, a mechanism, evidence pairs $E_k = \{(\ell, q)\}$ of span identifier and quotation, a confidence, and a fix sketch. Its grounding rate on a trace T is

$$g(H_k, T) = \frac{1}{|E_k|} \sum_{(\ell,q)\in E_k} \mathbf{1}[\ell \in \mathrm{ID}(T) \ \wedge \ q \sqsubseteq \mathrm{text}(T[\ell])]\,, \tag{1}$$

where $q \sqsubseteq \mathrm{text}(T[\ell])$ means the quotation is recoverable from the cited span; an admitted hypothesis requires $E_k \neq \varnothing$ and $g(H_k, T_k^\star) = 1$. Each sampled secondary member then receives a verdict in $\{\textsc{confirmed}, \textsc{partial}, \textsc{refuted}\}$, and partial or refuted verdicts are all kept, and a partial or refuted one triggers further sampling and mechanism-based re-diagnosis, so a proposed shared fix is a hypothesis another trace can contradict.

3 TRACEACE: ANALYSIS, CLUSTERING, AND EXPLANATION

TRACEACE organizes collection-level debugging into three stages, *Analysis*, *Clustering*, and *Explanation* (Figure 1), following the contract of Section 2. The **Failure Analyzer** detects, classifies, and localizes failures within individual traces; after clustering recurring issues, the **Root-Cause Analyzer** investigates representative traces to produce evidence-grounded hypotheses, which the **Cross-Trace Validator** tests on other cluster members. The Failure Analyzer is a configuration-driven workflow of LLM scanners, whereas the Root-Cause Analyzer is a tool-using agent. Algorithm 1 summarizes the pipeline; every intermediate object is persisted, so each explanation stays traceable to its supporting spans.

Algorithm 1: TRACEACE: collection-level Analysis, Clustering, and Explanation

Input: trace collection $\mathcal{T} = \{T_i\}_{i=1}^{N}$; taxonomy $\mathcal{Y}_d$ with its scanner set; sample sizes $n_0 = 2$, $n_{\max} = 5$
Output: localized issues $\widehat{\mathcal{I}}$, same-fix clusters $\widehat{\mathcal{C}}$, validated hypotheses $\widehat{\mathcal{H}}$, report

`/* Part 1: Analysis -- Failure Analyzer */`

1 **foreach** *trace* $T_i \in \mathcal{T}$ **do**
2 $W_i \leftarrow$ canonicalize T_i with verbatim `span_ids`, compact payloads, render workspace `// ground truth kept outside` W_i
3 chunks $\leftarrow$ episode-aware packing of W_i
4 candidates $\leftarrow$ proposals of every scanner over the chunks $\cup$ rule-based anchor hits (if enabled; off in reported runs)
5 $\widehat{\mathcal{I}}_i \leftarrow$ post-process candidates: exact dedup $\rightarrow$ false-positive filters $\rightarrow$ within-category merge

`/* Part 2: Clustering -- same-fix groups and representatives */`

6 seed groups $\leftarrow$ group $\widehat{\mathcal{I}} = \bigcup_i \widehat{\mathcal{I}}_i$ by signature (category, span kind, tool, error token)
7 $\widehat{\mathcal{C}} \leftarrow$ one LLM merge pass over the seed groups `// merge only, never split`
8 **foreach** *cluster* $C_k \in \widehat{\mathcal{C}}$ **do** $T_k^{\star} \leftarrow \arg\max_{T \in \mathrm{Traces}(C_k)} U(T; C_k)$ (Eq. 2)

`/* Part 3: Explanation -- Root-Cause Analyzer and Validator */`

9 **foreach** *cluster* $C_k \in \widehat{\mathcal{C}}$ **do**
10 $H_k \leftarrow$ Root-Cause Analyzer investigates the workspace of $T_k^{\star}$; accept only if $E_k \neq \varnothing$ and $g(H_k, T_k^{\star}) = 1$ `// evidence gate, Eq. 1`
11 $\mathcal{S}_k \leftarrow$ seeded sample of n_0 other members; $V(H_k, T_j) \leftarrow$ Cross-Trace Validator verdict for each $T_j \in \mathcal{S}_k$
12 **if** *some verdict is not* CONFIRMED **then** grow $\mathcal{S}_k$ to at most $n_{\max}$ members and validate the new ones
13 $\mathrm{Status}(H_k, C_k), \mathrm{Agree}(H_k, C_k) \leftarrow$ Eq. 3; **if** *status is* MIXED *or* REFUTED **then** re-diagnose: regroup C_k by observed mechanism

14 **return** $\widehat{\mathcal{I}}, \widehat{\mathcal{C}}, \widehat{\mathcal{H}} = \{(C_k, T_k^{\star}, H_k, \{V(H_k, T_j)\}_{T_j \in \mathcal{S}_k})\}_k$, and the rendered report

3.1 PREPARING LONG TRACES FOR DIAGNOSIS

Preparation. The Failure Analyzer (Lines 1–5 of Algorithm 1) begins with preparation. Tested adapters for TRAIL, AgentErrorBench, OpenInference-style exports (OpenInference Authors, 2026), and TELBench map each source into one canonical trace whose spans keep their verbatim source and parent identifiers, order, kind, status, attributes, and messages (OpenTelemetry Authors, 2026). Four label-free compaction rules then shrink payloads (base64 bytes become typed placeholders, empty wrappers leave the span files but stay in the tree, repeated tool schemas become references, and consecutive same-scope LLM calls keep only their new message suffix), and the result is rendered as a workspace directory W_i with an overview, a span tree, and one JSON file per span. Ground truth is stored outside W_i, label-bearing metadata is never serialized into prompts, and the Root-Cause Analyzer's tools cannot leave W_i.

Episode-aware chunking. Flat chunking can separate an LLM decision from the tool actions it triggered, so each trace is first segmented deterministically into *episodes*: one LLM "think" together with its descendant actions, or a sub-agent session header. A new episode starts whenever the agent scope changes or a second LLM call appears in the current scope; no model is involved. Chunks pack whole episodes up to a token budget and a span cap, flushing preferentially at session edges, and a deterministic *truncation rescue* reattaches small windows around error tokens that middle-out truncation of long fields would hide (boundary indicator and flush predicate in Appendix B).

3.2 ANALYSIS: LOCALIZING FAILURES WITH SPECIALIZED DETECTORS

Taxonomy-conditioned scanners. Each scanner ω is a prompted call of the same backbone that owns one slice $\mathcal{Y}_{d,\omega}$ of the taxonomy and may only report labels from that slice at span identifiers of its chunk. How the slices are cut follows the label space. TRAIL's 21 leaves name mutually exclusive failure kinds, so six scanners partition them (Equation A.6), and AgentErrorBench's module/type labels are likewise split into one scanner per module. TELBench's 18 labels are competing descriptions of one misstep, so two overlapping scanners each see the whole vocabulary

(Equation B.3); in the additional TRAIL/TELBench comparison, transplanting the partition to TELBench lowered joint accuracy on XBench (Appendix B).

Rule-based anchors and candidate union. Some failures are already stated by telemetry, for example a span marked ERROR with a known status signature; rule-based anchors map such events into the scored vocabulary without touching scanner output. The candidate set of a trace is the multiset union of every scanner's proposals over its chunks and, where a workflow enables them, the anchor hits; no reported benchmark run enables anchors. A separate zero-LLM structural channel records latency and chain anomalies beside these under its own channel tag and never enters scoring.

Provenance-preserving post-processing. Candidates then pass, in order, through exact deduplication, an optional per-location collapse, three model-assisted false-positive filters for recurrent confounders (dates fabricated as part of reasoning, expected test failures during debugging, and exploratory searches that are normal behavior), and a same-event merger whose within-category guard never folds different scored categories together, because one span can carry several labels. Items removed by any stage stay in the issue file with their status and reason, so every removal can be audited (Appendix B).

3.3 Clustering: Discovering Same-Fix Failure Groups

Interpretable signatures. Clustering (Lines 6–8) starts from a readable signature so that an engineer can see why two issues were grouped. The signature of an issue is the tuple (category, kind of the located span, tool of the located span, normalized error token), where the error token is extracted from the evidence and summary by deterministic regular expressions; issues with identical signatures form seed groups across traces.

Conservative merging. One LLM call then proposes which seed groups an engineer would fix together. The call is conservative in two ways: it may merge groups but never splits a seed group, and it runs once, so a model failure leaves the deterministic groups intact.

Representative selection. Each cluster is investigated through one representative trace, chosen by a transparent score rather than a learned model. With $\mathrm{impact}_k(T) \in \{0, 1, 2\}$ the highest severity among T's member issues, $\mathrm{density}_k(T)$ their count, and $\mathrm{size}(T)$ the trace's token estimate,

$$\begin{aligned} U(T; C_k) &= 3\,\mathrm{impact}_k(T) + \mathrm{density}_k(T) - 0.5 \log_{10}\big(\max\{10, \mathrm{size}(T)\}\big), \\ T_k^\star &= \operatorname*{arg\,max}_{T \in \mathrm{Traces}(C_k)} U(T; C_k), \end{aligned} \tag{2}$$

which favors severe, evidence-rich cases and mildly penalizes very large traces; the weights are fixed a priori and were not tuned on any evaluation set. The remaining members become validation candidates.

3.4 Explanation: Validating Evidence-Grounded Hypotheses

Root-Cause Analyzer. Explanation (Lines 9–13) begins with the Root-Cause Analyzer, a bounded tool-using agent confined to the representative's workspace. It has deterministic list, read, and search tools, two cached model-powered tools that summarize or filter the trace, and native function calling with a ReAct fallback (Yao et al., 2023); its terminal tool must return a root cause, a mechanism, evidence pairs, a confidence, and a fix sketch (Appendix F). An *evidence gate* accepts the hypothesis only when every cited `span_id` resolves in the workspace and its quotation is recoverable there (Equation 1); otherwise at most two re-quotation retries follow. The independent evaluator later repeats the same check from persisted artifacts.

Cross-Trace Validator. A lightweight validator receives the hypothesis and the workspace of a secondary member, never its annotation, and rules the mechanism confirmed, partial, or refuted while citing what it observed. Two members are sampled first; any non-confirming verdict expands the sample to at most five.

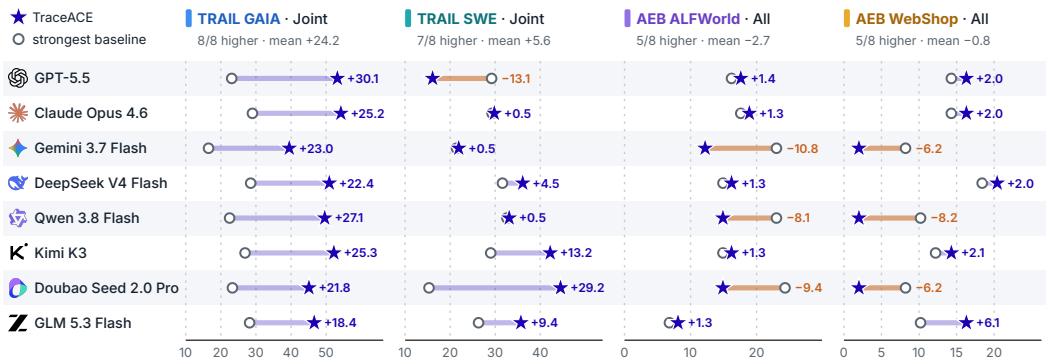


Figure 2: TRACEACE against the strongest baseline under each backbone. Each row compares TRACEACE (star) with the best of the seven baselines (circle) on the headline metric of each cohort in Table 1. Labels give the difference in points; orange segments mark backbones on which a baseline is higher.

Status and re-diagnosis. Every verdict is persisted. For n verdicts with n_c confirmed and n_r refuted, the persisted status and agreement rate are

$$\text{Status}(H_k, C_k) = \begin{cases} \text{CONFIRMED}, & n_c = n, \\ \text{REFUTED}, & n_c = 0 \ \wedge \ 2n_r \geq n, \\ \text{MIXED}, & \text{otherwise}, \end{cases} \qquad \text{Agree}(H_k, C_k) = \frac{n_c}{n}, \tag{3}$$

so any confirmed member prevents a refuted summary. A MIXED or REFUTED status triggers an optional re-diagnosis that regroups the cluster by the mechanisms actually observed (Appendix B).

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets. TRAIL (Deshpande et al., 2025) provides traces derived from GAIA (Mialon et al., 2024) and SWE-bench (Jimenez et al., 2024), with a 21-leaf error taxonomy and exact span locations; we use all 116 GAIA and 31 SWE traces. AgentErrorBench (AEB; Zhu et al., 2025) supplies ALFWorld and WebShop trajectories with critical-step annotations; we use 100 ALFWorld and 50 WebShop trajectories, of which 74 and 49 carry type labels for the All metric. Every method sees the same prepared traces within each backbone.

Models and methods. The eight backbones are GPT-5.5, Claude Opus 4.6, Gemini 3.7 Flash, DeepSeek V4 Flash, Qwen 3.8 Flash, Kimi K3, Doubao Seed 2.0 Pro, and GLM 5.3 Flash. Table 1 compares LLM, ReAct (Yao et al., 2023), CORRECT (Yu et al., 2025), AgentDebug (Zhu et al., 2025), AgentRx (Barke et al., 2026), AgentDebugX (Zhu et al., 2026), Agentic CLEAR (Yehudai et al., 2026), and TRACEACE. LLM is a single-pass trace judge; ReAct adds tool-based investigation. The external methods retain their diagnostic mechanisms with dataset-specific input and output adapters. AgentDebugX retains its root-only prediction when scored against TRAIL's set of annotated errors.

Metrics. TRAIL reports weighted category F1, exact localization (Loc.), and joint accuracy (Joint). AEB reports critical-step accuracy (Step), step-and-module accuracy (Step+M), and joint step, module, and type accuracy (All). All preserves the original type-label mask. Scores are percentages, and subscripts give changes from LLM within the same model and cohort.

4.2 MAIN COMPARISON

Multi-error detection. TRACEACE has the highest joint accuracy on TRAIL GAIA under all eight backbones, with a mean gain of 24.2 points over the strongest baseline in each block (range 18.4–30.1; Figure 2); it also leads weighted F1 and exact localization on GAIA under every backbone. On TRAIL SWE it has the highest joint accuracy under seven of eight backbones (mean gain 5.6

Table 1: Main comparison. **(A)** Multi-error detection on TRAIL; **(B)** critical root-cause attribution on AgentErrorBench (AEB; Zhu et al., 2025). All eight backbones are evaluated on the full cohorts: TRAIL GAIA (116) and SWE (31), AEB ALFWorld (100) and WebShop (50); AEB All is scored over the 74 and 49 traces with type labels. Scores are percentages. Subscripts give changes from LLM; bold/underline mark the best/second-best values within each model and cohort.

		(A) Multi-error detection						(B) Critical root-cause attribution					
		TRAIL (GAIA)			TRAIL (SWE)			AEB (ALFWorld)			AEB (WebShop)		
Model	**Method**	F1↑	Loc.↑	Joint↑	F1↑	Loc.↑	Joint↑	Step↑	Step+M↑	All↑	Step↑	Step+M↑	All↑
gpt-5.5	LLM	44.0	64.2	23.1	55.0	25.0	5.2	37.0	14.0	14.9	36.0	8.0	6.1
	ReAct	31.4$_{-12.6}$	59.3$_{-4.9}$	16.1$_{-7.0}$	46.8$_{-8.2}$	22.4$_{-2.6}$	4.0$_{-1.2}$	39.0$_{+2.0}$	15.0$_{+1.0}$	14.9$_{\pm0.0}$	28.0$_{-8.0}$	8.0$_{\pm0.0}$	6.1$_{\pm0.0}$
	CORRECT	31.0$_{-13.0}$	59.3$_{-4.9}$	12.6$_{-10.5}$	38.4$_{-16.6}$	9.3$_{-15.7}$	0.3$_{-4.9}$	37.0$_{\pm0.0}$	17.0$_{+3.0}$	16.2$_{+1.3}$	34.0$_{-2.0}$	10.0$_{+2.0}$	8.2$_{+2.1}$
	AgentDebug	34.9$_{-9.1}$	82.6$_{+18.4}$	14.0$_{-9.1}$	47.8$_{-7.2}$	48.5$_{+23.5}$	16.8$_{+11.6}$	42.0$_{+5.0}$	15.0$_{+1.0}$	9.5$_{-5.4}$	34.0$_{-2.0}$	4.0$_{-4.0}$	4.1$_{-2.0}$
	AgentRx	32.7$_{-11.3}$	64.2$_{\pm0.0}$	12.4$_{-10.7}$	41.5$_{-13.5}$	31.4$_{+6.4}$	5.4$_{+0.2}$	38.0$_{+1.0}$	15.0$_{+1.0}$	14.9$_{\pm0.0}$	34.0$_{-2.0}$	14.0$_{+6.0}$	14.3$_{+8.2}$
	AgentDebugX	16.1$_{-27.9}$	33.2$_{-31.0}$	3.6$_{-19.5}$	10.6$_{-44.4}$	7.8$_{-17.2}$	0.0$_{-5.2}$	21.0$_{-16.0}$	6.0$_{-8.0}$	5.4$_{-9.5}$	28.0$_{-8.0}$	4.0$_{-4.0}$	4.1$_{-2.0}$
	Agentic CLEAR	37.4$_{-6.6}$	62.4$_{-1.8}$	18.1$_{-5.0}$	53.8$_{-1.2}$	**65.3**$_{+40.3}$	**29.2**$_{+24.0}$	42.0$_{+5.0}$	17.0$_{+3.0}$	13.5$_{-1.4}$	22.0$_{-14.0}$	8.0$_{\pm0.0}$	4.1$_{-2.0}$
	⋆ TRACEACE	**58.7**$_{+14.7}$	**88.5**$_{+24.3}$	**53.2**$_{+30.1}$	**62.1**$_{+7.1}$	44.9$_{+19.9}$	16.1$_{+10.9}$	**43.0**$_{+6.0}$	**18.0**$_{+4.0}$	**17.6**$_{+2.7}$	**38.0**$_{+2.0}$	**16.0**$_{+8.0}$	**16.3**$_{+10.2}$
claude-opus-4-6	LLM	40.9	42.4	17.2	34.8	8.7	0.5	42.0	16.0	16.2	40.0	8.0	6.1
	ReAct	45.3$_{+4.4}$	73.5$_{+31.1}$	29.0$_{+11.8}$	39.1$_{+4.3}$	29.9$_{+21.2}$	5.1$_{+4.6}$	44.0$_{+2.0}$	18.0$_{+2.0}$	16.2$_{\pm0.0}$	32.0$_{-8.0}$	8.0$_{\pm0.0}$	6.1$_{\pm0.0}$
	CORRECT	17.2$_{-23.7}$	52.4$_{+10.0}$	10.8$_{-6.4}$	10.6$_{-24.2}$	1.5$_{-7.2}$	0.5$_{\pm0.0}$	41.0$_{-1.0}$	19.0$_{+3.0}$	17.6$_{+1.4}$	36.0$_{-4.0}$	10.0$_{+2.0}$	8.2$_{+2.1}$
	AgentDebug	39.6$_{-1.3}$	67.2$_{+24.8}$	22.6$_{+5.4}$	46.4$_{+11.6}$	41.2$_{+32.5}$	7.1$_{+6.6}$	**47.0**$_{+5.0}$	17.0$_{+1.0}$	10.8$_{-5.4}$	36.0$_{-4.0}$	6.0$_{-2.0}$	4.1$_{-2.0}$
	AgentRx	28.0$_{-12.9}$	70.2$_{+27.8}$	20.6$_{+3.4}$	44.0$_{+9.2}$	47.4$_{+38.7}$	13.6$_{+13.1}$	43.0$_{+1.0}$	18.0$_{+2.0}$	16.2$_{\pm0.0}$	38.0$_{-2.0}$	16.0$_{+8.0}$	14.3$_{+8.2}$
	AgentDebugX	21.4$_{-19.5}$	50.0$_{+7.6}$	11.4$_{-5.8}$	13.8$_{-21.0}$	3.7$_{-5.0}$	0.4$_{-0.1}$	25.0$_{-17.0}$	8.0$_{-8.0}$	6.8$_{-9.4}$	32.0$_{-8.0}$	6.0$_{-2.0}$	4.1$_{-2.0}$
	Agentic CLEAR	34.8$_{-6.1}$	66.7$_{+24.3}$	18.7$_{+1.5}$	50.0$_{+15.2}$	68.7$_{+60.0}$	29.3$_{+28.8}$	**47.0**$_{+5.0}$	19.0$_{+3.0}$	14.9$_{-1.3}$	24.0$_{-16.0}$	8.0$_{\pm0.0}$	4.1$_{-2.0}$
	⋆ TRACEACE	**56.3**$_{+15.4}$	**90.1**$_{+47.7}$	**54.2**$_{+37.0}$	**50.5**$_{+15.7}$	**71.7**$_{+63.0}$	**29.8**$_{+29.3}$	40.0$_{-2.0}$	**20.0**$_{+4.0}$	**18.9**$_{+2.7}$	**42.0**$_{+2.0}$	**18.0**$_{+10.0}$	**16.3**$_{+10.2}$
gemini-3.7-flash	LLM	27.9	33.9	10.0	32.3	19.2	3.0	**44.0**	**18.0**	**23.0**	10.0	2.0	2.0
	ReAct	31.9$_{+4.0}$	42.9$_{+9.0}$	14.3$_{+4.3}$	18.9$_{-13.4}$	8.8$_{-10.4}$	0.5$_{-2.5}$	38.0$_{-6.0}$	12.0$_{-6.0}$	5.4$_{-17.6}$	14.0$_{+4.0}$	2.0$_{\pm0.0}$	2.0$_{\pm0.0}$
	CORRECT	13.2$_{-14.7}$	52.0$_{+18.1}$	9.1$_{-0.9}$	8.5$_{-23.8}$	1.0$_{-18.2}$	0.4$_{-2.6}$	34.0$_{-10.0}$	16.0$_{-2.0}$	14.9$_{-8.1}$	16.0$_{+6.0}$	2.0$_{\pm0.0}$	2.0$_{\pm0.0}$
	AgentDebug	34.2$_{+6.3}$	44.3$_{+10.4}$	13.1$_{+3.1}$	39.8$_{+7.5}$	29.0$_{+9.8}$	4.9$_{+1.9}$	43.0$_{-1.0}$	15.0$_{-3.0}$	12.2$_{-10.8}$	22.0$_{+12.0}$	**8.0**$_{+6.0}$	**8.2**$_{+6.2}$
	AgentRx	23.2$_{-4.7}$	58.7$_{+24.8}$	16.5$_{+6.5}$	36.1$_{+3.8}$	31.8$_{+12.6}$	8.8$_{+5.8}$	28.0$_{-16.0}$	11.0$_{-7.0}$	9.5$_{-13.5}$	22.0$_{+12.0}$	4.0$_{+2.0}$	2.0$_{\pm0.0}$
	AgentDebugX	20.1$_{-7.8}$	50.8$_{+16.9}$	13.6$_{+3.6}$	10.6$_{-21.7}$	2.1$_{-17.1}$	0.3$_{-2.7}$	24.0$_{-20.0}$	2.0$_{-16.0}$	2.7$_{-20.3}$	20.0$_{+10.0}$	2.0$_{\pm0.0}$	0.0$_{-2.0}$
	Agentic CLEAR	23.3$_{-4.6}$	44.5$_{+10.6}$	7.3$_{-2.7}$	**42.9**$_{+10.6}$	**55.3**$_{+36.1}$	21.4$_{+18.4}$	43.0$_{-1.0}$	16.0$_{-2.0}$	13.5$_{-9.5}$	18.0$_{+8.0}$	**8.0**$_{+6.0}$	0.0$_{-2.0}$
	⋆ TRACEACE	**45.5**$_{+17.6}$	**69.1**$_{+35.2}$	**39.5**$_{+29.5}$	28.8$_{-3.5}$	26.1$_{+6.9}$	**21.9**$_{+18.9}$	29.0$_{-15.0}$	11.0$_{-7.0}$	12.2$_{-10.8}$	**24.0**$_{+14.0}$	2.0$_{\pm0.0}$	2.0$_{\pm0.0}$
DeepSeek V4 Flash	LLM	41.4	55.9	19.6	38.5	24.1	4.3	22.0	5.0	5.4	38.0	8.0	8.2
	ReAct	44.5$_{+3.1}$	62.1$_{+6.2}$	28.5$_{+8.9}$	37.1$_{-1.4}$	22.7$_{-1.4}$	5.1$_{+0.8}$	26.0$_{+4.0}$	12.0$_{+7.0}$	14.9$_{+9.5}$	22.0$_{-16.0}$	10.0$_{+2.0}$	8.2$_{\pm0.0}$
	CORRECT	15.7$_{-25.7}$	53.6$_{-2.3}$	14.4$_{-5.2}$	15.7$_{-22.8}$	0.2$_{-23.9}$	0.1$_{-4.2}$	29.0$_{+7.0}$	7.0$_{+2.0}$	6.8$_{+1.4}$	38.0$_{\pm0.0}$	10.0$_{+2.0}$	8.2$_{\pm0.0}$
	AgentDebug	35.1$_{-6.3}$	65.2$_{+9.3}$	25.7$_{+6.1}$	34.7$_{-3.8}$	31.1$_{+7.0}$	3.0$_{-1.3}$	38.0$_{+16.0}$	16.0$_{+11.0}$	13.5$_{+8.1}$	38.0$_{\pm0.0}$	2.0$_{-6.0}$	0.0$_{-8.2}$
	AgentRx	30.1$_{-11.3}$	61.4$_{+5.5}$	21.9$_{+2.3}$	45.2$_{+6.7}$	38.6$_{+14.5}$	11.0$_{+6.7}$	28.0$_{+6.0}$	13.0$_{+8.0}$	13.5$_{+8.1}$	30.0$_{-8.0}$	18.0$_{+10.0}$	18.4$_{+10.2}$
	AgentDebugX	17.0$_{-24.4}$	36.6$_{-19.3}$	7.5$_{-12.1}$	13.2$_{-25.3}$	5.0$_{-19.1}$	1.2$_{-3.1}$	8.0$_{-14.0}$	2.0$_{-3.0}$	2.7$_{-2.7}$	30.0$_{-8.0}$	2.0$_{-6.0}$	0.0$_{-8.2}$
	Agentic CLEAR	31.5$_{-9.9}$	50.5$_{-5.4}$	18.5$_{-1.1}$	47.7$_{+9.2}$	74.2$_{+50.1}$	31.6$_{+27.3}$	**44.0**$_{+22.0}$	17.0$_{+12.0}$	14.9$_{+9.5}$	20.0$_{-18.0}$	8.0$_{\pm0.0}$	8.2$_{\pm0.0}$
	⋆ TRACEACE	**53.9**$_{+12.5}$	**88.4**$_{+32.5}$	**50.9**$_{+31.3}$	**65.1**$_{+26.6}$	**74.7**$_{+50.6}$	**36.1**$_{+31.8}$	26.0$_{+4.0}$	**18.0**$_{+13.0}$	**16.2**$_{+10.8}$	**40.0**$_{+2.0}$	**20.0**$_{+12.0}$	**20.4**$_{+12.2}$
Qwen 3.8 Flash	LLM	42.3	52.2	22.5	35.6	26.3	3.3	31.0	12.0	13.5	**44.0**	2.0	2.0
	ReAct	41.2$_{-1.1}$	73.1$_{+20.9}$	20.6$_{-1.9}$	45.9$_{+10.3}$	35.9$_{+9.6}$	5.8$_{+2.5}$	38.0$_{+7.0}$	**19.0**$_{+7.0}$	**23.0**$_{+9.5}$	14.0$_{-30.0}$	2.0$_{\pm0.0}$	2.0$_{\pm0.0}$
	CORRECT	14.8$_{-27.5}$	32.6$_{-19.6}$	6.7$_{-15.8}$	11.9$_{-23.7}$	3.0$_{-23.3}$	1.5$_{-1.8}$	25.0$_{-6.0}$	13.0$_{+1.0}$	14.9$_{+1.4}$	36.0$_{-8.0}$	8.0$_{+6.0}$	8.2$_{+6.2}$
	AgentDebug	36.7$_{-5.6}$	69.1$_{+16.9}$	18.7$_{-3.8}$	53.9$_{+18.3}$	50.9$_{+24.6}$	12.4$_{+9.1}$	27.0$_{-4.0}$	6.0$_{-6.0}$	4.1$_{-9.4}$	16.0$_{-28.0}$	2.0$_{\pm0.0}$	0.0$_{-2.0}$
	AgentRx	21.7$_{-20.6}$	57.0$_{+4.8}$	9.7$_{-12.8}$	33.8$_{-1.8}$	16.2$_{-10.1}$	1.4$_{-1.9}$	31.0$_{\pm0.0}$	8.0$_{-4.0}$	6.8$_{-6.7}$	36.0$_{-8.0}$	**10.0**$_{+8.0}$	**10.2**$_{+8.2}$
	AgentDebugX	17.7$_{-24.6}$	37.8$_{-14.4}$	8.1$_{-14.4}$	14.5$_{-21.1}$	4.0$_{-22.3}$	0.1$_{-3.2}$	18.0$_{-13.0}$	2.0$_{-10.0}$	2.7$_{-10.8}$	28.0$_{-16.0}$	2.0$_{\pm0.0}$	0.0$_{-2.0}$
	Agentic CLEAR	31.1$_{-11.2}$	57.8$_{+5.6}$	14.4$_{-8.1}$	50.9$_{+15.3}$	69.6$_{+43.3}$	32.6$_{+29.3}$	27.0$_{-4.0}$	6.0$_{-6.0}$	5.4$_{-8.1}$	6.0$_{-38.0}$	2.0$_{\pm0.0}$	0.0$_{-2.0}$
	⋆ TRACEACE	**51.5**$_{+9.2}$	**86.1**$_{+33.9}$	**49.6**$_{+27.1}$	**68.2**$_{+32.6}$	**76.0**$_{+49.7}$	**33.1**$_{+29.8}$	**39.0**$_{+8.0}$	12.0$_{\pm0.0}$	14.9$_{+1.4}$	16.0$_{-28.0}$	2.0$_{\pm0.0}$	2.0$_{\pm0.0}$
Kimi K3	LLM	46.5	53.0	23.9	43.5	26.9	4.6	34.0	12.0	12.2	38.0	8.0	6.1
	ReAct	47.9$_{+1.4}$	68.3$_{+15.3}$	26.9$_{+3.0}$	43.7$_{+0.2}$	26.5$_{-0.4}$	5.0$_{+0.4}$	36.0$_{+2.0}$	13.0$_{+1.0}$	12.2$_{\pm0.0}$	28.0$_{-10.0}$	8.0$_{\pm0.0}$	6.1$_{\pm0.0}$
	CORRECT	18.3$_{-28.2}$	47.6$_{-5.4}$	10.1$_{-13.8}$	12.1$_{-31.4}$	1.4$_{-25.5}$	0.5$_{-4.1}$	35.0$_{+1.0}$	16.0$_{+4.0}$	14.9$_{+2.7}$	34.0$_{-4.0}$	8.0$_{\pm0.0}$	8.2$_{+2.1}$
	AgentDebug	41.8$_{-4.7}$	62.3$_{+9.3}$	21.1$_{-2.8}$	50.4$_{+6.9}$	37.7$_{+10.8}$	7.0$_{+2.4}$	40.0$_{+6.0}$	13.0$_{+1.0}$	8.1$_{-4.1}$	34.0$_{-4.0}$	4.0$_{-4.0}$	4.1$_{-2.0}$
	AgentRx	30.3$_{-16.2}$	64.4$_{+11.4}$	18.9$_{-5.0}$	49.0$_{+5.5}$	43.0$_{+16.1}$	13.4$_{+8.8}$	35.0$_{+1.0}$	13.0$_{+1.0}$	12.2$_{\pm0.0}$	36.0$_{-2.0}$	14.0$_{+6.0}$	12.2$_{+6.1}$
	AgentDebugX	23.1$_{-23.4}$	44.0$_{-9.0}$	10.4$_{-13.5}$	16.0$_{-27.5}$	3.2$_{-23.7}$	0.4$_{-4.2}$	19.0$_{-15.0}$	6.0$_{-6.0}$	5.4$_{-6.8}$	28.0$_{-10.0}$	4.0$_{-4.0}$	4.1$_{-2.0}$
	Agentic CLEAR	37.0$_{-9.5}$	61.5$_{+8.5}$	17.4$_{-6.5}$	54.3$_{+10.8}$	65.3$_{+38.4}$	29.0$_{+24.4}$	39.0$_{+5.0}$	15.0$_{+3.0}$	12.2$_{\pm0.0}$	22.0$_{-16.0}$	8.0$_{\pm0.0}$	4.1$_{-2.0}$
	⋆ TRACEACE	**58.3**$_{+11.8}$	**88.2**$_{+35.2}$	**52.2**$_{+28.3}$	**74.0**$_{+30.5}$	**75.4**$_{+48.5}$	**42.2**$_{+37.6}$	**41.0**$_{+7.0}$	**17.0**$_{+5.0}$	**16.2**$_{+4.0}$	**40.0**$_{+2.0}$	**16.0**$_{+8.0}$	**14.3**$_{+8.2}$
Doubao Seed 2.0 Pro	LLM	40.5	40.8	21.3	29.2	14.9	1.7	24.0	5.0	5.4	30.0	2.0	2.0
	ReAct	41.0$_{+0.5}$	61.6$_{+20.8}$	23.3$_{+2.0}$	33.7$_{+4.5}$	12.7$_{-2.2}$	0.5$_{-1.2}$	34.0$_{+10.0}$	5.0$_{\pm0.0}$	5.4$_{\pm0.0}$	28.0$_{-2.0}$	2.0$_{\pm0.0}$	2.0$_{\pm0.0}$
	CORRECT	17.5$_{-23.0}$	27.9$_{-12.9}$	4.5$_{-16.8}$	6.4$_{-22.8}$	1.1$_{-13.8}$	0.1$_{-1.6}$	38.0$_{+14.0}$	**23.0**$_{+18.0}$	**24.3**$_{+18.9}$	36.0$_{+6.0}$	**10.0**$_{+8.0}$	**8.2**$_{+6.2}$
	AgentDebug	37.0$_{-3.5}$	61.7$_{+20.9}$	15.3$_{-6.0}$	41.7$_{+12.5}$	22.1$_{+7.2}$	4.1$_{+2.4}$	**44.0**$_{+20.0}$	9.0$_{-4.0}$	4.1$_{-1.3}$	38.0$_{+8.0}$	2.0$_{\pm0.0}$	0.0$_{-2.0}$
	AgentRx	21.8$_{-18.7}$	60.6$_{+19.8}$	14.4$_{-6.9}$	28.9$_{-0.3}$	57.0$_{+42.1}$	15.3$_{+13.6}$	19.0$_{-5.0}$	5.0$_{\pm0.0}$	4.1$_{-1.3}$	18.0$_{-12.0}$	4.0$_{+2.0}$	2.0$_{\pm0.0}$
	AgentDebugX	21.0$_{-19.5}$	23.1$_{-17.7}$	1.6$_{-19.7}$	14.1$_{-15.1}$	0.4$_{-14.5}$	0.0$_{-1.7}$	18.0$_{-6.0}$	9.0$_{+4.0}$	10.8$_{+5.4}$	28.0$_{-2.0}$	8.0$_{+6.0}$	**8.2**$_{+6.2}$
	Agentic CLEAR	30.7$_{-9.8}$	63.1$_{+22.3}$	8.7$_{-12.6}$	36.0$_{+6.8}$	23.8$_{+8.9}$	7.1$_{+5.4}$	36.0$_{+12.0}$	16.0$_{+11.0}$	13.5$_{+8.1}$	26.0$_{-4.0}$	2.0$_{\pm0.0}$	0.0$_{-2.0}$
	⋆ TRACEACE	**50.7**$_{+10.2}$	**87.8**$_{+47.0}$	**45.1**$_{+23.8}$	**68.6**$_{+39.4}$	**76.2**$_{+61.3}$	**44.5**$_{+42.8}$	34.0$_{+10.0}$	12.0$_{+7.0}$	14.9$_{+9.5}$	**46.0**$_{+16.0}$	2.0$_{\pm0.0}$	2.0$_{\pm0.0}$
GLM 5.3 Flash	LLM	43.7	60.9	28.2	43.6	25.8	5.2	28.0	8.0	5.4	32.0	8.0	8.2
	ReAct	42.0$_{-1.7}$	81.1$_{+20.2}$	27.7$_{-0.5}$	46.4$_{+2.8}$	31.1$_{+5.3}$	7.8$_{+2.6}$	19.0$_{-9.0}$	6.0$_{-2.0}$	5.4$_{\pm0.0}$	36.0$_{+4.0}$	10.0$_{+2.0}$	8.2$_{\pm0.0}$
	CORRECT	14.9$_{-28.8}$	53.4$_{-7.5}$	8.9$_{-19.3}$	9.8$_{-33.8}$	0.2$_{-25.6}$	0.1$_{-5.1}$	**29.0**$_{+1.0}$	7.0$_{-1.0}$	6.8$_{+1.4}$	16.0$_{-16.0}$	2.0$_{-6.0}$	2.0$_{-6.2}$
	AgentDebug	36.9$_{-6.8}$	60.2$_{-0.7}$	17.8$_{-10.4}$	45.4$_{+1.8}$	29.5$_{+3.7}$	2.9$_{-2.3}$	28.0$_{\pm0.0}$	9.0$_{+1.0}$	4.1$_{-1.3}$	30.0$_{-2.0}$	2.0$_{-6.0}$	0.0$_{-8.2}$
	AgentRx	21.0$_{-22.7}$	54.3$_{-6.6}$	10.1$_{-18.1}$	34.9$_{-8.7}$	41.9$_{+16.1}$	10.9$_{+5.7}$	19.0$_{-9.0}$	5.0$_{-3.0}$	4.1$_{-1.3}$	24.0$_{-8.0}$	10.0$_{+2.0}$	10.2$_{+2.0}$
	AgentDebugX	18.1$_{-25.6}$	48.6$_{-12.3}$	11.5$_{-16.7}$	10.4$_{-33.2}$	1.5$_{-24.3}$	0.1$_{-5.1}$	8.0$_{-20.0}$	2.0$_{-6.0}$	2.7$_{-2.7}$	8.0$_{-24.0}$	2.0$_{-6.0}$	0.0$_{-8.2}$
	Agentic CLEAR	35.9$_{-7.8}$	68.5$_{+7.6}$	25.1$_{-3.1}$	51.5$_{+7.9}$	**72.9**$_{+47.1}$	26.3$_{+21.1}$	24.0$_{-4.0}$	7.0$_{-1.0}$	5.4$_{\pm0.0}$	12.0$_{-20.0}$	2.0$_{-6.0}$	0.0$_{-8.2}$
	⋆ TRACEACE	**51.3**$_{+7.6}$	**84.2**$_{+23.3}$	**46.6**$_{+18.4}$	**66.5**$_{+22.9}$	61.7$_{+35.9}$	**35.7**$_{+30.5}$	22.0$_{-6.0}$	**10.0**$_{+2.0}$	**8.1**$_{+2.7}$	**38.0**$_{+6.0}$	**18.0**$_{+10.0}$	**16.3**$_{+8.1}$

Table 2: Component ablations. DeepSeek V4 Flash on the full cohorts. Each row removes or replaces one component; Analysis variants are scored on TRAIL and Explanation variants on AEB (All over 74 / 49 typed traces).

Analysis	TRAIL (GAIA)			TRAIL (SWE)		
Variant	F1↑	Loc.↑	Joint↑	F1↑	Loc.↑	Joint↑
Flat chunks	**54.1**	87.8	50.5	64.7	75.0	35.8
Unified scanner	50.9	84.2	47.6	61.9	71.2	33.4
w/o FP filters	53.2	**89.2**	**51.4**	64.6	**75.4**	**36.6**
⋆ TRACEACE	53.9	88.4	50.9	**65.1**	74.7	36.1
Explanation	**AEB (ALFWorld)**			**AEB (WebShop)**		
Variant	Step↑	Step+M↑	All↑	Step↑	Step+M↑	All↑
w/o evidence gate	**27.0**	17.0	14.9	**42.0**	18.0	18.4
Random representative	23.0	16.0	13.5	38.0	18.0	18.4
w/o cross-trace validation	21.0	15.0	12.2	34.0	16.0	16.3
⋆ TRACEACE	26.0	**18.0**	**16.2**	40.0	**20.0**	**20.4**

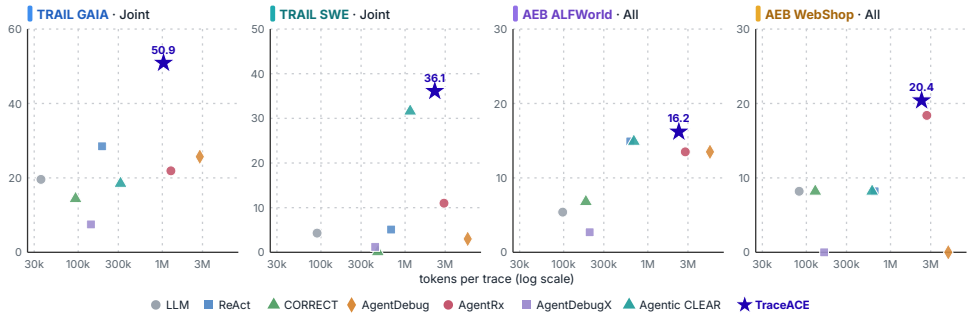


Figure 3: Accuracy against per-trace token cost. DeepSeek V4 Flash; accuracy from Table 1 and prompt-plus-completion tokens per trace from Table D.1, on a logarithmic axis. One-time setup is excluded.

points); under GPT-5.5, Agentic CLEAR is higher on localization and joint accuracy (29.2 vs. 16.1 joint).

Critical root-cause attribution. AEB is harder for every method, and the margins are small. TRACEACE has the highest All score under five of eight backbones on each environment, but trails the strongest baseline under the remaining three (Gemini 3.7 Flash, Qwen 3.8 Flash, and Doubao Seed 2.0 Pro), by 9.4 points on ALFWorld and 6.9 on WebShop on average; over all eight backbones the mean differences are -2.7 and -0.8. Critical-step accuracy is similarly mixed: TRACEACE leads Step on WebShop under seven backbones but on ALFWorld under only three, so we read (B) as parity with the strongest per-trace attributors.

4.3 Component Ablations

Table 2 (protocol in Appendix C.5) removes one component at a time. Replacing the specialized scanners with a single unified scanner costs the most on TRAIL (joint -3.3 on GAIA and -2.7 on SWE), while flat chunking and removing the false-positive filters change scores by less than one point. For explanation, removing cross-trace validation lowers All by 4.0 points on ALFWorld and 4.1 on WebShop, and a random representative lowers it by 2.7 and 2.0. Without the evidence gate, All drops by 1.3 and 2.0 points although Step rises, so the gate mainly filters unsupported attributions.

4.4 Inference and Setup Cost

Appendix Table D.1 separates per-trace inference from one-time setup. TRACEACE uses 1.0–2.3 million tokens and 53–81 calls per trace: 37–48% of AgentDebug's tokens, 13–16% fewer

Table 3: Collection-level diagnosis. DeepSeek V4 Flash on AEB ALFWorld (100) and WebShop (50). Reports counts primary RCA investigations; Cov. is the rate of non-empty final root-cause outputs, Ground. is evidence grounding, and All is root accuracy over 74 / 49 typed traces (%). Tok. covers all post-analysis stages per trace.

Dataset	Variant	Reports↓	Cov.↑	Ground.↑	All↑	Tok. (k)↓
AEB (ALFWorld)	Per-trace RCA	100	100.0	91.4	14.9	498.6
	Signature-only grouping	300	100.0	92.3	13.5	2242.1
	⋆ TRACEACE	140	100.0	93.3	16.2	1529.0
AEB (WebShop)	Per-trace RCA	50	100.0	88.3	18.4	685.5
	Signature-only grouping	125	100.0	89.3	16.3	2337.6
	⋆ TRACEACE	45	100.0	90.2	20.4	1612.3

than AgentRx on AEB, and 5–29 times those of single-pass LLM, CORRECT, and AgentDebugX; CORRECT adds a one-time setup of 10.7 million tokens. In Figure 3, the only methods that spend more tokens, AgentDebug and AgentRx, are less accurate on all four cohorts.

4.5 Collection-Level Diagnosis

Table 3 holds the analysis outputs fixed and compares investigating every trace, grouping by signature only, and the full cluster workflow. **TRACEACE** attains the highest root accuracy (16.2 vs. 14.9 on ALFWorld and 20.4 vs. 18.4 on WebShop against per-trace RCA) and evidence grounding, with the fewest reports on WebShop (45 vs. 50); on ALFWorld a trace can join several clusters, giving 140 reports against 300 for signature-only grouping.

5 Related Work

Trace representation and benchmarks. Distributed tracing represents a request as spans carrying context, timing, status, and attributes (OpenTelemetry Authors, 2026), and OpenInference adds model, retrieval, and tool attributes for AI applications (OpenInference Authors, 2026); **TRACEACE** consumes this span tree for post-hoc diagnosis. TRAIL (Deshpande et al., 2025) defines the typed-detection and exact-localization contract adopted here; TELBench (Wang et al., 2026) localizes harmful spans in deep-research trajectories; AgentRewardBench (Lù et al., 2025) judges web trajectories, and TraceElephant (Chen et al., 2026), Who&When (Zhang et al., 2025b), MAST (Cemri et al., 2025), and AgenTracer (Zhang et al., 2025a) attribute failures of multi-agent systems.

Trajectory diagnosis and controls. AgentDiagnose (Ou et al., 2025) scores trajectory-level competencies, AgentDebug (Zhu et al., 2025) pairs an error taxonomy with per-step analysis and corrective feedback, AgentRx (Barke et al., 2026) checks execution invariants to localize a critical failure, and CORRECT (Yu et al., 2025) distills recurrent errors into reusable schemata; AgentDebugX (Zhu et al., 2026) and Agentic CLEAR (Yehudai et al., 2026) complete the baselines of Table 1. **TRACEACE** differs in clustering the collection before investigation and replaying each hypothesis on other members. Every baseline is a transparent adaptation to a shared trace contract, not an exact reproduction (Appendix C).

6 Discussion and Conclusion

TRACEACE connects localized issue detection, collection-level grouping, and evidence-constrained root-cause investigation in one auditable workflow. It leads joint accuracy on TRAIL GAIA under every backbone and on TRAIL SWE under seven of eight, matches the strongest attributors on AEB, and gains most from scanner specialization and cross-trace validation. *Limitations.* Scores are single-run point estimates, external methods require task adapters, and without same-fix or intervention labels, grounding and cross-trace agreement do not establish causal correctness.

AI USE STATEMENT

We used generative AI tools only to aid or polish the writing of this paper, for example to improve grammar, wording, and readability. They were not used to design the method, run experiments, analyze data, or produce results. The authors reviewed and edited all text and take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

All runs use frozen source and configuration manifests, fixed trace manifests per cohort, and scoring receipts. Tables 1–3 and D.1 are generated by one script from a single versioned result file. Appendix B describes the pipeline, and Appendices C and F give the protocols and prompt contracts. Code, configurations, evaluation scripts, and table generators are provided in the Supplementary Material.

ETHICS STATEMENT

This work uses only publicly available benchmark traces. TRAIL is a gated dataset whose no-redistribution terms are respected: no trace content is released or reproduced beyond the short evidence quotations that appear in generated reports and figures, and both datasets are obtained from their official distributions under their respective terms. No human subjects were involved, and no personal data were collected or processed. The method produces diagnostic hypotheses about agent failures rather than verified facts; its explanations are anchored to quoted evidence, but they are intended to be validated by an engineer before any corrective action is taken, and the paper reports refutation and partial-agreement rates so that this limitation remains visible.

REFERENCES

Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. Recommending Root-Cause and mitigation steps for cloud incidents using large language models. *arXiv preprint arXiv:2301.03797*, 2023. URL https://arxiv.org/abs/2301.03797.

Alibaba Cloud. qwen3.8-flash model information, 2026. URL https://www.alibabacloud.com/help/en/model-studio/qwen3-8-flash.

Anthropic. Introducing Claude Opus 4.6, 2026. URL https://www.anthropic.com/news/claude-opus-4-6.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. AgentRx: Diagnosing AI agent failures from execution trajectories. *arXiv preprint arXiv:2602.02475*, 2026. doi: 10.48550/arXiv.2602.02475. URL https://arxiv.org/abs/2602.02475.

ByteDance Seed. Doubao-Seed-2.0, 2026. URL https://seed.bytedance.com/en/seed2.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do Multi-Agent LLM systems fail? *arXiv preprint arXiv:2503.13657*, 2025. URL https://arxiv.org/abs/2503.13657.

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. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://arxiv.org/abs/2604.22708. arXiv:2604.22708.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to Self-Debug. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2304.05128. arXiv:2304.05128.

Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents. *arXiv preprint arXiv:2305.15778*, 2023. URL https://arxiv.org/abs/2305.15778.

DeepSeek. DeepSeek-V4-Flash, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. TRAIL: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025. doi: 10.48550/arXiv.2505.08638. URL https://arxiv.org/abs/2505.08638.

Google. Gemini 3.7 Flash: Model documentation, 2026. URL https://ai.google.dev/gemini-api/docs/models/gemini-3.7-flash.

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. In *International Conference on Learning Representations*, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot Self-Correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.01798. arXiv:2310.01798.

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://openreview.net/forum?id=VTF8yNQM66.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2305.20050. arXiv:2305.20050.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 2024a. URL https://arxiv.org/abs/2307.03172. arXiv:2307.03172.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024b. URL https://arxiv.org/abs/2308.03688. arXiv:2308.03688.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025. doi: 10.48550/arXiv.2504.08942. URL https://arxiv.org/abs/2504.08942.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2401.13178. arXiv:2401.13178.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative

refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. SelfCheckGPT: Zero-Resource Black-Box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. URL https://arxiv.org/abs/2303.08896. arXiv:2303.08896.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=fibxvahvs3.

Moonshot AI. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026. doi: 10.48550/arXiv.2607.24653. URL https://arxiv.org/abs/2607.24653.

OpenAI. Introducing GPT-5.5, 2026. URL https://openai.com/index/introducing-gpt-5-5/.

OpenInference Authors. OpenInference: OpenTelemetry instrumentation for AI observability, 2026. URL https://github.com/Arize-ai/openinference.

OpenTelemetry Authors. Traces: The path of a request through your application, 2026. URL https://opentelemetry.io/docs/concepts/signals/traces/.

Tianyue Ou, Wanyao Guo, Apurva Gandhi, Graham Neubig, and Xiang Yue. AgentDiagnose: An open toolkit for diagnosing LLM agent trajectories. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 207–215. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-demos.15. URL https://aclanthology.org/2025.emnlp-demos.15/.

Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. Exploring LLM-based agents for root cause analysis. *arXiv preprint arXiv:2403.04123*, 2024. URL https://arxiv.org/abs/2403.04123.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2302.04761. arXiv:2302.04761.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2023.

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

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023. URL https://arxiv.org/abs/2309.02427.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022. URL https://arxiv.org/abs/2211.14275.

Jiaming Wang, Ziteng Feng, Jiangtao Wu, Ruihao Li, Qianqian Xie, Yuxiang Ren, He Zhu, Xueming Han, Fanyu Meng, Junlan Feng, and Jiaheng Liu. Where do deep-research agents go wrong? span-level error localization in agent trajectories. *arXiv preprint arXiv:2606.02060*, 2026. doi: 10.48550/arXiv.2606.02060. URL https://arxiv.org/abs/2606.02060.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023a. URL https://arxiv.org/abs/2308.11432.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634. Association for Computational Linguistics, 2023b. doi: 10.18653/v1/2023.acl-long.147. URL https://aclanthology.org/2023.acl-long.147/.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2407.16741. arXiv:2407.16741.

Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. RCAgent: Cloud root cause analysis by autonomous agents with Tool-Augmented large language models. *arXiv preprint arXiv:2310.16340*, 2023c. URL https://arxiv.org/abs/2310.16340.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2201.11903. arXiv:2201.11903.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. BrowseComp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025. URL https://arxiv.org/abs/2504.12516.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Gen LLM applications via Multi-Agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. URL https://arxiv.org/abs/2308.08155.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024. URL https://arxiv.org/abs/2407.01489.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2405.15793. arXiv:2405.15793.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable Real-World web interaction with grounded language agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2207.01206. arXiv:2207.01206.

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*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for Tool-Agent-User interaction in Real-World domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Asaf Yehudai, Lilach Eden, and Michal Shmueli-Scheuer. Agentic CLEAR: Automating multi-level evaluation of LLM agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2026. URL https://aclanthology.org/2026.acl-demo.74/. arXiv:2605.22608.

Yifan Yu, Moyan Li, Shaoyuan Xu, Jinmiao Fu, Xinhai Hou, Fan Lai, and Bryan Wang. CORRECT: Condensed error recognition via knowledge transfer in multi-agent systems. *arXiv preprint arXiv:2509.24088*, 2025. doi: 10.48550/arXiv.2509.24088. URL https://arxiv.org/abs/2509.24088.

Z.ai. GLM-5.3-Flash: Overview, 2026. URL https://docs.z.ai/guides/vlm/glm-5.3-flash.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? *arXiv preprint arXiv:2509.03312*, 2025a. doi: 10.48550/arXiv.2509.03312. 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 (ICML)*, 2025b. URL https://arxiv.org/abs/2505.00212. arXiv:2505.00212.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. *arXiv preprint arXiv:2404.05427*, 2024. URL https://arxiv.org/abs/2404.05427.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. URL https://arxiv.org/abs/2308.10144. arXiv:2308.10144.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2306.05685. arXiv:2306.05685.

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 (ICLR)*, 2024. URL https://arxiv.org/abs/2307.13854. arXiv:2307.13854.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where LLM agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025. doi: 10.48550/arXiv.2509.25370. URL https://arxiv.org/abs/2509.25370.

Kunlun Zhu, Xuyan Ye, Zhiguang Han, Yuchen Zhao, Bingxuan Li, Weijia Zhang, Muxin Tian, Xiangru Tang, Pan Lu, James Zou, Jiaxuan You, and Heng Ji. AgentDebugX: An open-source toolkit for failure observability, attribution, and recovery in LLM agents. *arXiv preprint arXiv:2607.18754*, 2026. URL https://arxiv.org/abs/2607.18754.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-Judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024. URL https://arxiv.org/abs/2410.10934.

A Formal Problem Definitions

This appendix completes the definitions of Section 2: the trace and span-identity objects, the taxonomy contracts, the analysis operator, the clustering partition and representative contract, and the hypothesis, verdict, and collection-level explanation objects that Section 2 introduces. Diagnosis is not exhausted by labeling each trace: it must preserve where an issue occurred, decide which manifestations are jointly actionable, and ground every explanation in the recorded executions.

A.1 Traces and Span Identity

Trace i is represented as

$$T_i = (u_i, d_i, x_i, y_i, S_i, \mathcal{R}_i), \qquad S_i = (s_{i1}, \ldots, s_{in_i}), \tag{A.1}$$

where u_i is the source trace identifier, d_i identifies the data source, x_i and y_i are an optional task input and final output, and S_i is the timestamp-ordered span sequence. The parent identifiers of these spans induce the span forest $\mathcal{R}_i$. A span s_{ij} contains a verbatim source identifier ℓ_{ij}, an optional parent identifier, a canonical kind, timing and status fields, source attributes, and normalized message or tool payloads.

For a trace T_i, let

$$\mathrm{ID}(T_i) = \{\ell(s) : s \in S_i\} \tag{A.2}$$

be its set of source span identifiers. A canonical index is useful for display and filesystem ordering, but it is not a replacement for an element of $\mathrm{ID}(T_i)$. Every localized output must satisfy the admissibility constraint

$$\forall I \in \widehat{\mathcal{I}}_i, \qquad \mathrm{loc}(I) \in \mathrm{ID}(T_i), \tag{A.3}$$

whose membership test is implemented by exact string equality. Providers use leading zeros, mixed case, or prefixes in their span identifiers, so re-numbering leaves a trace visually unchanged while silently invalidating its predicted location; **TraceACE** therefore keeps the one-based display index and the verbatim source identifier as distinct fields through adaptation, compaction, chunking, prediction, and scoring.

A.2 Taxonomies and Scanner Coverage

The admissible categories depend on the deployment or benchmark. That vocabulary is represented as the leaves of a rooted taxonomy

$$\mathfrak{Y}_d = (V_d, E_d), \qquad \mathcal{Y}_d = \mathrm{Leaf}(\mathfrak{Y}_d), \tag{A.4}$$

where d identifies the workflow. A workflow may distribute these leaves among a set of scanners Ω_d. If $\mathcal{Y}_{d,\omega}$ is the set owned by scanner ω, the coverage contract is

$$\mathcal{Y}_d = \bigcup_{\omega \in \Omega_d} \mathcal{Y}_{d,\omega}. \tag{A.5}$$

Equation A.5 permits overlap when a workflow deliberately asks more than one scanner to inspect the same category. The TRAIL configuration used in this paper is a disjoint specialization:

$$\mathcal{Y}_{\text{TRAIL}} = \biguplus_{\omega \in \Omega_{\text{TRAIL}}} \mathcal{Y}_{\text{TRAIL},\omega}, \qquad |\mathcal{Y}_{\text{TRAIL}}| = 21, \qquad |\Omega_{\text{TRAIL}}| = 6. \tag{A.6}$$

TELBench supplies a separate typed-error vocabulary, which is not a relabeling of the TRAIL leaves and is scanned through an overlapping cover rather than a disjoint partition (Equation B.3):

$$\mathcal{Y}_{\text{TELBENCH}} \neq \mathcal{Y}_{\text{TRAIL}}, \qquad |\mathcal{Y}_{\text{TELBENCH}}| = 18. \tag{A.7}$$

AgentErrorBench's module/type vocabulary $\mathcal{Y}_{\text{AEB}}$ is, like TRAIL's, a disjoint partition with one scanner per module (Figure B.2). Equation A.7 concerns admissible label strings only; trace distributions, prompts, context policies, and post-processing are configured per workflow. TELBench's official task is span localization; treating its 18 released error types as prediction targets is a **TraceACE** extension evaluated separately. These equations constrain the output space; they do not assume that every observable failure has exactly one cause or that a scanner will detect every admissible category.

A.3 Issue Records and the Analysis Operator

Analysis maps one trace to a finite set of evidence-bearing issue records. The semantic content of an issue is

$$I^{\text{sem}} = (c, \ell, q, e, h), \tag{A.8}$$

where c is an issue category, ℓ is the exact source span location, q is a short summary, e records the supporting observation, and $h \in \{\text{LOW}, \text{MEDIUM}, \text{HIGH}\}$ denotes impact. Scanner identity, analysis channel, and lifecycle status are provenance fields rather than part of the semantic claim; Section 3 and Appendix B describe how they are persisted for auditing.

Taxonomy-constrained localized issue extraction. For a target vocabulary $\mathcal{Y}$, the analysis operator returns

$$\mathsf{A}(T_i; \mathcal{Y}) = \widehat{\mathcal{I}}_i = \left\{ I_{ij}^{\text{sem}} = (c_{ij}, \ell_{ij}, q_{ij}, e_{ij}, h_{ij}) \right\}_{j=1}^{m_i}. \tag{A.9}$$

Each predicted issue must obey

$$c_{ij} \in \mathcal{Y}, \qquad \ell_{ij} \in \text{ID}(T_i), \qquad \text{Supported}(e_{ij}; T_i, \ell_{ij}) = 1. \tag{A.10}$$

Here Supported is the task-level requirement that the stated evidence be recoverable from the observable trace at the cited location. It excludes a category or narrative that has no corresponding recorded event.

The collection-level issue set is $\widehat{\mathcal{I}} = \bigcup_{i=1}^{N} \widehat{\mathcal{I}}_i$. Multiple issues may legitimately share one span because one event can violate more than one category. The analysis contract therefore does not collapse predictions solely by location. It also does not expose annotations to the inference operator: ground truth is an input to evaluation, not to A.

A.4 Same-Fix Partition and Representative Contract

Topical similarity is insufficient for collection-level diagnosis. Two timeout messages may originate from different mechanisms and require different interventions; conversely, surface-different errors may both be removed by correcting one malformed tool schema. For an issue I, $\mathcal{F}_{\text{dir}}(I)$ denotes the conceptual set of concrete interventions that directly address its evidenced mechanism, and the operational same-fix target relation

$$I_a \sim_f I_b \quad \Longleftrightarrow \quad \mathcal{F}_{\text{dir}}(I_a) \cap \mathcal{F}_{\text{dir}}(I_b) \neq \varnothing \tag{A.11}$$

declares two issues related whenever these sets intersect. The clustering operator proposes a partition

$$\mathsf{C}(\widehat{\mathcal{I}}) = \widehat{\mathcal{C}} = \{C_1, \ldots, C_K\}, \qquad \bigcup_{k=1}^{K} C_k = \widehat{\mathcal{I}}, \qquad C_k \cap C_{k'} = \varnothing \ \ (k \neq k'). \tag{A.12}$$

For each cluster, a representative trace is defined abstractly as

$$T_k^{\star} = \underset{T \in \text{Traces}(C_k)}{\arg\max} \ U(T; C_k), \tag{A.13}$$

where U trades off diagnostic value and investigation cost. The concrete transparent score used by **TRACEACE** is given in Equation 2.

Equation A.11 defines an operational target, not a theorem that ambiguous evidence always induces a transitive equivalence relation. Consequently, $\widehat{\mathcal{C}}$ is a proposed jointly actionable partition whose validity must be checked rather than inferred from topical resemblance. Where human same-fix labels exist, the partition can be evaluated directly. Otherwise, only observable properties such as coverage and agreement under secondary-trace validation are reported; detector-derived signatures are not reused as synthetic clustering ground truth. The representative similarly initiates investigation but is not evidence that all cluster members share its mechanism.

A.5 Hypotheses, Verdicts, and the Collection-Level Explanation

For cluster C_k, investigation of its representative returns a root-cause hypothesis

$$H_k = (r_k, m_k, E_k, \gamma_k, f_k), \qquad E_k = \{(\ell_{kb}, q_{kb})\}_{b=1}^{B_k}, \tag{A.14}$$

where r_k names the proposed root cause, m_k describes the failure mechanism, E_k contains source-span and quotation pairs, $\gamma_k \in [0, 1]$ is self-reported confidence, and f_k is a fix type and sketch. The term *hypothesis* is deliberate: the grounding contract of Equation 1 supports an observational diagnosis, not a causal claim, and Section 6 discusses the limits of causal interpretation. An admitted representative-trace hypothesis requires $E_k \neq \varnothing$ and $g(H_k, T_k^\star) = 1$.

For each sampled secondary trace $T_j \in \mathcal{S}_k \subseteq \mathrm{Traces}(C_k) \setminus \{T_k^\star\}$, validation returns

$$V(H_k, T_j) \in \{\textsc{confirmed}, \textsc{partial}, \textsc{refuted}\}, \tag{A.15}$$

and, when $\mathcal{S}_k$ is non-empty, the confirmed cross-trace agreement $\mathrm{Agree}(H_k, C_k)$ of Equation 3 is the fraction of sampled secondary traces whose verdict is CONFIRMED.

A validation record also retains the observed mechanism, cited evidence, and confidence for each secondary trace. Partial or refuted verdicts are therefore not averaged away: they can trigger additional sampling and mechanism-based re-diagnosis. The final collection-level explanation is the auditable object

$$\widehat{\mathcal{H}} = \{(C_k, T_k^\star, H_k, \{(T_j, V(H_k, T_j)) : T_j \in \mathcal{S}_k\})\}_{k=1}^{K}, \tag{A.16}$$

which connects each proposed cause and fix to the cluster it explains, the trace used for discovery, and the independent traces used to support, qualify, or falsify it.

B Method Details

This appendix expands Section 3. Figure 1 summarizes that flow and the one-way boundary that keeps annotations available to evaluation but invisible to inference. The single-trace analysis and filesystem investigation rest on three mechanisms: compaction, a navigable workspace, and taxonomy-conditioned failure analysis. **TraceACE** carries that pattern from inspecting one trace to diagnosing a collection: it introduces interpretable same-fix signatures, representative selection, evidence-constrained cluster hypotheses, secondary-trace validation, and disagreement-driven re-diagnosis. These requirements couple semantic diagnosis with evidence integrity: an attractive root-cause narrative is insufficient if its quotation cannot be recovered or its mechanism fails on secondary traces. Section 3 and this appendix follow that methodological progression rather than the implementation's command-line stages.

The first four subsections mirror the four subsections of Section 3 and collect the packing predicate, the taxonomy-cover discussion, the dual-channel candidate union, the merge-only constraint, the adaptive validation sample, and the stage figures; the remaining subsections record the canonical-trace invariants, the scanner ownership sets, the general form of the representative score, the evidence gate, and the inference settings used for the reported runs. Sentences trimmed from Section 3 for the page limit (the canonical-trace field inventory and leak-prevention rules, the compaction caveat, the workspace layout, the segmentation caveat, the scanner contract, the partition-versus-cover argument, the embedding-free signature rationale, the representative-is-not-proof caveat, the Root-Cause Analyzer tool inventory and budget, the strict refuted branch, and the re-diagnosis outcomes) are kept here under the matching subsection.

B.1 Preparing Long Traces: Chunk Packing, Truncation Rescue, and Triage

Adapter boundary. Figure B.1 summarizes the preparation stage. Source-specific structure is not discarded wholesale: the adapter retains attributes that are useful for diagnosis, and the canonical span forest records nesting even when a wrapper is not materialized as a separate workspace file. The adapter boundary is deliberately narrower than a claim of universal compatibility. A new telemetry source must implement and test its mapping; merely using a span-shaped JSON does not make two sources semantically equivalent. OpenTelemetry defines the general trace/span model (OpenTelemetry Authors, 2026), while OpenInference specifies common AI attributes (OpenInference Authors, 2026); **TraceACE** consumes useful subsets through tested adapters rather than claiming exhaustive conformance. The compaction rules of Section 3 reduce representation cost for observed formats, but they are not lossless for every possible telemetry consumer: binary bytes are deliberately removed, so a downstream application that needs to inspect image pixels would require a different policy. For the text-and-tool debugging contract, the transformation keeps the surrounding description, source identity, and tree position visible.

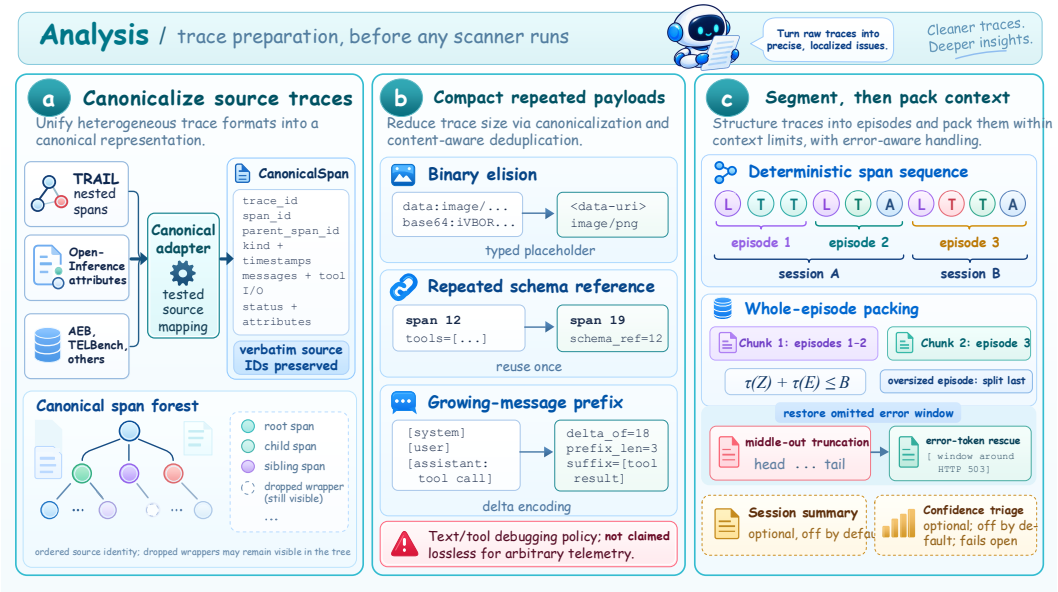


Figure B.1: Trace preparation preserves identity while making long trajectories inspectable. Source-specific adapters emit a canonical span tree, repeated payload structure is compacted, and deterministic episode/session boundaries guide chunk packing. Optional summaries and triage operate only after segmentation and fail open when confidence is insufficient.

Episode segmentation and chunk packing. Boundaries come from span kinds, ancestry, and agent scope; no model is called. A session is a contiguous run of episodes in the same scope, and re-entering a scope later starts a new session rather than merging distant activity. Segmentation is total: every non-dropped span is assigned once and the source order is preserved. The rule is deterministic, not a claim about the uniquely correct semantic boundary. The episode-aware chunker packs whole episodes until adding the next unit would exceed a token or span-count budget. It prefers to flush at session edges after a configurable fraction of the budget has been used. An episode that is individually too large may be split as a last resort. Unlike flat chunking (the *Flat chunks* ablation of Table 2, consecutive spans with a one-span overlap), semantic boundaries make cross-chunk span overlap unnecessary. Each chunk records its session membership, which later enables optional triage.

Episode boundary. Writing $S_i^+ = (s_1, \ldots, s_m)$ for the non-dropped spans in source order, $a(s)$ for the agent scope, $\kappa(s)$ for the span kind, and E_{j^-} for the episode open when s_j arrives, the implemented boundary indicator is

$$b_1 = 1, \qquad b_j = \mathbf{1}\big[a(s_j) \neq a(E_{j^-}) \ \vee \ \big(\kappa(s_j) = \text{LLM} \ \wedge \ n_{\text{LLM}}(E_{j^-}) \geq 1\big)\big] \quad (j > 1), \tag{B.1}$$

a deterministic segmentation rule, not a claim about the uniquely correct semantic boundary.

Chunk packing. For scanner ω, let Z be the current non-empty chunk, E the next episode, $\tau_\omega(\cdot)$ the rendered-token estimate, B the token budget, and M_ω the span-count limit. The current packer flushes Z before E exactly when

$$\begin{aligned} \text{Flush}_\omega(Z, E) &= \mathbf{1}[Z \neq \varnothing \ \wedge \ \Phi_\omega(Z, E)]\,, \\ \Phi_\omega(Z, E) &:= \big(\tau_\omega(Z) + \tau_\omega(E) > B\big) \ \vee \ \big(|Z| + |E| > M_\omega\big) \\ &\quad \vee \ \big(\text{NewSession}(E) \wedge \tau_\omega(Z) \geq 0.6B\big). \end{aligned} \tag{B.2}$$

If $\tau_\omega(E) > B$, the episode is greedily split at span boundaries as a last resort; an indivisible oversized span may still exceed the estimate. The rule in Equation B.2 records the actual packing policy and does not assert globally optimal context allocation.

Truncation rescue. Two safeguards address long fields. Middle-out truncation keeps the beginning and end of oversized payloads. Because a content-blind cut can hide an error token in the middle,

the chunker searches the original field for error indicators and reattaches small windows around indicators omitted from the rendered block. This *truncation rescue* is deterministic and does not create a predicted category. A workflow can also set a smaller per-scanner budget than the global ceiling when a very large but technically admissible chunk would dilute attention.

Session summaries and triage. Session summaries and confidence-aware triage are optional. When enabled, one cached summary per session provides a map of the full trace to chunks that otherwise see only a local window. A cheap dispatcher then selects scanner/session pairs. Triage fails open: an absent entry, an unparseable response, a confidence below the configured threshold, or a chunk without session metadata causes the scanner to run. In the default configuration used for the reported runs, episode chunking is enabled, whereas session summaries and triage are disabled. Their quality/cost contribution is left to a future ablation rather than asserted here.

B.2 Analysis: Taxonomy Covers, Anchors, and the Structural Channel

Partition versus cover. **TRACEACE** treats high-recall failure analysis as a configuration-driven workflow, not a free-form agent. Figure B.2 expands the dataset contracts in Equation A.7. TRAIL's 21 verbatim leaves are scanned by a six-way disjoint scanner partition; workflow-specific prompts, visible context, anchors, abstention guidance, and merge policy adapt scanning to a telemetry source and its annotation convention. The conceptual hierarchy and scanner partition are deliberately shown as different layers: *Resource Abuse*, for example, remains a System Execution leaf while being scanned by the Planning and Coordination scanner. AgentErrorBench's module/type labels are likewise partitioned, with one scanner per module.

TELBench instead defines 18 typed errors in six conceptual groups. Its current workflow uses two overlapping scanner views—one evidence-oriented and one process-oriented— and each sees the complete 18-label vocabulary:

$$\mathcal{Y}_{\text{TELBENCH, evidence}} = \mathcal{Y}_{\text{TELBENCH, process}} = \mathcal{Y}_{\text{TELBENCH}}. \tag{B.3}$$

This is an operational cover, not the six-way partition used for TRAIL. AgentErrorBench's 19 module/type labels are instead partitioned disjointly like TRAIL's, one scanner per module (Figure B.2).

The difference is forced by the label space, not by convenience. A disjoint partition is sound when the groups name mutually exclusive kinds of failure: a hallucination is not a rate limit, so a scanner that owns only its slice still recognises its own cases. TELBench's 18 labels instead read as competing descriptions of a single misstep—the same step can be argued as `unsupported_commitment`, as `source_verification_error`, or as `constraint_semantics_error` depending on which requirement one takes as violated. A scanner holding three of those labels has no admissible way to abstain in favour of a sibling it cannot see, so it forces every error it notices into its own slice and mislabels it. Measured on the XBench evaluation set with Gemini 3.7 Flash in the additional comparison, transplanting the six-way partition to TELBench yields joint accuracy between 0.0 and 2.6; the overlapping cover of Equation B.3 reaches 14.1 on the same set and 17.9 on BrowseComp. The partition is therefore kept where the taxonomy is disjoint (TRAIL, AgentErrorBench) and the cover where it is not, and both configurations are reported rather than presenting one detector architecture as universal.

Each scanner specification fixes its owned labels, model alias, prompt, relevant span kinds, chunk size, truncation budget, and contextual fields. A scanner cannot legitimately invent a label outside its ownership. Returned locations are checked against the span IDs supplied in that chunk. All workflows reuse the runner, parser, and artifact contract while keeping their taxonomy and scan configuration explicit.

Rule-based anchors. Some failures are already expressed by telemetry: a span is marked ERROR and its status or output matches a known signature. The anchor detector maps these rule-decidable events into the scored vocabulary, collapsing related ancestor errors toward the most specific event. Anchors share the scored analysis channel because their labels are valid for the scored contract. They complement rather than silently override model proposals, and the evaluator separately reports how often the model channel covered anchor locations. The anchor detector is enabled per workflow; no reported benchmark run enables it, so the anchor term of Equation B.4 is empty there.

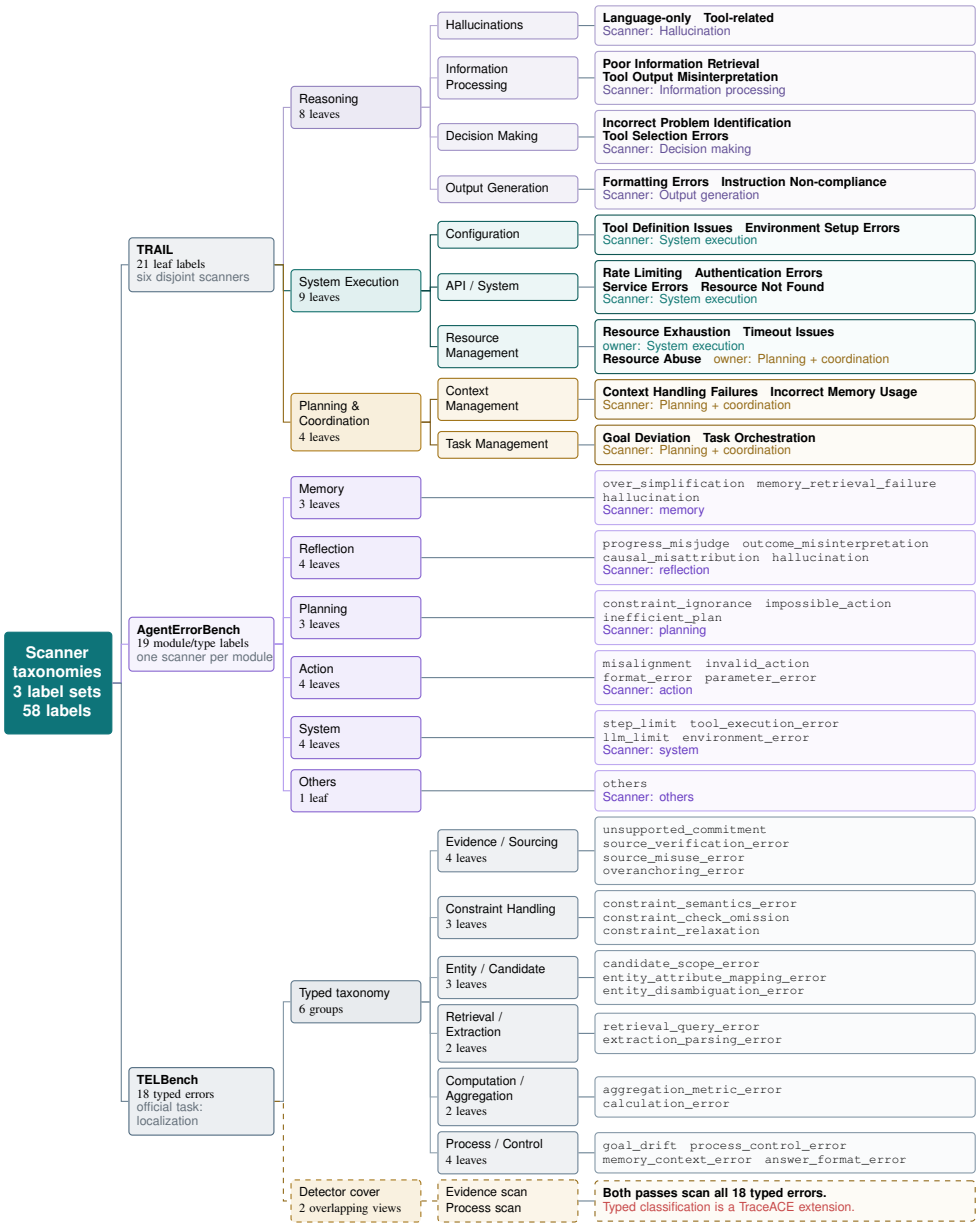


Figure B.2: Scanner taxonomies of the instantiated workflows (Analysis stage). TRAIL's 21-label vocabulary is scanned by six disjoint scanners, and AgentErrorBench's 19 module/type labels by one scanner per module (Table 1). TELBench, used in the additional comparison of Table D.2, supplies an 18-type vocabulary grouped into six conceptual families. Its official task is harmful-span localization; TraceACE additionally evaluates typed errors using two overlapping full-vocabulary passes. Taxonomy membership and detector ownership are distinct relations, as illustrated by *Resource Abuse*.

Dual-channel candidate union. Let $\mathcal{J}_i$ contain the scanner–chunk pairs actually dispatched for trace T_i after fail-open triage, and let A_ω denote scanner ω. Before any post-processing, the candidate issue multiset is

$$\mathcal{I}_i^{(0)} = \left(\biguplus_{(\omega,Z)\in\mathcal{J}_i} \mathsf{A}_\omega(Z;\mathcal{Y}_{d,\omega}) \right) \uplus \mathsf{A}_{\text{anchor}}(T_i). \tag{B.4}$$

The union makes the two detection routes explicit; anchor output does not suppress a model proposal before provenance-preserving post-processing.

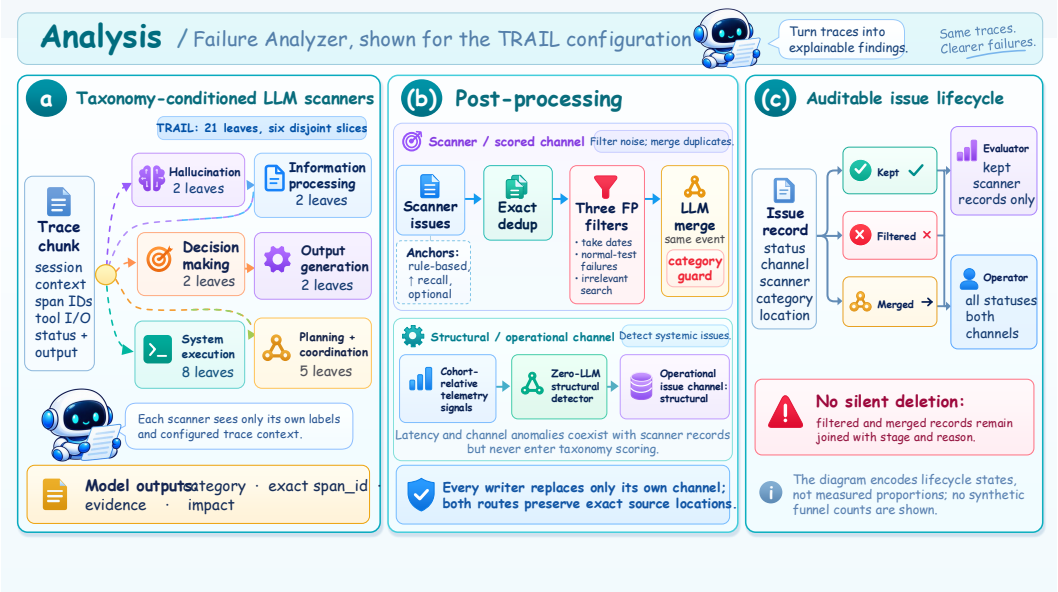


Figure B.3: The Failure Analyzer, shown for the TRAIL configuration. (a) Six taxonomy-conditioned scanners each own a disjoint slice of the 21 labels. (b) Scanner issues and optional rule-based anchors pass through exact deduplication, three false-positive filters, and a within-category LLM merge; a zero-LLM structural channel runs beside them. (c) The issue store keeps every status, but the evaluator admits only kept scored issues.

Structural side channel. A second detector computes zero-LLM, cohort-relative operational signals such as latency or chain anomalies. Structural issues use a different vocabulary and `channel=structural`. Both channels coexist in each trace's issue file; each writer replaces only its own channel. This allows an operator to combine semantic diagnosis and telemetry triage without contaminating benchmark labels.

False-positive filters and same-event merging. All proposals first undergo exact deduplication. Three model-assisted false-positive filters target recurrent confounders: fabricated dates that are part of reasoning, expected test failures encountered during debugging, and irrelevant search attempts that are normal exploration. A same-event merger can then merge redundant wording. Its `within_category` guard forbids folding different scored categories together, even when they occur at the same location. This matters because one span can correctly carry multiple TRAIL labels. Every removed or merged item remains in the file with its status and reason. With D exact deduplication, L the optional per-location collapse, $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3$ the configured filters, and M_{cat} the within-category merger, the kept view of the candidate multiset $\mathcal{I}_i^{(0)}$ is

$$\widehat{\mathcal{I}}_i = \mathrm{Keep}\Big[(\mathsf{M}_{\text{cat}} \circ \mathsf{F}_3 \circ \mathsf{F}_2 \circ \mathsf{F}_1 \circ \mathsf{L} \circ \mathsf{D})\left(\mathcal{I}_i^{(0)}\right)\Big], \tag{B.5}$$

where an inactive optional stage is the identity map; the composition records the execution order rather than claiming that the transformations commute, and Figure B.3 shows the resulting issue store.

B.3 Clustering: Seed Groups and Merge-Only Coarsening

Clustering begins with a signature rather than a dense embedding, so an engineer can see why two issues share a group and which component caused fragmentation. For an issue I on trace T_i, with ρ a deterministic error-token normalizer applied to its evidence and summary, the signature is

$$\sigma(I) = \Big(c(I),\ \kappa\big(T_i[\mathrm{loc}(I)]\big),\ \mathrm{tool}\big(T_i[\mathrm{loc}(I)]\big),\ \rho\big(e(I) \,\|\, q(I)\big)\Big), \tag{B.6}$$

and the seed groups and the initial partition are

$$G_z = \{I \in \widehat{\mathcal{I}} : \sigma(I) = z\}, \qquad \mathcal{C}^{(0)} = \{G_z : G_z \neq \varnothing\}. \tag{B.7}$$

One LLM call receives summaries of the initial groups and proposes merges that an engineer would likely address with one fix. The operation is conservative in two ways: it may merge groups but never split an exact group, and it runs once rather than iteratively rewriting membership. A model failure therefore leaves deterministic groups available rather than destroying the clustering stage. The output records cluster titles, rationales, categories, signatures, issue and trace counts, member traces, and sample issues.

If $\widehat{\mathcal{C}}$ is the partition after the merge pass, its merge-only constraint is

$$\forall G_z \in \mathcal{C}^{(0)}, \qquad \exists!\, C_k \in \widehat{\mathcal{C}} \quad \text{such that} \quad G_z \subseteq C_k. \tag{B.8}$$

Consequently, a model can coarsen the deterministic partition but cannot separate a seed group or duplicate it across output clusters.

B.4 Explanation: Adaptive Validation and Re-Diagnosis

Investigator budget. The agent is bounded by a maximum turn count and a context budget. Old tool results can be elided from the conversation while remaining in workspace files. If the ordinary loop exhausts its budget, a forced terminal call requests a best-effort structured answer and marks the abortion rather than silently dropping the cluster. The Cross-Trace Validator's task is narrower than discovery: determine whether the proposed mechanism is confirmed, partially present, or refuted, and cite the observed mechanism, under a smaller per-trace turn budget than the main Root-Cause Analyzer.

Adaptive validation sample. For a cluster with a non-empty hypothesis, let $\mathcal{M}_k = \mathrm{Traces}(C_k) \setminus \{T_k^\star\}$, let $\mathcal{S}_k^{(0)}$ be the initial sample of at most n_0 members, and let $n_{\max}$ be the validation cap. Define the disagreement indicator d_k and expansion size r_k by

$$d_k = \mathbf{1}\Big[\exists T \in \mathcal{S}_k^{(0)} : V(H_k, T) \neq \textsc{confirmed}\Big], \tag{B.9}$$

$$r_k = d_k \min\Big\{n_{\max} - |\mathcal{S}_k^{(0)}|, |\mathcal{M}_k \setminus \mathcal{S}_k^{(0)}|\Big\}. \tag{B.10}$$

The adaptive sample used by the implementation is then

$$\mathcal{S}_k = \mathcal{S}_k^{(0)} \cup \mathrm{Pick}\Big(\mathcal{M}_k \setminus \mathcal{S}_k^{(0)}, r_k\Big). \tag{B.11}$$

Here Pick is reproducible seeded sampling over the sorted eligible trace identifiers, rather than annotation-aware selection. The current defaults are $n_0 = 2$ and $n_{\max} = 5$. Singleton clusters and empty hypotheses are handled separately rather than being forced into this recurrence.

Escalation and re-diagnosis. If all sampled members support the hypothesis, the cluster receives a confirmed status and an agreement rate. A partial or refuted verdict expands sampling up to the cap. **TRACEACE** preserves each verdict rather than replacing disagreement with a majority summary. When the final status is mixed or refuted and re-diagnosis is enabled, another step groups members by their observed mechanisms. The result can be a revised shared explanation, multiple mechanism groups, or unassigned traces. This is an internal validity procedure, not a substitute for human same-fix labels; every verdict and revised group is persisted in the cluster record, and the aggregate status follows Equation 3. Its second branch is stricter than a refuted majority: any confirmed member prevents the cluster from being summarized as refuted. Figure B.4 traces the path from recurring symptoms to a validated cluster hypothesis.

The resulting cluster record preserves the representative evidence, every secondary verdict, the final validation status, and any revised mechanism groups. These records feed deterministic engineering reports and regression candidates. Their evaluation is defined in Section 4; the structured prompt/output contracts appear in Appendix F.

B.5 Canonical Trace Representation

An input trace is represented as a rooted span forest $\mathcal{T} = (V, E, x, y)$, where V is the set of spans, E contains parent–child relations, $x(v)$ is the observable content and telemetry of span v, and y

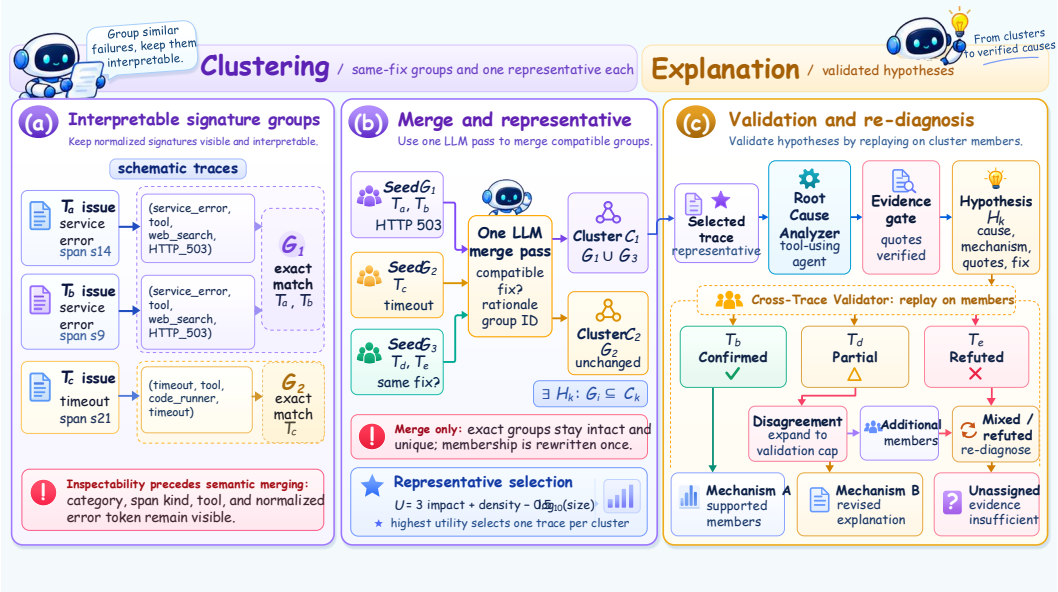


Figure B.4: From recurring symptoms to a validated cluster hypothesis. (a) Exact signatures form seed groups. (b) One merge-only LLM pass coarsens them into clusters, and a transparent utility selects one representative each. (c) The Root-Cause Analyzer investigates the representative behind an evidence gate; the Cross-Trace Validator replays the hypothesis on sampled members, and disagreement expands the sample and may trigger re-diagnosis into revised mechanism groups.

is an optional final response. Each span retains its source `span_id`, parent identifier, temporal order, kind, status, duration, attributes, and normalized model or tool messages. Concretely, source adapters emit a single `CanonicalTrace` object whose spans carry verbatim source and parent identity, timestamp order, a broad kind (LLM, tool, retriever, chain, agent, or other), timing and status, attributes, messages, and normalized tool fields. The source identifier is never renumbered: localization is evaluated by exact string equality, so a semantically equivalent replacement would still be an incorrect prediction. Two leak-prevention rules apply at adaptation time: ground truth is stored under `traces/<id>/gt/`, outside the inference workspace, and label-bearing dataset metadata is never serialized into prompts; accordingly, compaction and workspace construction accept no ground-truth argument.

Raw agent traces often repeat more bytes than semantic events: agent frameworks may embed images as data URIs, resend the same tool definitions at each model call, and prepend an ever-growing conversation history to consecutive calls. Trace compaction therefore applies four label-independent transformations after canonicalization. Inline binary data are replaced by typed placeholders; repeated tool schemas are referenced once; empty wrapper spans remain in the tree but need not receive a separate content record; and repeated message prefixes are represented by a pointer to an earlier model span plus the elided-prefix length (consecutive LLM calls in the same agent scope retain only the new message suffix after a shared prefix of at least two messages, recording `delta_of` and `delta_prefix_len`). Let g denote the resulting transform. The required invariants are

$$\mathrm{ID}(g(\mathcal{T})) = \mathrm{ID}(\mathcal{T}), \qquad E(g(\mathcal{T})) \simeq E(\mathcal{T}), \qquad g \perp \mathcal{Y}_{\mathrm{eval}}, \tag{B.12}$$

where the second relation preserves ancestry after explicitly marked empty-wrapper elision and the third states that evaluation annotations are not inputs to compaction. The compact trace is rendered as an overview, a span tree, and individually addressable span records: a Markdown overview (`overview.md`), a text span tree (`span_tree.txt`), and one numbered JSON file per non-dropped span, each still carrying the exact source ID. An Root-Cause Analyzer can first read the overview and tree, search for a tool or error token, and open only the spans necessary to support a conclusion; the representation thus prevents a long trace from being injected as one undifferentiated prompt, and the Root-Cause Analyzer's tool layer resolves paths underneath the workspace root and cannot reach the sibling ground-truth directory.

Table B.1: Six Scanners and their 21 exact TRAIL leaf categories.

Scanner	Owned categories
Hallucination	Language-only; Tool-related
Information processing	Poor Information Retrieval; Tool Output Misinterpretation
Decision making	Incorrect Problem Identification; Tool Selection Errors
Output generation	Formatting Errors; Instruction Non-compliance
System execution	Tool Definition Issues; Environment Setup Errors; Rate Limiting; Authentication Errors; Service Errors; Resource Not Found; Resource Exhaustion; Timeout Issues
Planning and coordination	Context Handling Failures; Incorrect Memory Usage; Resource Abuse; Goal Deviation; Task Orchestration

B.6 Episode-Aware Analysis Operator and Scanner Ownership

Deterministic segmentation introduces episode boundaries around model decisions and agent-scope changes while keeping triggered actions with the decision that produced them whenever the tree permits; the session and packing rules are those of Equation B.2.

For scanner ω and packed chunk Z_j, let $Q_\omega(B_j; \mathcal{K}_\omega)$ be the issues emitted under the exact category set $\mathcal{K}_\omega$. The analysis stage first forms

$$\mathcal{I}_0(\mathcal{T}) = \bigcup_{\omega \in \Omega_d} \bigcup_{j} Q_\omega(Z_j; \mathcal{K}_\omega) \cup \mathcal{A}(\mathcal{T}), \qquad \widehat{\mathcal{I}} = \mathsf{Merge}_{\mathrm{cat}}(\mathsf{Filter}(\mathsf{Dedup}(\mathcal{I}_0))), \tag{B.13}$$

where $\mathcal{A}$ denotes rule-based error anchors (empty in every reported run). In the order of Equation B.5, exact deduplication removes identical proposals, filters remove declared false-positive patterns, and the within-category merger $\mathsf{Merge}_{\mathrm{cat}}$ merges repeated descriptions of one event, permitting a fold only within the same scored category.

Table B.1 lists the six scanner ownership sets. The strings are scoring contracts; apparent stylistic inconsistencies are retained because the TRAIL scorer compares category names exactly.

B.7 Conservative Same-Fix Clustering: General Form of the Representative Score

The merge criterion applied to the seed groups of Equation B.7 is stricter than topical similarity, and uncertainty leaves groups separate: repeated symptoms from different components remain distinct, while different surface errors from one broken component may merge when the mechanism and owner coincide.

Representative selection is transparent. For trace T in cluster C_k, the score combines issue coverage, evidence richness, structural specificity, and an inspection cost penalty,

$$\begin{aligned} U(T; C_k) &= \lambda_1 \ \mathrm{coverage}(T, C_k) + \lambda_2 \ \mathrm{evidence}(T, C_k) \\ &\quad + \lambda_3 \ \mathrm{specificity}(T, C_k) - \lambda_4 \ \mathrm{cost}(T), \\ T_k^\star &= \operatorname*{arg\,max}_{T \in \mathrm{Traces}(C_k)} U(T; C_k). \end{aligned} \tag{B.14}$$

Equation 2 is the concrete instantiation used in the reported runs: impact_k as coverage ($\lambda_1 = 3$), $\mathrm{density}_k$ as evidence ($\lambda_2 = 1$), no specificity term ($\lambda_3 = 0$), and $\log_{10} \max\{10, \mathrm{size}\}$ as cost ($\lambda_4 = 0.5$). The representative supplies the initial hypothesis; it is not treated as privileged ground truth, and the remaining members become validation candidates, since the representative is never proof that all members share its mechanism.

B.8 Evidence-Constrained Explanation and the Evidence Gate

The Root-Cause Analyzer receives the cluster description and one representative trace environment. Its actions are limited to listing, searching, paginated reading, batch reading, whole-trace orientation, and span filtering: deterministic tools list, read, and search workspace files, and two cached model-powered tools summarize large chunks or filter the trace by a question. Native function calling is used

Table B.2: Inference and validation settings used in the reported experiments.

Component	Setting	Reported value
Scanner packing	global token cap	30,000 estimated tokens
Span rendering	field cap	4,000 characters; scanner-overridable
Flat fallback	chunk overlap	one span
Segmentation	packing policy	episode-aware; summaries and triage disabled
Representative RCA	investigation budget	30 turns; 100,000-token context; 400 lines/read
Secondary validation	sample and budget	two to five members; eight turns/member
Evidence gate	correction budget	at most two retries
Model response	output ceiling	32,000 tokens, including shared hidden-reasoning limits

when the endpoint supports tools, with a structured ReAct-compatible fallback otherwise (Yao et al., 2023). It must distinguish agent-application defects, model behavior, and environment or service faults. The terminal output contains a root cause, a mechanism, two to five `span_id`–quotation pairs, confidence, a fix type, and a concrete fix sketch.

Every evidence item is resolved against the cited span before the report is accepted. For a hypothesis H with evidence pairs $\mathcal{E}(H) = \{(s_j, q_j)\}_{j=1}^{J}$, the gate evaluates

$$\Gamma(H) = \prod_{j=1}^{J} \mathbf{1}[s_j \in \mathrm{ID}(W)]\, \mathbf{1}[\mathrm{match}(q_j, W_{s_j})], \tag{B.15}$$

where W is the Root-Cause Analyzer-visible workspace, W_{s_j} is its serialization of span s_j, and match performs a raw-or-decoded exact substring check. An absent span or unrecoverable quotation ($\Gamma(H) = 0$) rejects the submission and returns a structured correction request, with at most two retries. This gate does not inspect reference annotations and does not grade causal correctness; an independent evaluator later measures validity, verbatim match, broader grounding, reference-span hits, and cause verdicts.

B.9 Inference Settings

Table B.2 records the settings used for the reported runs. They are implementation choices held fixed across the relevant comparisons, not universally optimal hyperparameters.

C Experimental Details and Protocols

Scope of this appendix. This appendix gives the scoring definitions and protocols shared by all experiments, and the protocols of an additional TRAIL/TELBench comparison with three frontier backbones and nine methods (Table D.2). The additional comparison uses a different method line-up and implementation from Section 4; its numbers are reported only in Appendix D and are not pooled with Tables 1–3.

C.1 Scoring Definitions

Section 4 scores TRAIL by category F1 weighted by reference support, exact localization (the macro-averaged recall of gold span identifiers under exact string equality), and joint accuracy, which credits a prediction only when both the category and the location are correct. AEB is scored by the benchmark's critical-step accuracy (Step), step-and-module accuracy (Step+M), and joint step, module, and type accuracy (All), with All kept under the original type-label mask. In the additional explanation diagnostics on the held-out TRAIL-GAIA cohort (Appendix E), root-cause explanation is additionally evaluated through the rates of span-valid, verbatim, and grounded evidence items, hits on any annotated error span and on the annotated first error, a full/partial/none cause judge, the refutation rate on fabricated hypotheses, and the review reduction from per-trace to cluster-level investigation. In all experiments, cost is measured as prompt-plus-completion tokens and model calls per prepared trace, compared only on settings in which every method has a positive usage record only in the additional comparison; Table D.1 covers complete cohorts. The formal definitions follow.

Failure-type evaluation. For trace i, let $\mathcal{G}_i$ and $\widehat{\mathcal{G}}_i$ be the reference and predicted sets of exact location–category pairs, and let $\mathcal{C}_i$ and $\widehat{\mathcal{C}}_i$ be their category projections:

$$\mathcal{G}_i, \widehat{\mathcal{G}}_i \subseteq \mathrm{ID}(T_i) \times \mathcal{Y}_d, \quad \mathcal{C}_i = \pi_c(\mathcal{G}_i), \qquad \widehat{\mathcal{C}}_i = \pi_c(\widehat{\mathcal{G}}_i),$$
$$y_{ic} = \mathbf{1}[c \in \mathcal{C}_i], \qquad \widehat{y}_{ic} = \mathbf{1}[c \in \widehat{\mathcal{C}}_i]. \tag{C.1}$$

Set semantics collapse duplicate predictions. For category c, the evaluator computes

$$\mathrm{TP}_c = \sum_i y_{ic}\widehat{y}_{ic}, \qquad \mathrm{FP}_c = \sum_i (1 - y_{ic})\widehat{y}_{ic}, \quad \mathrm{FN}_c = \sum_i y_{ic}(1 - \widehat{y}_{ic}),$$
$$P_c = \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c}, \qquad R_c = \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FN}_c}, \qquad F1_c = \frac{2P_cR_c}{P_c + R_c}, \tag{C.2}$$

with zero-denominator cases set to zero, as in the released scorer. Let $n_c = \sum_i y_{ic}$ be reference support. The reported failure-type score is

$$F1_{\mathrm{w}} = \frac{\sum_{c \in \mathcal{Y}_d} n_c F1_c}{\sum_{c \in \mathcal{Y}_d} n_c}. \tag{C.3}$$

Exact localization and joint evaluation. Let $\mathcal{L}_i = \pi_\ell(\mathcal{G}_i)$ and $\widehat{\mathcal{L}}_i = \pi_\ell(\widehat{\mathcal{G}}_i)$. Matching uses exact string equality of source `span_id` values. Localization and joint recovery are macro-averaged reference-set coverage:

$$a_i^{\mathrm{loc}} = \begin{cases} |\mathcal{L}_i \cap \widehat{\mathcal{L}}_i|/|\mathcal{L}_i|, & |\mathcal{L}_i| > 0, \\ 0, & |\mathcal{L}_i| = 0, \end{cases} \qquad \mathrm{Loc} = \frac{1}{N}\sum_{i=1}^{N} a_i^{\mathrm{loc}},$$
$$a_i^{\mathrm{joint}} = \begin{cases} |\mathcal{G}_i \cap \widehat{\mathcal{G}}_i|/|\mathcal{G}_i|, & |\mathcal{G}_i| > 0, \\ 0, & |\mathcal{G}_i| = 0, \end{cases} \qquad \mathrm{Joint} = \frac{1}{N}\sum_{i=1}^{N} a_i^{\mathrm{joint}}. \tag{C.4}$$

The tables report $100F1_{\mathrm{w}}$, $100\,\mathrm{Loc}$, and $100\,\mathrm{Joint}$. TELBench localization retains its official scorer; typed F1 and joint recovery use the same set construction over its released 18-type vocabulary only as extensions.

Critical root-cause attribution. For AEB trajectory i with annotated critical step, module, and type (s_i, m_i, t_i) and prediction $(\hat{s}_i, \hat{m}_i, \hat{t}_i)$, Step= $\frac{1}{N}\sum_i \mathbf{1}[\hat{s}_i = s_i]$, Step+M= $\frac{1}{N}\sum_i \mathbf{1}[\hat{s}_i = s_i \wedge \hat{m}_i = m_i]$, and All additionally requires $\hat{t}_i = t_i$ and averages only over the type-labelled trajectories (74 ALFWorld, 49 WebShop). In Table 3, Reports counts primary root-cause investigations, Cov. is the percentage of traces with a non-empty final root-cause output, Ground. is the Grounded rate of Equation C.5 over evidence items in primary reports, and Tok. applies Equation C.8 to all post-analysis stages.

Root-cause explanation evaluation. Root-cause analysis is evaluated separately from failure detection. For cited evidence items $\mathcal{E}$, let $v(e)$, $q(e)$, and $g(e)$ indicate a resolvable span, an exact quotation, and evidence grounded in that span. The deterministic evidence layer reports

$$\mathrm{Valid} = \frac{1}{|\mathcal{E}|}\sum_{e \in \mathcal{E}} v(e), \quad \mathrm{Verbatim} = \frac{1}{|\mathcal{E}|}\sum_{e \in \mathcal{E}} q(e), \quad \mathrm{Grounded} = \frac{1}{|\mathcal{E}|}\sum_{e \in \mathcal{E}} g(e). \tag{C.5}$$

Reference-aware evaluation additionally reports any-error-span hit, first-error hit, and a separate full/partial/no-support cause verdict. Cross-trace validation reports confirmed, partial, refuted, and singleton outcomes without pooling them into the reference score. Let $(\widetilde{H}_j, T_j)$, $j = 1, \ldots, M$, enumerate the secondary-trace verdicts returned on deliberately fabricated hypotheses, each replayed on the sampled members of its cluster; the negative-control rejection rate over these M verdicts is

$$\mathrm{RefuteRate} = \frac{1}{M}\sum_{j=1}^{M} \mathbf{1}[V(\widetilde{H}_j, T_j) = \textsc{refuted}]. \tag{C.6}$$

If per-trace analysis would produce N primary reports and clustered analysis produces K, human-facing review reduction is

$$\mathrm{ReviewReduction} = 1 - \frac{K}{N}. \tag{C.7}$$

This last quantity measures reports to review, not token or compute savings. The complete four-layer protocol is specified later in this appendix.

Efficiency and run admission. For run r with N_r prepared traces and Q_r recorded model calls, prompt and completion tokens p_{rq} and o_{rq} yield

$$\mathrm{Tok/Trace}_r = \frac{\sum_{q=1}^{Q_r}(p_{rq} + o_{rq})}{N_r}, \qquad \mathrm{Calls/Trace}_r = \frac{Q_r}{N_r}. \tag{C.8}$$

In the additional comparison, a model–dataset setting enters a matched cost comparison only when all nine methods have positive token and call records; the cost table of Appendix D.1 covers complete cohorts under one backbone, so no matched-subset rule applies there. A result cell is admitted only when the expected trace count and required metrics are complete and every available scan-health record contains zero failed scanner calls. Ground truth remains outside all method workspaces until predictions or explanations have been fixed for scoring.

C.2 Evaluation Sets and Reported Scopes

Each public evaluation set is scored on its complete trace collection (Table C.1), and within each model–dataset setting all methods share the same deterministic manifest, which makes the method comparisons paired. The 30-trace TRAIL-GAIA cohort used in the additional explanation diagnostics of Appendix E is a deterministic holdout that is separate from the detection tables and is not pooled into their averages. The TRAIL evaluation uses the benchmark's 116 accepted GAIA traces and 31 SWE traces. TELBench GAIA is omitted because its task source substantially overlaps the TRAIL GAIA evaluation.

Table C.1: Dataset statistics and evaluation scopes. Tables 1–3 use TRAIL and AEB; TELBench appears only in the additional comparison of Table D.2. The same complete trace set is shared by all methods in the corresponding model–dataset setting.

Evaluation set	Source size	Reported scope	Reference labels and endpoints
TRAIL GAIA (Deshpande et al., 2025; Mialon et al., 2024)	116 accepted traces	All 116 traces; eight backbones in Table 1, DeepSeek V4 Flash in Tables 2 and D.1, and three frontier backbones in Table D.2	21-leaf failure type and exact `span_id`; weighted F1, localization, and joint recovery
TRAIL SWE (Deshpande et al., 2025; Jimenez et al., 2024)	31 traces	All 31 traces; eight backbones in Table 1, DeepSeek V4 Flash in Table 2, and three frontier backbones in Table D.2	21-leaf failure type and exact `span_id`; weighted F1, localization, and joint recovery
AEB ALFWorld (Zhu et al., 2025)	100 trajectories	All 100 (All over the 74 typed traces); eight backbones in Table 1, DeepSeek V4 Flash in Tables 2, 3 and D.1	Critical step, module, and type; Step, Step+M, and All
AEB WebShop (Zhu et al., 2025)	50 trajectories	All 50 (All over the 49 typed traces); same backbones as ALFWorld	Critical step, module, and type; Step, Step+M, and All
TELBench XBench (Wang et al., 2026)	279 traces	All 279 traces for GPT-5.5, Claude Opus 4.6, and Gemini 3.7 Flash	Official harmful-span localization; 18-type F1 and joint recovery are TRACEACE extensions
TELBench BrowseComp (Wang et al., 2026)	373 traces	All 373 traces for GPT-5.5, Claude Opus 4.6, and Gemini 3.7 Flash	Official harmful-span localization; 18-type F1 and joint recovery are TRACEACE extensions

C.3 LLM MODELS

Table C.2 records the model labels used in the paper and their primary provider sources. All eight backbones are evaluated on the four cohorts of Table 1; the three frontier models are additionally used in the additional TRAIL/TELBench comparison of Table D.2.

Table C.2: LLM models used in the reported experiments.

Model label	Provider source	Scope in this study
GPT-5.5	OpenAI (OpenAI, 2026)	Tables 1 and D.2
Claude Opus 4.6	Anthropic (Anthropic, 2026)	Tables 1 and D.2
Gemini 3.7 Flash	Google (Google, 2026)	Tables 1 and D.2
DeepSeek V4 Flash	DeepSeek (DeepSeek, 2026)	Tables 1–3 and D.1
Qwen 3.8 Flash	Alibaba Cloud (Alibaba Cloud, 2026)	Table 1
Kimi K3	Moonshot AI (Moonshot AI, 2026)	Table 1
Doubao Seed 2.0 Pro	ByteDance Seed (ByteDance Seed, 2026)	Table 1
GLM 5.3 Flash	Z.ai (Z.ai, 2026)	Table 1

Model names are report-level labels; within a model–cohort setting, all methods use the same label and trace manifest.

C.4 COMPARISON METHODS

Within each model–dataset setting, every method receives the same prepared trace manifest and canonical trace content and must emit the same dataset-specific output schema (category and location on TRAIL; critical step, module, and type on AEB), so the comparison is matched at the trace level. Table C.3 specifies every comparison method and the table in which it appears: the main comparison (Table 1) uses LLM, ReAct, CORRECT, AgentDebug, AgentRx, AgentDebugX, and Agentic CLEAR, and the additional comparison (Table D.2) uses Vanilla LLM, ReAct, Self-Refine, Plan-and-Solve, CRITIC, CORRECT, AgentDebug, and AgentRx. It also gives the inference budget that materially affects cost. These comparison rows are transparent task-contract adaptations rather than claims of bit-for-bit reproduction of the source systems. The adapted methods preserve their characteristic inference mechanism but are not claimed as exact reproductions.

Table C.3: Detailed implementation of the comparison methods.

Method	Used in	Implementation under the shared trace-diagnosis contract
Vanilla LLM (LLM) (Deshpande et al., 2025)	1, D.2	One call over the complete compacted trace using the TRAIL-style monolithic judge prompt and parser. The rendered input is capped at 600,000 characters.
ReAct (Yao et al., 2023)	1, D.2	A generic reasoning-and-action agent navigates the same read-only trace workspace and terminates in the common prediction schema. It receives no scanners, structural anchors, clustering, or root-cause stage.
Self-Refine (Madaan et al., 2023)	D.2	One initial diagnosis followed by as many as two self-feedback and revision rounds. Feedback targets category and exact-location errors in the preceding diagnosis.

Method	Used in	Implementation under the shared trace-diagnosis contract
Plan-and-Solve (Wang et al., 2023b)	D.2	A first call produces a trace-inspection plan and analysis; a second call extracts the final typed locations in the shared schema.
CRITIC (Gou et al., 2024)	D.2	An initial diagnosis is followed by up to three tool-assisted critique–revision iterations. Each iteration permits at most seven read-only trace-tool turns and stops early after two unchanged answers. The evidence gate is disabled.
CORRECT (Yu et al., 2025)	1, D.2	A frozen, test-disjoint development cache is retrieved with character TF–IDF 3–5 grams and cosine similarity (top five), followed by detector calls (one per trace in Table D.2; per-trace calls for Table 1 are in Table D.1). Test overlap and online cache growth are forbidden.
AgentDebug (Zhu et al., 2025)	1, D.2	One phase-1 analysis call is issued for each visible span across five diagnostic modules, followed by one synthesis call. Canonical spans replace native module identifiers; counterfactual rerolling is not used.
AgentRx (Barke et al., 2026)	1, D.2	The canonical trace serves as the intermediate representation. Static-invariant, dynamic-invariant, checker, and final-judge calls; this yields four calls per trace in the additional comparison of Table D.2, and Table D.1 records the per-trace calls for Table 1. No task-specific domain policy is supplied.
AgentDebugX (Zhu et al., 2026)	1	The pinned upstream DeepDebug analyzer: a global read, structure-guided investigation, cross-examination, and an evidence-backed diagnosis with a repair suggestion. Its single native root is mapped to the dataset vocabulary, so it predicts one error per trace when scored against TRAIL's set of annotated errors; agent recovery is not executed.
Agentic CLEAR (Yehudai et al., 2026)	1	The upstream step-, trajectory-, and rubric-level judges followed by CLEAR's cohort-level synthesis, deduplication, and issue-membership functions. Only transport, persistence, canonical input conversion, and the benchmark readout are local.
TRACEACE	all	The complete proposed workflow: scanner analysis (anchors disabled; the structural channel is recorded but unscored), filtering and merge guards, same-fix clustering, representative investigation, evidence verification, and secondary-trace validation.

C.5 COMPONENT ABLATION PROTOCOL

Table 2 changes one component of **TRACEACE** at a time with DeepSeek V4 Flash, holding the prepared traces and all other stages fixed. *Analysis variants* are scored on TRAIL. *Flat chunks* replaces episode-aware packing (Appendix B) with fixed-size span windows that ignore episode and session boundaries. *Unified scanner* replaces the six taxonomy-conditioned scanners with one scanner that owns the full 21-leaf vocabulary. *w/o FP filters* disables the model-assisted false-positive filters while keeping the merge guards. *Explanation variants* are scored on AEB with the analysis outputs fixed. *w/o evidence gate* admits hypotheses without the evidence-integrity check of Appendix B. *Random representative* replaces the representative utility with a uniformly random cluster member. *w/o cross-trace validation* keeps the representative's hypothesis for the whole cluster without replaying it on secondary members, so no member can refute it or trigger re-diagnosis. Table 3 freezes the analysis outputs and cohort: *Per-trace RCA* runs the Root-Cause Analyzer on every trace, *Signature-only grouping* disables only the LLM cluster merge, and all arms share the same per-trace readout and typed-root scorer; Reports counts primary investigations, and Tok. counts every post-analysis call.

C.6 Detailed Detection and Localization Scoring

The scorer normalizes category strings to the vocabulary of each dataset and converts both references and final predictions into sets; duplicate records therefore receive no extra credit. Category presence is scored once per trace and category. Precision, recall, and F1 use zero when their corresponding denominator is zero, and weighted F1 uses reference category support as in Equation C.3. Exact localization requires string equality with the source `span_id`. Localization and joint recovery use the reference-set denominators in Equation C.4, so extra locations are not directly penalized in those two columns; category F1, whose false positives count every predicted category absent from the reference, provides the complementary precision diagnostic. TELBench retains its released harmful-span localization as the official endpoint. Applying the category and joint definitions to the released 18-type vocabulary is a separate **TRACEACE** analysis, not an official TELBench score.

C.7 Root-Cause Evaluation Protocol

Root-cause reports are evaluated in four layers (Table C.4) rather than collapsed into one score. This separation prevents a valid quotation from being mistaken for a correct causal account, or an internal Cross-Trace Validator verdict from being mistaken for reference accuracy. The layered diagnostics of this protocol are computed on the separate 30-trace held-out TRAIL-GAIA cohort; the AEB root-accuracy results of Tables 2 and 3 are scored by the AEB All metric.

Table C.4: Layered evaluation of root-cause explanations.

Layer	Question	Measurements	Reference access
L0	Is the cited evidence recoverable?	Valid span, exact quotation, and broader span-grounding rates from Equation C.5	Investigator-visible workspace only
L1	Does the evidence reach the annotated failure?	Any reference-error-span hit and annotated first-error hit, scoped to the cluster's categories	Exact locations opened after submission
L2	Does the proposed cause agree with the reference account?	Separate full, partial, or no-support verdict from a post-hoc reference judge	Full reference used by the post-hoc judge
L3	Does the hypothesis survive other traces?	Confirmed, partial, refuted, and singleton outcomes; cross-trace agreement reported separately	Other trace workspaces, without their labels

The fabricated-hypothesis control injects clearly trace-irrelevant causes into the same clusters as the real hypotheses and reuses the ordinary secondary-trace Cross-Trace Validator and verdict schema for both sets. On the held-out cohort, 5 fabricated hypotheses (one per selected cluster) are each replayed on two secondary traces, and all 10 resulting verdicts are refuted; the 22 verdicts on the real hypotheses of the same clusters split 9/4/9 confirmed/partial/refuted. RefuteRate (Equation C.6) measures rejection on these intentionally obvious negative controls, not recall on plausible causes, discrimination among near-miss mechanisms, or ordinary RCA accuracy.

The per-trace versus clustered comparison applies the same Root-Cause Analyzer, evidence settings, and independent evaluator either to every trace separately or to one representative per discovered cluster, and counts the primary investigation reports shown to an engineer; secondary checks are not counted as additional primary reports. ReviewReduction (Equation C.7) therefore measures human-facing review volume only. The cluster arm still performs signature grouping, conservative merging, secondary-trace validation, and possible re-diagnosis, whose calls remain part of the computation; the reported totals of 2.07M versus 3.19M tokens and 200 versus 334 calls in Table D.6 cover prompt plus completion over each whole arm.

C.8 Matched Comparisons and Uncertainty

Methods are compared only within the same model–dataset setting, under the cell-admission rule stated with Equation C.8. The usage analysis of the additional comparison (Table D.2) is stricter:

it includes only the 10 settings in which all nine methods have positive token and call totals, and computes prompt-plus-completion tokens per prepared trace using Equation C.8.

Each reported cell is one realized remote-model run. The tables show single-run point estimates rather than random-seed means, and no confidence interval or significance test is attached. Cross-setting averages give each represented setting one unweighted vote; they are descriptive stability summaries rather than pooled estimates over traces.

C.9 Additional Result Commentary

This subsection comments on the additional TRAIL/TELBench comparison of Table D.2.

Scope of the comparison. The additional experiments evaluate failure-type detection, exact localization, and evidence-grounded root-cause explanation: **TRACEACE** is compared with eight methods on TRAIL GAIA, TRAIL SWE, TELBench XBench, and TELBench BrowseComp under GPT-5.5, Claude Opus 4.6, and Gemini 3.7 Flash, and model-call usage is measured under matched conditions (Table D.5).

Reading Table D.2. Table D.2 compares the nine methods under the three frontier models on the four public sets, with F1, localization, and joint columns per dataset and subscripts giving the change from Vanilla LLM. Because the evaluation sets differ in size and label density, methods are compared within the same model–dataset setting: each of the 12 settings receives one unweighted vote in the cross-setting summaries, which are therefore not pooled trace-level estimates, and every cell is a single realized run without a confidence interval.

The joint-accuracy tie and loss. The one tie is Gemini 3.7 Flash on XBench, where **TRACEACE** and CORRECT both reach 14.1% joint accuracy. The one loss is GPT-5.5 on SWE-31, where AgentDebug reaches 15.1% joint accuracy and 45.8% localization against **TRACEACE**'s 14.6% and 42.5%; SWE-31 is also the smallest split, where a single trace moves a macro-averaged score by up to 3.2 points.

Why all three metrics are kept. On TELBench, ReAct or AgentDebug often localize harmful spans at least as well as **TRACEACE** yet reach lower joint accuracy: the pattern is not uniform localization dominance but a more consistent conjunction of a correct category with a correct location, which is why Table D.2 keeps all three metrics rather than a single summary column.

GPT-5.5 reference levels. Under GPT-5.5, **TRACEACE** reports 50.3/81.8/54.7 weighted F1/localization/joint on TRAIL GAIA and 28.2% and 46.1% joint accuracy on XBench and BrowseComp (Table D.2); these are the absolute levels against which the other two backbones in the same table are read.

Vanilla LLM is not a fixed floor. The same single-pass judge falls to zero joint accuracy on SWE-31 under Claude Opus 4.6 and Gemini 3.7 Flash, and its GAIA localization falls far below its GPT-5.5 level, so the subscripts in Table D.2 are meaningful only within a model block; **TRACEACE** retains high localization under all three backbones, consistent with the taxonomy-conditioned scanners reducing dependence on single-pass span selection.

Layered explanation diagnostics. The layered diagnostics in Table E.1 (Appendix E) show that the cluster arm attains fully span-valid and grounded evidence with 74.2% verbatim quotation but hits an annotated error span for only 37.5% of clusters (50.0% unscoped), with cause verdicts of 2 full, 3 partial, and 3 none, so grounding and reference correctness are distinct properties. The per-trace arm, scored per trace, reaches 90.0% GT-span and 66.7% first-error hits with 66.1% verbatim quotation; the two arms thus trade reference coverage per unit against review volume and evidence quality, and because their units differ, the columns are not directly comparable.

Cost analysis scope. The matched usage subset, the treatment of missing ledgers, and the per-method token, call, and accuracy means behind Table D.7 are given in Appendix D (Table D.5).

Table D.1: Inference and setup cost. DeepSeek V4 Flash on the same cohorts as Table 2. Tok. (k) and Calls are per trace; Setup gives one-time LLM totals shared by all four cohorts. Recorded retries and reused responses are included.

	TRAIL				AEB				Setup (total)	
	GAIA		SWE		ALFWorld		WebShop			
Method	Tok.↓	Calls↓	Tok.↓	Calls↓	Tok.↓	Calls↓	Tok.↓	Calls↓	Tok.↓	Calls↓
LLM	36.1	1.0	90.0	1.1	97.4	1.0	82.1	1.0	0.0	0.0
ReAct	191.8	7.6	681.4	21.2	618.7	13.0	648.1	17.5	0.0	0.0
CORRECT	93.0	2.0	474.5	2.5	183.7	2.2	127.5	2.0	10710.5	128.0
AgentDebug	2787.4	79.4	5551.8	119.0	5481.6	120.1	4856.3	119.4	0.0	0.0
AgentRx	1263.7	24.9	2900.9	37.7	2789.5	39.2	2700.6	40.1	0.0	0.0
AgentDebugX	141.9	7.0	440.9	8.3	203.9	8.5	162.5	8.3	0.0	0.0
Agentic CLEAR	318.3	23.9	1149.9	32.1	682.8	61.0	604.4	63.0	0.0	0.0
⋆ TRACEACE	1033.2	53.3	2255.3	80.8	2338.9	74.6	2342.2	73.6	0.0	0.0

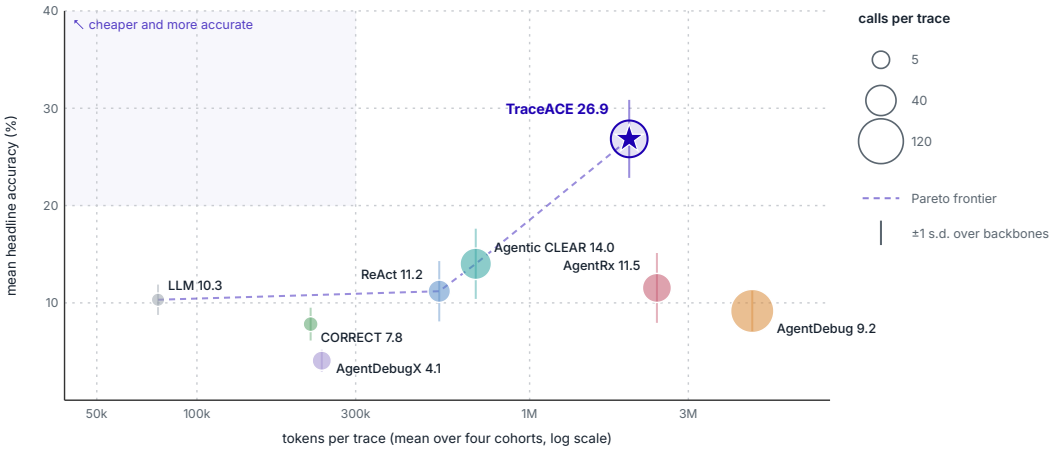


Figure D.1: Accuracy–token trade-off. Mean headline accuracy over the eight backbones of Table 1 against mean tokens per trace of Table D.1; bubble area encodes calls per trace, bars give one standard deviation over backbones, and the dashed line joins the non-dominated methods.

D Additional Results

D.1 Per-Trace Inference and Setup Cost

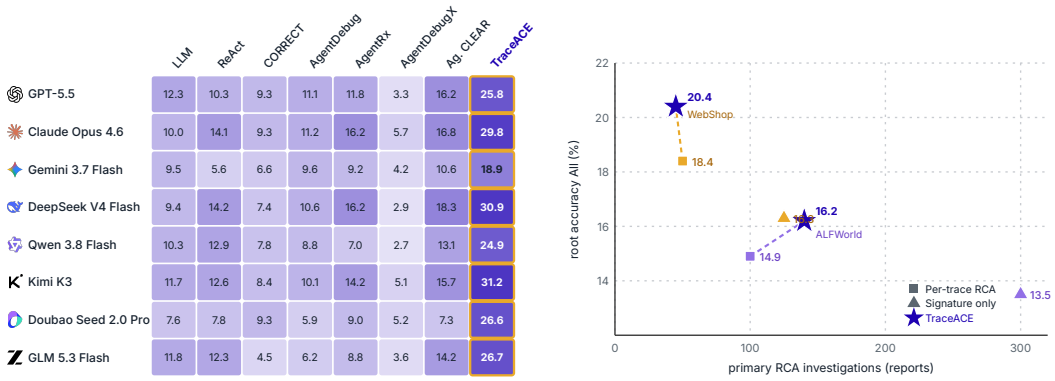
Table D.1 gives the full cost accounting behind Figure 3: tokens and calls per trace for every method on the four cohorts, and the one-time setup that CORRECT requires.

Figure D.1 condenses Tables 1 and D.1 into one accuracy–token trade-off. For each method, the vertical axis averages the four headline metrics (TRAIL GAIA and SWE Joint, AEB ALFWorld and WebShop All) over the eight backbones, and the horizontal axis averages its DeepSeek V4 Flash per-trace tokens (Table D.1) over the four cohorts.

Figure D.2(a) shows that **TRACEACE** has the highest mean headline accuracy under every backbone, and Figure D.3 shows the component ablations of Table 2 as signed changes (variants defined in Appendix C.5).

D.2 Additional TRAIL/TELBench Experiments

This appendix reports an additional comparison on TRAIL and TELBench with three frontier backbones and nine methods (Table D.2). It uses a different method line-up and implementation from Section 4, so its numbers are not pooled with Tables 1–3 or Table D.1; all statistics in this appendix refer to Table D.2 unless stated otherwise.



(a) Mean headline accuracy per backbone and method (b) Investigations against root accuracy

Figure D.2: Where the gains appear. (a) Each cell averages the four headline metrics of Table 1; outlined cells mark the best method per backbone. (b) Collection-level diagnosis of Table 3: the full cluster workflow raises root accuracy over per-trace investigation on both AEB environments while needing far fewer reports than signature-only grouping.

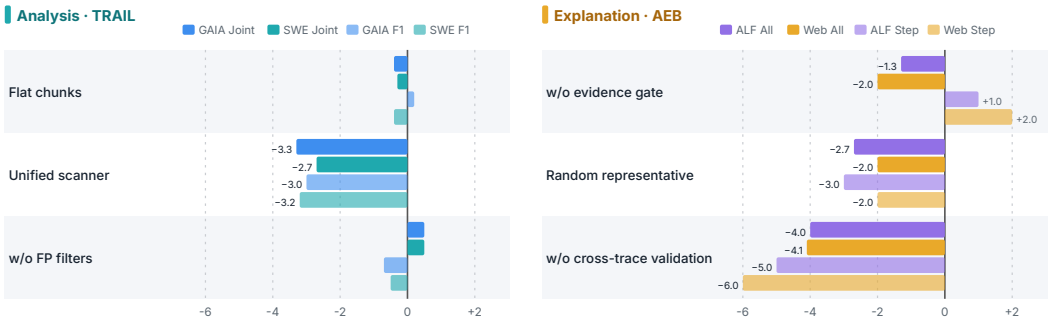


Figure D.3: Component ablations as changes from the full system. Each bar is the difference between one variant of Table 2 and full TRACEACE (DeepSeek V4 Flash), in points; negative values mean the component helps.

D.3 CROSS-SETTING METHOD RANKING

Table D.2 is easiest to read block by block, but it does not directly show whether a method is consistently near the top across settings. Table D.3 therefore aggregates the 12 public model–evaluation settings (three frontier backbones on TRAIL GAIA, TRAIL SWE, TELBench XBench, and TELBench BrowseComp) in two descriptive ways: an unweighted mean of the reported percentages and an average within-block rank. This avoids giving a large evaluation set more votes than a small one while still exposing how often a method is competitive.

TRACEACE attains the maximum joint value in 11 of the 12 blocks (10 strict wins and 1 tie) and has a mean joint rank of 1.13. Its category F1 is on top in 7 blocks (mean rank 2.33), whereas localization is on top in 5 blocks (mean rank 2.25). AgentDebug and ReAct are the closest competitors on localization, with mean ranks 3.38 and 4.42, and AgentDebug is also the closest competitor on F1 (mean rank 3.92). The difference between these columns is substantive: the proposed method's most stable advantage is the conjunction of a correct category and a useful location, not universal leadership on either marginal metric alone.

D.4 WHERE THE PAIRED GAINS ARISE

Table D.4 compares TRACEACE with the strongest baseline separately for every metric and evaluation set. Over the 12 blocks, the joint gap has mean 10.9 and median 7.7 points and ranges from -0.5 to 33.6 points. Reporting the paired mean and the W/T/L counts per evaluation set, rather than one pooled figure, ensures that one large gain on one benchmark cannot hide a loss on another.

Table D.2: Additional comparison on TRAIL and TELBench under three frontier LLM models. Nine methods on TRAIL GAIA-116, TRAIL SWE-31, TELBench XBench-279, and TELBench BrowseComp-373, with a separate method line-up and implementation. Each dataset reports weighted category F1, exact localization, and joint accuracy (%). Subscripts give the percentage-point change from Vanilla LLM within the same model–dataset–metric setting; bold and underlined mark the best and second-best values within a model block. TELBench localization is the official endpoint; typed F1 and joint accuracy are **TRACEACE** extensions.

Model	Method	TRAIL (GAIA)			TRAIL (SWE)			TELBench (XBench)			TELBench (BrowseComp)		
		F1↑	Loc.↑	Joint↑	F1↑	Loc.↑	Joint↑	F1↑	Loc.↑	Joint↑	F1↑	Loc.↑	Joint↑
gpt-5.5	Vanilla LLM	40.4	61.6	20.8	51.0	22.8	4.5	45.0	37.4	23.3	63.0	64.7	37.8
	ReAct	28.1$_{-12.3}$	56.5$_{-5.1}$	14.3$_{-6.5}$	42.6$_{-8.4}$	20.2$_{-2.6}$	3.5$_{-1.0}$	53.5$_{+8.5}$	**63.6**$_{+26.2}$	27.2$_{+3.9}$	64.1$_{+1.1}$	**84.6**$_{+19.9}$	38.5$_{+0.7}$
	Self-Refine	41.4$_{+1.0}$	68.3$_{+6.7}$	21.1$_{+0.3}$	51.3$_{+0.3}$	27.0$_{+4.2}$	11.1$_{+6.6}$	53.3$_{+8.3}$	49.0$_{+11.6}$	25.9$_{+2.6}$	63.8$_{+0.8}$	54.5$_{-10.2}$	33.3$_{-4.5}$
	Plan-and-Solve	38.7$_{-1.7}$	60.5$_{-1.1}$	18.3$_{-2.5}$	49.7$_{-1.3}$	18.4$_{-4.4}$	3.9$_{-0.6}$	54.7$_{+9.7}$	41.3$_{+3.9}$	23.3$_{\pm0.0}$	63.0$_{\pm0.0}$	69.2$_{+4.5}$	37.2$_{-0.6}$
	CRITIC	29.4$_{-11.0}$	61.1$_{-0.5}$	16.0$_{-4.8}$	34.3$_{-16.7}$	12.9$_{-9.9}$	1.5$_{-3.0}$	60.1$_{+15.1}$	37.4$_{\pm0.0}$	21.8$_{-1.5}$	36.0$_{-27.0}$	46.1$_{-18.6}$	5.8$_{-32.0}$
	CORRECT	28.5$_{-11.9}$	57.1$_{-4.5}$	11.5$_{-9.3}$	35.4$_{-15.6}$	8.5$_{-14.3}$	0.2$_{-4.3}$	50.2$_{+5.2}$	41.3$_{+3.9}$	21.8$_{-1.5}$	56.1$_{-6.9}$	62.2$_{-2.5}$	31.4$_{-6.4}$
	AgentDebug	31.9$_{-8.5}$	81.1$_{+19.5}$	12.6$_{-8.2}$	44.3$_{-6.7}$	**45.8**$_{+23.0}$	**15.1**$_{+10.6}$	**61.3**$_{+16.3}$	36.1$_{-1.3}$	25.6$_{+2.3}$	45.2$_{-17.8}$	51.9$_{-12.8}$	15.4$_{-22.4}$
	AgentRx	29.2$_{-11.2}$	61.4$_{-0.2}$	10.9$_{-9.9}$	37.3$_{-13.7}$	28.7$_{+5.9}$	4.7$_{+0.2}$	60.3$_{+15.3}$	41.3$_{+3.9}$	21.8$_{-1.5}$	53.8$_{-9.2}$	55.1$_{-9.6}$	18.6$_{-19.2}$
	⋆ **TRACEACE**	**50.3**$_{+9.9}$	**81.8**$_{+20.2}$	**54.7**$_{+33.9}$	**59.1**$_{+8.1}$	42.5$_{+19.7}$	14.6$_{+10.1}$	50.6$_{+5.6}$	41.3$_{+3.9}$	**28.2**$_{+4.9}$	**65.1**$_{+2.1}$	69.2$_{+4.5}$	**46.1**$_{+8.3}$
claude-opus-4-6	Vanilla LLM	38.5	9.4	1.3	40.1	7.2	0.0	44.4	25.9	19.5	57.4	49.4	36.5
	ReAct	24.2$_{-14.3}$	39.1$_{+29.7}$	4.5$_{+3.2}$	14.2$_{-25.9}$	9.9$_{+2.7}$	0.8$_{+0.8}$	37.2$_{-7.2}$	33.6$_{+7.7}$	14.1$_{-5.4}$	55.2$_{-2.2}$	44.2$_{-5.2}$	28.9$_{-7.6}$
	Self-Refine	28.5$_{-10.0}$	22.1$_{+12.7}$	5.1$_{+3.8}$	44.6$_{+4.5}$	2.8$_{-4.4}$	0.0$_{\pm0.0}$	44.4$_{\pm0.0}$	27.4$_{+1.5}$	17.9$_{-1.6}$	61.6$_{+4.2}$	57.0$_{+7.6}$	38.5$_{+2.0}$
	Plan-and-Solve	42.3$_{+3.8}$	27.6$_{+18.2}$	12.2$_{+10.9}$	43.1$_{+3.0}$	6.3$_{-0.9}$	0.0$_{\pm0.0}$	40.6$_{-3.8}$	25.9$_{\pm0.0}$	15.4$_{-4.1}$	56.0$_{-1.4}$	49.4$_{\pm0.0}$	32.7$_{-3.8}$
	CRITIC	34.0$_{-4.5}$	19.2$_{+9.8}$	2.4$_{+1.1}$	6.2$_{-33.9}$	7.8$_{+0.6}$	0.0$_{\pm0.0}$	37.2$_{-7.2}$	45.1$_{+19.2}$	15.6$_{-3.9}$	48.2$_{-9.2}$	51.3$_{+1.9}$	31.4$_{-5.1}$
	CORRECT	28.3$_{-10.2}$	14.9$_{+5.5}$	1.3$_{\pm0.0}$	14.3$_{-25.8}$	8.7$_{+1.5}$	0.0$_{\pm0.0}$	46.2$_{+1.8}$	**50.5**$_{+24.6}$	15.6$_{-3.9}$	53.1$_{-4.3}$	48.7$_{-0.7}$	30.8$_{-5.7}$
	AgentDebug	29.8$_{-8.7}$	46.7$_{+37.3}$	6.4$_{+5.1}$	20.1$_{-20.0}$	25.3$_{+18.1}$	1.4$_{+1.4}$	**48.2**$_{+3.8}$	42.8$_{+16.9}$	14.1$_{-5.4}$	56.5$_{-0.9}$	**73.7**$_{+24.3}$	32.7$_{-3.8}$
	AgentRx	24.1$_{-14.4}$	23.1$_{+13.7}$	5.7$_{+4.4}$	14.0$_{-26.1}$	6.9$_{-0.3}$	0.0$_{\pm0.0}$	43.7$_{-0.7}$	36.7$_{+10.8}$	10.3$_{-9.2}$	**69.8**$_{+12.4}$	48.7$_{-0.7}$	22.4$_{-14.1}$
	⋆ **TRACEACE**	**57.4**$_{+18.9}$	**85.7**$_{+76.3}$	**43.3**$_{+42.0}$	**44.7**$_{+4.6}$	**41.6**$_{+34.4}$	**13.9**$_{+13.9}$	42.8$_{-1.6}$	41.3$_{+15.4}$	**23.3**$_{+3.8}$	63.8$_{+6.4}$	69.2$_{+19.8}$	**48.1**$_{+11.6}$
gemini-3.7-flash	Vanilla LLM	18.7	26.2	6.3	18.8	1.9	0.0	13.2	14.4	2.6	34.7	24.4	11.5
	ReAct	22.7$_{+4.0}$	38.1$_{+11.9}$	8.2$_{+1.9}$	8.0$_{-10.8}$	8.7$_{+6.8}$	0.0$_{\pm0.0}$	14.5$_{+1.3}$	19.5$_{+5.1}$	7.7$_{+5.1}$	34.9$_{+0.2}$	**36.5**$_{+12.1}$	8.3$_{-3.2}$
	Self-Refine	21.8$_{+3.1}$	28.5$_{+2.3}$	6.9$_{+0.6}$	20.5$_{+1.7}$	4.2$_{+2.3}$	0.0$_{\pm0.0}$	17.5$_{+4.3}$	22.1$_{+7.7}$	10.3$_{+7.7}$	34.7$_{\pm0.0}$	19.9$_{-4.5}$	14.1$_{+2.6}$
	Plan-and-Solve	17.2$_{-1.5}$	28.4$_{+2.2}$	5.6$_{-0.7}$	21.1$_{+2.3}$	4.8$_{+2.9}$	0.0$_{\pm0.0}$	21.1$_{+7.9}$	23.3$_{+8.9}$	10.3$_{+7.7}$	32.6$_{-2.1}$	23.7$_{-0.7}$	9.6$_{-1.9}$
	CRITIC	16.7$_{-2.0}$	26.5$_{+0.3}$	7.9$_{+1.6}$	8.8$_{-10.0}$	6.8$_{+4.9}$	0.6$_{+0.6}$	23.7$_{+10.5}$	23.3$_{+8.9}$	10.3$_{+7.7}$	42.8$_{+8.1}$	25.6$_{+1.2}$	11.5$_{\pm0.0}$
	CORRECT	14.3$_{-4.4}$	25.4$_{-0.8}$	4.2$_{-2.1}$	7.7$_{-11.1}$	6.9$_{+5.0}$	0.0$_{\pm0.0}$	**32.4**$_{+19.2}$	29.7$_{+15.3}$	**14.1**$_{+11.5}$	**43.3**$_{+8.6}$	16.7$_{-7.7}$	7.7$_{-3.8}$
	AgentDebug	23.2$_{+4.5}$	37.4$_{+11.2}$	6.2$_{-0.1}$	23.0$_{+4.2}$	16.2$_{+14.3}$	4.3$_{+4.3}$	19.6$_{+6.4}$	**33.6**$_{+19.2}$	10.3$_{+7.7}$	40.0$_{+5.3}$	23.1$_{-1.3}$	6.4$_{-5.1}$
	AgentRx	15.4$_{-3.3}$	17.8$_{-8.4}$	3.1$_{-3.2}$	6.9$_{-11.9}$	10.6$_{+8.7}$	0.6$_{+0.6}$	18.4$_{+5.2}$	**33.6**$_{+19.2}$	10.3$_{+7.7}$	30.2$_{-4.5}$	18.6$_{-5.8}$	5.8$_{-5.7}$
	⋆ **TRACEACE**	**45.8**$_{+27.1}$	**71.8**$_{+45.6}$	**28.2**$_{+21.9}$	**25.0**$_{+6.2}$	**26.2**$_{+24.3}$	**12.1**$_{+12.1}$	25.9$_{+12.7}$	29.7$_{+15.3}$	**14.1**$_{+11.5}$	38.9$_{+4.2}$	24.4$_{\pm0.0}$	**17.9**$_{+6.4}$

Table D.3: Unweighted descriptive summary over the 12 public model–evaluation settings (three frontier backbones on TRAIL GAIA, TRAIL SWE, TELBench XBench, and TELBench BrowseComp). Mean is the block mean; Rank is the mean within-block descending rank with average ranks for ties (lower is better); Top counts blocks in which a method attains the maximum. Cohort sizes differ, so this is a cross-setting stability diagnostic rather than a pooled-dataset score.

Method	F1			Localization			Joint		
	Mean	Rank↓	Top	Mean	Rank↓	Top	Mean	Rank↓	Top
Vanilla LLM	38.8	4.96	0	28.8	6.50	0	13.7	5.08	0
ReAct	33.3	6.29	0	37.9	4.42	3	13.0	5.38	0
Self-Refine	40.3	4.08	0	31.9	5.58	0	15.4	3.96	0
Plan-and-Solve	40.0	4.63	0	31.6	5.79	0	14.0	5.21	0
CRITIC	31.5	6.29	0	30.3	5.67	0	10.4	5.75	0
CORRECT	34.2	6.00	2	30.9	6.04	1	11.6	6.75	1
AgentDebug	36.9	3.92	2	42.8	3.38	3	12.5	4.83	1
AgentRx	33.6	6.50	1	31.9	5.38	1	9.5	6.92	0
⋆ **TRACEACE**	47.5	2.33	7	52.1	2.25	5	28.7	1.13	11

The largest joint improvement occurs on TRAIL GAIA, where all three models favor **TRACEACE** and the mean paired gain is 28.2 points. TRAIL SWE is more mixed: two models improve and one declines; under GPT-5.5, AgentDebug reaches 15.1 joint accuracy and 45.8 localization on SWE-31 against 14.6 and 42.5 for **TRACEACE**, and this block is the minimum of the joint gap (-0.5 points). On both TELBench sets, standalone F1 and localization trail their respective best baselines on average, yet joint accuracy is positive in five blocks and tied in one (Gemini 3.7 Flash on XBench, 14.1 against CORRECT). Because the best baseline is selected independently by metric, this pattern indicates a better balance of type and location decisions rather than simultaneous dominance of both marginal metrics.

Table D.4: Paired difference between TRACEACE and the strongest non-TRACEACE baseline in each of the 12 public model–evaluation settings (three frontier backbones × four public evaluation sets). Δ is the unweighted mean absolute percentage-point difference; W/T/L counts improvements, ties, and losses. The strongest baseline is selected independently for each metric.

Evaluation set	Blocks	ΔF1	F1 W/T/L	ΔLoc.	Loc. W/T/L	ΔJoint	Joint W/T/L
TRAIL GAIA	3	+15.5	3/0/0	+24.5	3/0/0	+28.2	3/0/0
TRAIL SWE	3	+3.3	3/0/0	+7.7	2/0/1	+6.6	2/0/1
TELBench XBench	3	-7.5	0/0/3	-11.8	0/0/3	+1.6	2/1/0
TELBench BrowseComp	3	-3.1	1/0/2	-10.7	0/0/3	+7.0	3/0/0
Overall	12	+2.0	7/0/5	+2.4	5/0/7	+10.9	10/1/1

D.5 MATCHED COST–QUALITY BREAKDOWN

The additional comparison also records token usage. Table D.5 gives all nine methods on the same 10-setting matched subset, that is, the public settings in which every method has a positive usage ledger (103 of the 108 public result cells carry one; missing records are excluded rather than interpreted as zero cost). It is intentionally based on tokens and calls, not provider prices, because the models and relay routes do not share one stable price schedule.

Table D.5: Cost–quality summary of the additional comparison (Table D.2) on the 10 matched public settings with positive usage ledgers for all nine methods. Columns are unweighted block means; ΔJoint is paired against Vanilla LLM (the dash marks the reference row) and the token multiple uses its mean as 1. Tokens cover prompt plus completion only, not price, latency, or review time.

Method	F1	Loc.	Joint	ΔJoint	Tokens/trace (k)	Calls/trace	Token ×
Vanilla LLM	38.7	32.9	16.3	—	21.9	1.0	1.00
ReAct	36.1	40.6	15.1	-1.2	136.5	7.6	6.23
Self-Refine	41.0	35.8	17.9	+1.6	81.3	3.6	3.71
Plan-and-Solve	39.5	34.5	15.6	-0.7	43.3	2.0	1.98
CRITIC	33.7	33.6	12.2	-4.0	155.1	14.0	7.07
CORRECT	36.7	34.7	13.7	-2.6	21.2	1.0	0.96
AgentDebug	39.3	44.2	14.3	-2.0	116.6	20.6	5.31
AgentRx	36.5	35.3	10.9	-5.4	75.8	4.0	3.46
⋆ TRACEACE	46.7	49.7	28.7	+12.5	79.2	12.2	3.61

TRACEACE averages 28.7% joint accuracy at 79.2 thousand tokens and 12.2 model calls per trace. Self-Refine has nearly the same token volume (81.3 thousand) but 17.9% joint accuracy. ReAct, AgentDebug, and CRITIC consume more tokens in this subset (136.5, 116.6, and 155.1 thousand) without matching the proposed method's joint score (15.1%, 14.3%, and 12.2%). Conversely, CORRECT (21.2 thousand), the single-call Vanilla LLM baseline (21.9 thousand), Plan-and-Solve (43.3 thousand), and AgentRx (75.8 thousand, at 10.9% joint accuracy) are cheaper than TRACEACE, the first three substantially so; the data therefore support a cost–quality trade-off rather than a claim of minimum cost.

D.6 ERROR PATTERNS AND INTERPRETATION LIMITS

Two patterns recur across the aggregate results. First, category detection wins more often than exact span selection: TRACEACE wins 7 of 12 F1 comparisons but only 5 localization comparisons, and the TELBench rows make this residual localization problem especially visible. Second, strong joint results can coexist with a negative localization gap when a method avoids pairing a plausible location with the wrong error type.

These are cross-setting descriptions of single runs. They do not estimate variance under repeated sampling or remote-model nondeterminism. A larger evaluation should repeat each cell, publish fixed public manifests, and add human same-fix annotations so that cluster quality can be evaluated with pairwise or B-cubed metrics rather than only through coverage and downstream validation.

D.7 Explanation Controls on a Held-Out TRAIL-GAIA Cohort

Additional root-cause explanation controls are evaluated on a 30-trace held-out TRAIL-GAIA cohort separate from the detection tables (Table D.6), with two controls on whether the Cross-Trace Validator rationalizes any cause it is given. First, 5 fabricated hypotheses, each replayed on two secondary traces, receive 10 refuted verdicts out of 10, whereas the 22 verdicts on the real hypotheses of the same clusters split 9/4/9 confirmed/partial/refuted; this establishes rejection of clear negatives, not explanation accuracy. Second, investigating one representative per cluster reduces primary reports from 30 to 8 (73.3%) and raises grounded evidence from 96.5% to 100.0%, yet tokens rise from 2.07M to 3.19M and calls from 200 to 334: the reduction quantifies material for an engineer, not compute. Appendix E (Table E.1) shows that grounding and reference correctness are distinct properties.

Table D.6: Root-cause controls on the TRAIL-GAIA held-out cohort (30 traces). (a) One constructed trace-irrelevant cause per cluster is injected beside that cluster's real hypothesis, and both pass through the ordinary Cross-Trace Validator; counts are verdicts, one per secondary trace, not hypotheses. (b) Per-trace versus cluster-level investigation of the same traces; reports are human-facing investigation units, grounding is re-measured by the independent evaluator, and tokens cover the whole arm. Fewer reports do not mean lower token cost: the cluster-level arm also merges clusters, validates across traces, and may re-diagnose.

(a) Fabricated-hypothesis negative control (5 clusters)

Hypothesis set	Verdicts	Confirmed	Partial	Refuted	Refuted (%)
Fabricated hypotheses (5, two secondary traces each)	10	0	0	**10**	**100.0**
Real hypotheses (same clusters)	22	9	4	9	40.9

(b) Per-trace vs. cluster-level investigation

Investigation unit	Reports	Grounded evidence (%)	Tokens (M)	LLM calls
Per trace	30	96.5	2.07	200
Cluster-level	**8**	**100.0**	3.19	334

Table D.7: Matched token–quality comparison for the additional TRAIL/TELBench comparison of Table D.2. Values are unweighted means over the 10 public model–evaluation settings with positive usage for all nine methods; ΔJoint is paired against Vanilla LLM (the dash marks the reference row) and Token × normalizes mean prompt-plus-completion tokens per trace to Vanilla LLM. Tokens do not represent price, latency, or human review time.

Method	Joint↑	ΔJoint	Tokens/trace (k)↓	Calls/trace↓	Token × ↓
Vanilla LLM	16.3	—	21.9	1.0	1.00
ReAct	15.1	-1.2	136.5	7.6	6.23
Self-Refine	17.9	+1.6	81.3	3.6	3.71
Plan-and-Solve	15.6	-0.7	43.3	2.0	1.98
CRITIC	12.2	-4.0	155.1	14.0	7.07
CORRECT	13.7	-2.6	21.2	1.0	0.96
AgentDebug	14.3	-2.0	116.6	20.6	5.31
AgentRx	10.9	-5.4	75.8	4.0	3.46
⋆ **TRACEACE**	28.7	+12.5	79.2	12.2	3.61

E Explanation Diagnostics

The layered explanation diagnostics of this appendix are computed on the separate deterministic 30-trace held-out TRAIL-GAIA cohort; AEB root accuracy is reported in Tables 2 and 3. The four evaluation layers follow the protocol of Appendix C: recoverability of the cited evidence (Equation C.5), reference-span and first-error hits, a separate full/partial/no-support cause verdict, and cross-trace validation (Equations A.15 and 3). The reference annotations remain hidden from the Root-Cause Analyzer throughout and are opened only by the post-hoc evaluator.

E.1 Layered Diagnostics for the Two Investigation Arms

Table E.1 reports the layered diagnostics behind the two investigation arms that Table D.6 compares at the report level. The cluster-level arm performs signature grouping, conservative merge, representative investigation, secondary validation, and possible re-diagnosis, and yields one primary report per discovered cluster; the per-trace arm applies the same Root-Cause Analyzer, evidence settings, and evaluator independently to every trace. The units therefore differ: eight clusters with 31 cited evidence items in the cluster-level arm and 30 traces with 115 items in the per-trace arm, so evidence rates use arm-specific denominators.

Table E.1: Layered RCA diagnostics for the two investigation arms on the TRAIL-GAIA held-out cohort (30 traces). Units are clusters for the cluster-level arm and traces for the per-trace arm. Grounding is computed over evidence items; GT-span hit and first-error hit are computed over GT-bearing units. For the cluster-level arm the GT-span hit is scoped to the cluster's own category (unscoped, any-category hit: 50.0). Cause verdicts (full/partial/none) and agreement, the persisted cross-trace confirmation rate, exist only for the cluster-level arm; the per-trace arm has no reference-judge pass and no cross-trace validation, so those cells are left blank.

Arm	Units	Evidence	Valid	Verbatim	Grounded	GT hit	First	Full/partial/none	Agreement
Cluster-level (8 clusters)	8	31	100.0	74.2	100.0	37.5	37.5	2/3/3	37.5
Per trace (30 traces)	30	115	100.0	66.1	96.5	90.0	66.7	—	—

Both arms cite only resolvable spans (100% valid). The cluster-level arm reaches 100% grounded evidence with 74.2% strict verbatim quotation; the per-trace arm reaches 96.5% grounded evidence with 66.1% verbatim quotation. The GT-span hit of the cluster-level arm is scoped: a cluster's cited evidence counts as a hit only when it reaches a reference error span of the cluster's own category, which gives 37.5% (three of eight clusters). Relaxing the scope to any annotated error span raises the hit to 50.0%, that is, one additional cluster cites a reference span that the annotation assigns to a category other than the cluster's. The first-error hit is likewise 37.5%. The per-trace arm's 90.0% GT-span hit and 66.7% first-error hit are computed over GT-bearing traces without category scoping; the two rows differ in unit, denominator, and scoping, and are not a head-to-head comparison of localization quality.

Recoverable quotations and correct causes are different endpoints. The cluster-level arm reaches 100% grounded evidence yet only a 37.5% GT-span hit, and its reference-judge verdicts are 2 full, 3 partial, and 3 no-support: a quotation may be recoverable from the trace without citing the benchmark's annotated error span, and citing the annotated span does not by itself make the causal account agree with the reference. The 37.5% agreement is the persisted cross-trace confirmation rate of Equation 3; it is an internal validation outcome rather than a reference-answer score and is never merged with the reference-judge scores. The per-trace arm has no reference-judge pass and no cross-trace validation, so its verdict and agreement cells are blank.

E.2 Fabricated-Hypothesis Rejection

This negative control tests whether the Cross-Trace Validator merely rationalizes a supplied cause. For each of 5 selected clusters of the held-out cohort, the original hypothesis is replaced by one constructed cause unrelated to the trace, while the ordinary Cross-Trace Validator and verdict schema remain unchanged; each fabricated hypothesis is replayed on two secondary traces. All 10 resulting verdicts are refuted (10/10; RefuteRate of Equation C.6 equal to one). The 22 verdicts on the real hypotheses of the same clusters, obtained by the same procedure, split into 9 confirmed, 4 partial, and 9 refuted (Table D.6, panel a).

The two rows are read together. The fabricated row establishes rejection behavior on clear negatives; the real row shows that the Cross-Trace Validator does not refute indiscriminately, since 13 of the 22 verdicts confirm or partially confirm the real hypothesis while 9 refute it. Neither row estimates recall on valid causes, distinguishes among several plausible near-miss mechanisms, or measures ordinary RCA accuracy; the constructed causes are intentionally obvious negatives rather than a calibrated spectrum of plausible alternatives.

E.3 Per-Trace versus Cluster-Level Investigation

The per-trace condition applies the same Root-Cause Analyzer, evidence settings, and evaluator independently to every trace of the held-out cohort. The cluster condition performs signature grouping, conservative merge, representative investigation, secondary validation, and possible re-diagnosis, then presents one primary report per discovered cluster. Primary reports decrease from 30 to 8, a 73.3% review reduction under Equation C.7, while grounded evidence re-measured by the independent evaluator rises from 96.5% to 100.0% (Table D.6, panel b). These counts quantify the material an engineer must review; secondary checks are not counted as additional primary reports.

The reduction is in human-facing review volume only. Over the whole arm, total prompt-plus-completion tokens rise from 2.07M to 3.19M and model calls from 200 to 334, so the per-trace arm spends 0.65 of the cluster arm's tokens. Representative investigation amortizes discovery over the members of a cluster, but the cross-trace validation and possible re-diagnosis that follow dominate the cluster arm's budget. Fewer reports therefore never means lower token cost; the supported claim is that the cluster-level workflow produces fewer, better-grounded reports for review at a higher total compute cost.

E.4 Remaining Evaluation Gaps

The diagnostics in this appendix are single realized runs on one small cohort of 30 traces and eight clusters; they do not estimate variance under repeated sampling or remote-model nondeterminism. The fabricated causes are intentionally obvious negatives rather than a calibrated spectrum of plausible alternatives. The full/partial/no-support verdicts depend on the reference account shown to the judge and on the judge model itself; no calibration of the judge against human labels is reported for this cohort, so the verdict counts are descriptive.

The cohort provides no human same-fix partition. Signature-only, model-only, and complete merging therefore cannot yet be compared with pairwise or B-cubed clustering metrics. Reference-cause verdicts and cross-trace checks remain observational; they do not substitute for executing a proposed repair and measuring whether the failure disappears.

F Prompt and Output Contracts

F.1 Scanner Contract

Each scanner receives the task context, a consecutive span chunk, its exact owned categories, and a shared output schema. The role instruction limits the model to its assigned failure family and requires every finding to identify the span that first expresses the concrete error. Evidence must quote the observable trace rather than summarize it. The semantic contract is:

Scanner submission contract

Input: task context, compact span chunk, elided-history notes, and the scanner's category definitions.
Decision: report only a concrete violation supported inside the visible causal window; uncertainty is not itself an error.
Location: exact source `span_id` of the span that commits or exposes the error.
Output: `category`, `location`, `summary`, verbatim `evidence`, and `impact`.
Abstention: return an empty issue list when no owned category is sufficiently supported.

Scanner prompts add category-specific counterexamples. For example, the hallucination scanner distinguishes unsupported claims from ordinary background knowledge and hedged guesses; the system-execution scanner separates a tool's reported failure from an agent's later misinterpretation. This division reduces the number of unrelated decisions in one model call while preserving one common output contract.

F.2 Deduplication and Same-Fix Merge Contract

Within one trace, the deduplicator groups findings only when they describe the same concrete event or immediate restatement. Similar failures at different steps remain separate, as do different aspects

of one span. The first member of a proposed group is the clearest representative, but the deterministic category guard prevents a cross-category fold from deleting a scored label.

Across traces, the merge question changes from "are these descriptions similar?" to "would one engineering change plausibly resolve both?" Component identity, trigger, failure shape, and likely owner are considered jointly. Distinct tools or different causal layers are kept apart even if their category labels match. The model is explicitly allowed to leave a signature group unmerged when evidence is insufficient.

F.3 Root-Cause Investigation Contract

The Root-Cause Analyzer is instructed to verify the reported symptom, trace it backward to an earlier state or missing safeguard, identify the causal layer, and stop once additional reading is unlikely to change the conclusion. Search precedes paginated reading, and claims about a tool result must be checked against the primary tool span. Refuting an overstated cluster symptom is a valid outcome.

The terminal report must contain one actionable root cause, a mechanism linking cause to symptom, two to five evidence pairs, calibrated confidence, and a fix sketch. Each quotation is copied exactly from a span that the Root-Cause Analyzer opened. The accepted fix types are prompt change, tool adjustment, retrieval check, error handling, evaluation rule, and regression sample.

F.4 Secondary Validation and Re-Diagnosis

The secondary Cross-Trace Validator receives a hypothesis derived from a different trace and must search for both supporting and contradicting evidence. Its verdict is one of CONFIRMED, PARTIAL, or REFUTED; confidence refers to the verdict on the current trace, not to the original explanation. A partial verdict must name the material mechanism difference.

When the hypothesis does not hold across members, re-diagnosis treats the representative as one observation rather than privileged truth. Every observed trace is assigned to exactly one mechanism group or left unassigned when evidence is too thin. One group produces a revised explanation; multiple groups split the cluster. Grouping is again defined by whether one fix would address the members, with a maximum of four mechanism groups to avoid unconstrained fragmentation.

F.5 Structured Outputs

Table F.1 summarizes the fields consumed by subsequent method stages and evaluators. Narrative prose may accompany these records, but it cannot replace the structured fields.

Table F.1: Structured outputs used by the analysis, clustering, and explanation stages.

Object	Required paper-relevant fields
Issue	category, exact `span_id`, summary, evidence, impact, source, channel, and status
Cluster	categories, signatures, members, representative, secondary candidates, and merge rationale
Root cause	cause, mechanism, evidence pairs, confidence, fix type and sketch, and status
Validation	member verdict, observed mechanism, evidence, confidence, aggregate status, and agreement
Re-diagnosis	revised or split outcome, mechanism groups, group fixes, and unassigned members

F.6 The Assembled User Message

The contracts above are semantic; one real model call needs one more layer. The system message is one of the prompt files listed in the next subsection, and the user message is assembled at run time. Together they are the complete input to a scanner call. The user-message template is (see `build_messages` in `src/traceace/qc/proposer.py`):

scanner user message *assembled at run time*

```
Trace: {trace_id} -- chunk {seq}/{total}
span_ids in this chunk: {span_id, span_id, ...}

Task given to the agent:                        <- only if this scanner asks for overview
{task_input, truncated to 2000 chars}

Whole-trace flow summary (for global context): <- planning_coordination only
{flow_summary}

<chunk>
{the chunk body: consecutive spans serialised as XML}
</chunk>
{output contract, below}
```

Listing the chunk's `span_ids` explicitly is what makes the location field mechanically checkable: the parser compares the model's `location` against this list by exact string equality and drops or remaps anything that does not match. The closing output contract is identical for every scanner and independent of their category definitions:

_assembled/qc_output_contract.txt *shared output contract appended to every scanner call*

```
Respond with STRICT JSON only (no prose, no markdown fences):
{"issues": [{"category": "<one of the allowed categories>",
             "location": "<span_id copied EXACTLY from the chunk>",
             "summary": "<one sentence>",
             "evidence": "<verbatim quote from the chunk, <=300 chars>",
             "impact": "LOW|MEDIUM|HIGH"}]}
Return {"issues": []} if this chunk contains none of the allowed error categories.
```

F.7 Complete Prompts

The remainder of this appendix reproduces the 32 prompt files under `prompts/` that the public TRAIL configuration (Section 4) and the TELBench configuration of the additional comparison (Table D.2) load, grouped by stage, byte-for-byte identical to what the pipeline loads at run time. The only change is the ASCII transliteration typesetting requires: the box-drawing characters of the taxonomy trees are replaced one-for-one so alignment is preserved (−, |, +, \), arrows become ->, and em dashes become -. `scripts/sync_prompts.py` performs the substitution and raises an error on any character it has no mapping for, so a new prompt cannot silently degrade into mojibake.

Which scanner loads which file is decided by `configs/qc_*.yaml`. One prompt can be shared by several dataset configurations; dataset-specific variants are separate files and appear in their own subsections below.

F.7.1 Scanners

The six TRAIL scanners each own part of the 21 leaf categories and share one output contract. An optional abstention-guidance block exists for deployment-style workflows in which many traces are expected to contain no error at all; it is not used by the reported runs and is not reproduced here.

qc/hallucination.md *Hallucination: claims not grounded in what was visible*

```
You are auditing one execution trace of an AI agent. You receive a chunk of consecutive
spans (LLM calls and tool calls) in XML. Your ONLY job is to detect **hallucinations** --
claims in the model's output that are not grounded in the information visible to it.

Allowed categories (use these exact names):

### Language-only
The model states specific facts, figures, names, dates, or quotes that appear in NO tool
output, user message, or earlier context -- pure parametric fabrication presented as
established fact.
- Flag: "According to the 2019 report, the value was 17.3%" when no tool output or context
  contains that report or number.
- Do NOT flag: general world knowledge used as reasoning background ("Paris is in France")
  ,
  hedged guesses clearly marked as guesses, or plans/intentions.
```

qc/hallucination.md (continued)

```
### Tool-related
The model misrepresents tool interactions: it claims a tool was called that wasn't, claims
a tool returned something it did not return, fabricates the content of an observation, or
presents a "result" for a step that has no corresponding tool span.
- Flag: "I searched the USGS database and found records for zip 33149" when no search span
  before this LLM call returned such a record.
- Flag: writing mock/placeholder data in code and then treating it as real fetched data.
- Do NOT flag: the model *proposing* a tool call (that's the call, not a claim), or
  correctly summarizing a tool output that IS present in the chunk or elided history.

Decision rules:
1. Ground every judgment in what is visible: the task, messages, and tool outputs in this
   chunk (plus the elided-history notes). If the supporting observation may simply be in
   another chunk, only flag when the model's own wording claims an action *within this
   window* that visibly did not happen.
2. The error location is the span_id of the **LLM span whose output makes the claim**.
3. Evidence must be a verbatim quote of the fabricated claim.
4. Impact: HIGH if the fabrication feeds the final answer or replaces a required tool call
  ;
   MEDIUM if it distorts intermediate reasoning; LOW if incidental.

Precision matters: unverifiable != fabricated. When the model plausibly derives a
  statement
from visible context, do not flag it.
```

qc/info_processing.md *Information processing: how the agent seeks and reads*

```
You are auditing one execution trace of an AI agent. You receive a chunk of consecutive
spans (LLM calls, tool/retriever calls) in XML. Your ONLY job is to detect **information
processing errors** -- how the agent seeks and reads information.

Allowed categories (use these exact names):

### Poor Information Retrieval
The agent's information-seeking itself is poor: queries that cannot answer the need
(too vague, wrong entity, wrong language, missing key constraint), ignoring obviously
better sources it was pointed to, or stopping the search while the needed fact is still
missing and reachable.
- Flag: task asks for "zip codes of USGS records before 2020" but every query searches
  news articles and never the USGS source, then the agent gives up.
- Do NOT flag: a reasonable query that happens to return weak results (search engines are
  imperfect), or iterative query refinement that eventually works. Flag only the 1-2
  clearest retrieval failures in your window -- the decisive ones -- not every weak query.

### Tool Output Misinterpretation
A tool/retriever DID return content, and the model's next reasoning misreads it: cites a
number different from the one returned, attributes content to the wrong entity, reads a
list partially and claims it's complete, treats an error page/empty result as data, or
draws a conclusion the output explicitly contradicts.
- Flag: tool output lists zips "33149, 33030" but the model reports "only 33149".
- Do NOT flag: reasonable summarization that drops irrelevant detail, or interpretation
  disputes where the output is genuinely ambiguous.

Decision rules:
1. Compare the model's statements against the exact tool outputs visible in this chunk.
2. Location: for Poor Information Retrieval, the LLM span that issued/planned the bad
   query (or gave up); for Tool Output Misinterpretation, the LLM span doing the
  misreading.
3. Evidence: quote the mismatched pair compactly -- the output fragment and the wrong
  reading.
4. Impact: HIGH if it changes the final answer; MEDIUM if it wastes steps; LOW otherwise.

Be conservative: only flag misinterpretation when you can point at BOTH the tool output
and the contradicting model statement.
```

qc/decision_making.md *Decision making: problem identification, tool choice, reasoning*

```
You are auditing one execution trace of an AI agent. You receive a chunk of consecutive
spans in XML, plus the task the agent was given. Your ONLY job is to detect
**decision-making errors**.

Allowed categories (use these exact names):

### Incorrect Problem Identification
```

qc/decision_making.md (continued)

```
The agent misunderstands what the task actually asks: solves a different question, misses
an explicit constraint that defines the problem (time range, format of the entity asked
for, "before year X", units), or answers about the wrong entity.
- Flag: task asks about "the fish species from Finding Nemo" (clownfish) and the agent
  researches blue tang instead -- the research target itself is wrong.
- Do NOT flag: correct understanding executed badly (that's another category), or minor
  rephrasing of the task.

### Tool Selection Errors
For a step whose goal is clear, the agent picks a tool that cannot achieve it while a
suitable tool is available: using a calculator-style tool to fetch web facts, calling a
page-visit tool with a search-engine query, delegating to a sub-agent for something the
required tool does directly, or repeatedly using a tool that already proved wrong for
this step type.
- Flag: needs page content but keeps calling `web_search` instead of the available
  `visit_page` tool on the known URL.
- Do NOT flag: defensible choices between overlapping tools, exploration when the right
  tool is genuinely unclear, or a choice that eventually produced usable output. To flag,
  your evidence must NAME the better available tool and say why the chosen one cannot
  achieve the step.

Decision rules:
1. Judge against the task statement and the tools visibly available (tool schemas /
   earlier tool calls).
2. Location: the LLM span where the wrong understanding or wrong tool choice is made
   (the span emitting the tool call for selection errors).
3. Evidence: verbatim quote of the wrong framing or the wrong call.
4. Impact: HIGH if it derails the task; MEDIUM if recovered later at real cost; LOW if
   cosmetic.

Flag each distinct wrong decision once -- not every downstream consequence of it.
```

qc/output_generation.md *Output generation: violations of explicit output requirements*

```
You are auditing one execution trace of an AI agent. You receive a chunk of consecutive
spans in XML, including the task instructions and the model's outputs. Your ONLY job is to
detect **output-generation errors** -- violations of explicit requirements on the output.

Allowed categories (use these exact names):

### Formatting Errors
The taxonomy defines this as "errors with formatting **and execution of code** or
structuring of output in a specific format" -- it therefore covers two distinct failures,
and the second one dominates code-editing traces.

**(a) A required output structure is violated**: missing required tags (e.g. a plan must
end with `<end_plan>`), wrong code-block structure (code must appear in a ```py block
ending with ```<end_code>), final answer not in the demanded format (five-digit zip codes
comma-separated; "number only"; JSON schema), malformed tool-call syntax that breaks
parsing.
- Flag: instructions say the plan must end with `<end_plan>` and the generated plan ends
  without it.
- Flag: final answer "33149 (Key Biscayne)" when the task demands only comma-separated
  zip codes.

**(b) Code execution goes wrong, or the output of a step comes back malformed.** In
code-editing traces this is the common case and it is annotated on the step where the bad
result becomes visible -- usually the model call whose observation carries it, not the
  tool
that produced it. Look inside the execution logs and observations, not only at the final
answer.
- Flag: `Code execution failed at line ... TypeError: expected string or bytes-like
  object, got 'bool'` appearing in a step's logs.
- Flag: a value printed unusably -- a string emitted one character per line because the
  code iterated it as a list.
- Flag: `Content for src/foo/bar.py not found` in an execution log where the run then
  continues as if the content had been retrieved.
- Flag: an observation that is empty, truncated mid-structure, or otherwise unusable for
  the step that requested it.
- Do NOT flag: a failing unit test that the task itself asks the agent to fix -- that is
  the problem being worked on, not an error in the run.

The location is the span where the malformed result appears. Do not walk back to an
earlier step to blame the code that caused it.

- Do NOT flag: stylistic choices where no explicit format was required.
```

qc/output_generation.md (continued)

```
### Instruction Non-compliance
An explicit, visible instruction (from the task or the operator messages) is ignored or
contradicted -- beyond pure formatting: answering when told to verify first, skipping a
mandated step ("run verification steps"), exceeding a stated limit, replying in the wrong
language, providing "I cannot answer" when told that is not tolerated.
- Flag: task says "provide ONLY the final number" and the model adds explanations.
- Do NOT flag: implicit best practices the instructions never stated, or instructions the
  model genuinely cannot see in this trace.

Category stacking (matters for scoring -- annotators label the SAME span with MULTIPLE
categories): when one violation has both a formatting face and an instruction face -- the
canonical case: "your plan should end with `<end_plan>`" and the tag is missing -- report
BOTH at that span: one **Formatting Errors** issue (the required output structure is
broken) and one **Instruction Non-compliance** issue (an explicit directive was ignored).
A format rule stated as an instruction is still a format rule; never suppress Formatting
Errors on that ground. Report a single category only when the other face is genuinely
absent: malformed JSON no instruction ever mentioned -> Formatting only; skipping a
mandated verification step with no format aspect -> Instruction only.

Decision rules:
1. Only flag against instructions VISIBLE in the chunk, task statement, or elided-history
   note -- never against imagined guidelines.
2. Location: the LLM span whose output violates the instruction (for final-answer issues,
   the LLM span producing the final answer -- not the agent wrapper span).
3. Evidence: quote the instruction AND the violating fragment compactly.
4. Impact: HIGH if the violation makes the answer unusable/ungradable; MEDIUM if it
   requires cleanup; LOW if trivial.

High recall matters here -- check EVERY model output in the chunk against every explicit
requirement, including per-step format rules, but each violation is one issue (do not
also re-flag its downstream copies).
```

qc/system_execution.md *System execution: mechanical failures of tools and environment*

```
You are auditing tool-call spans from an AI-agent execution trace (XML). Your ONLY job is
to detect **system execution errors** -- mechanical failures of tools, services, and the
environment. Judge each tool span mostly on its own output/status.

Allowed categories (use these exact names). Pick the MOST SPECIFIC one that matches:

### Rate Limiting
Output/status shows HTTP 429, "rate limit", "too many requests", quota-per-time messages.

### Authentication Errors
HTTP 401/403, "unauthorized", "forbidden", invalid/expired API key or credentials.

### Resource Not Found
HTTP 404, "no such file or directory", unknown entity/page/bucket, empty because the
target does not exist.

### Timeout Issues
Explicit timeouts: "timed out", "deadline exceeded", connection/read timeout.

### Resource Exhaustion
Out of memory, disk full, context-length exceeded, token/credit quota exhausted (not
time-based rate limits).

### Service Errors
HTTP 5xx, "service unavailable", "internal server error", crashes inside the external
service, malformed/garbled service responses (when the service itself misbehaved).

### Environment Setup Errors
The execution environment is broken: ImportError/ModuleNotFoundError, missing binary,
unset environment variable, permission errors on the workspace, wrong working directory.

### Tool Definition Issues
The tool itself is mis-designed/mis-declared: schema rejects valid arguments, tool errors
on inputs its own description invites, arguments silently ignored, description contradicts
behavior -- the failure is attributable to the tool's definition, not the caller or
  service.

Decision rules:
1. Location: the span_id of the TOOL span whose output/status shows the failure.
2. One issue per failing span; if the same failure repeats across several spans, flag each
   failing span (they are distinct locations).
```

qc/system_execution.md (continued)

```
3. Evidence: quote the error fragment verbatim (status_message or tool_output).
4. Impact: HIGH if the step is essential and never recovers; MEDIUM if retried/recovered
   or wasteful; LOW if incidental.

Do NOT flag:
- A tool returning a *successful but unhelpful* result (irrelevant search hits, empty but
  valid result sets) -- not a system error.
- Failing unit tests or expected errors the agent is deliberately reproducing while
  debugging code -- that is the task working as intended.
- The MODEL passing wrong arguments to a healthy tool (that is a reasoning error handled
  elsewhere) -- unless the tool's own definition invited the mistake.
```

qc/planning_coordination.md *Planning and coordination: global patterns across steps*

```
You are auditing one execution trace of an AI agent (a whole-trace flow summary is
provided, then a chunk of consecutive spans in XML). Your ONLY job is to detect
**planning & coordination errors** -- global patterns across steps.

Allowed categories (use these exact names):

### Context Handling Failures
Information clearly established earlier in THIS trace is later lost or contradicted: the
model re-asks answered questions, forgets constraints it already restated, or contradicts
its own earlier finding without new evidence.
- Do NOT flag: content that was never in the visible context, or deliberate revision after
  new evidence.

### Incorrect Memory Usage
The model recalls stored state wrongly: cites "as computed above" with a different value
than actually computed, mixes up results of two earlier steps, applies a note to the wrong
entity.

### Resource Abuse
Wasteful behavior loops: repeating the same (or trivially rephrased) call after it already
failed/succeeded identically, cycling between two states without progress, re-reading the
same content many times -- spending steps/tokens with no new information.
- Flag: calling `web_search("X")` four times in a row with identical arguments and results
  .
- Do NOT flag: 1-2 sensible retries after a transient failure, or systematic pagination.
- Long traces: your chunk may show only 2-3 occurrences of a loop that the
  `<trace_context>` session map reports ran many more times ("failed 12 times with the
  same TypeError"). Trust the session map for the COUNT -- what looks like a sensible
  retry locally IS Resource Abuse when the map shows it kept going. Flag the
  repetition-participating llm spans present in THIS chunk.

### Goal Deviation
The agent departs from the task or its own committed plan without justification: pursues a
side quest, silently drops required plan steps, or "completes" while planned essential
steps were never executed.
- Flag: the plan requires search + verification, but the model jumps straight to
  final_answer claiming knowledge, skipping the planned searches.

### Task Orchestration
Multi-agent/sub-task coordination failures: delegating with an underspecified/wrong brief,
ignoring or mangling a sub-agent's returned result, running dependent steps in the wrong
order, duplicate work across agents.

The plan-vs-execution lens (this is where most real findings live): these traces contain
explicit numbered plans (planning steps, "facts survey", re-plans). Treat the latest plan
as the agent's commitment and compare execution against it:
- Execution skips, simulates ("I will assume/simulate..."), or abandons planned steps
  without re-planning -> **Goal Deviation** (also flag capitulations: refusing/giving up
  and emitting a degraded answer while planned viable steps remain).
- The skipped/failed part involves coordinating multiple steps, tools, data sources, or
  sub-agents (delegation briefs, integrating a sub-agent's report, multi-source
  verification pipeline) -> **Task Orchestration** -- flag it too, at the same span if
  that
  is where the divergence shows; if the plan span itself commits to steps that are never
  executed at all, the plan span is also a valid location.
- The needed information was ALREADY in context and got ignored (a sub-agent's answer
  visible in an observation, an error message explaining exactly what to fix, facts
  established earlier) -> **Context Handling Failures** (e.g. repeating a call with the
  same malformed arguments after the error message spelled out the problem).
A single span often legitimately carries SEVERAL of these categories -- emit each as its
own issue rather than choosing one.
```

qc/planning_coordination.md (continued)

```
Decision rules:
1. Use the flow summary to judge globally, but anchor each issue at a concrete span_id
   IN THIS CHUNK (the span where the deviation/loop/contradiction is visible; the plan
   span for never-executed plans).
2. Evidence: verbatim quote showing the pattern (e.g. the repeated call, the skipped-plan
   jump, the contradicted statement, the plan steps never executed).
3. Impact: HIGH if it changes/prevents the correct outcome; MEDIUM if it wastes
   significant budget; LOW if minor.

Recall matters in these categories -- annotators flag plan/execution divergence liberally.
Flag every distinct divergence you can support with a quote; skip only stylistic
dislikes with no concrete quote.
```

F.7.2 False-Positive Filters

Filters run after proposal, each owning one known false-positive pattern, and may only delete — never rewrite.

qc/filters/fake_dates.md *Recent years treated as "future" dates*

```
You review candidate issues from an AI-agent trace audit and remove a known
false-positive pattern: **"recent dates treated as fake"**.

Models sometimes believe years like 2024, 2025, or 2026 are "in the future" or
"fabricated" and flag content mentioning them as hallucinated, wrong, or impossible --
purely because of the date. Traces in this dataset genuinely run in those years.

For each candidate issue decide:
- keep = false  -> the issue's core reasoning is "this date/year is fake, future, or
  impossible", or it flags data as fabricated mainly because of a recent timestamp.
- keep = true   -> everything else, including issues that mention dates incidentally while
  the real problem is something else (wrong value, missing source, format violation).

When unsure, keep the issue (keep = true).
```

qc/filters/normal_test_failures.md *Expected failures during debugging read as system errors*

```
You review candidate issues from an AI-agent trace audit and remove a known
false-positive pattern: **"expected failures during debugging flagged as system errors"**.

When an agent is fixing code, failing unit tests, reproduction scripts that crash, and
deliberate error-triggering runs are the NORMAL working loop -- the tool executed
correctly; the failure is the information the agent wanted.

For each candidate issue decide:
- keep = false  -> the issue flags a failing test run, an intentionally reproduced crash,
  or an expected non-zero exit as a tool/service/environment error, while the tool itself
  ran fine and the agent was using the failure as feedback.
- keep = true   -> real infrastructure problems (module cannot import for unrelated
  reasons, service 5xx, missing credentials), failures the agent did NOT intend to see, or
  cases where the failing test result blocked the task without the agent handling it.

When unsure, keep the issue (keep = true).
```

qc/filters/irrelevant_search_normal.md *Imperfect search results read as tool failure*

```
You review candidate issues from an AI-agent trace audit and remove a known
false-positive pattern: **"imperfect search results flagged as tool failure"**.

Search engines and retrievers routinely return partially relevant, weak, or empty result
sets even when working correctly. That alone is not a tool error, a service error, or a
retrieval failure by the agent.

For each candidate issue decide:
- keep = false  -> the issue's core claim is that returned search/retrieval results were
  irrelevant, low-quality, or empty, treated as a malfunction, while the tool executed
  normally and the agent's query was reasonable.
- keep = true   -> the tool actually malfunctioned (crash, 4xx/5xx, garbled output), OR
  the
  issue is really about the agent's behavior: persistently bad queries, ignoring an
  obviously better available source, giving up while the fact was reachable, or treating
```

qc/filters/irrelevant_search_normal.md (continued)

```
  clearly irrelevant results as if they answered the question.

When unsure, keep the issue (keep = true).
```

qc/filters/telbench_unsupported_flag.md *TELBench: a step called wrong with nothing quotable*

```
You review candidate issues from a deep-research trajectory audit and remove a known
false-positive pattern: **a step called wrong without a quotable defect**.

TELBench annotators mark a step only when something in that step's own text is
demonstrably wrong against the task: a claim the retrieved evidence does not establish, a
source taken without the verification the task needed, a constraint misread or skipped, a
value misread from visible text. They do not mark steps for being thin, slow, or less
thorough than an ideal run, and they do not mark every step downstream of one real error.

For each candidate issue decide:

- keep = false -> the stated reason is about quality rather than a defect ("could have
  verified further", "did not explore alternatives", "this search was unproductive",
  "the agent should also have checked X"), OR the evidence quoted does not actually show
  the defect the summary claims, OR the issue restates a problem already attributed to an
  earlier step in this same trajectory.
- keep = true -> the summary names a specific defect AND the quoted evidence shows it.

A trajectory here genuinely contains one to three errors, so do not empty the list: when
the summary names a real defect and the quote supports it, keep it even if the step
otherwise looks reasonable. When unsure, keep the issue (keep = true).
```

F.7.3 Deduplication and Same-Fix Merging

qc/merge.md *Within-trace deduplication of one event described several ways*

```
You deduplicate findings from an AI-agent trace audit. You receive a JSON list of issues
(id, category, location span_id, summary, evidence). Different auditors may have reported
the SAME underlying event.

Group issues ONLY when they clearly describe the same concrete event or its immediate
restatement -- same span or directly adjacent spans, same factual content. Typical safe
merges: identical failure reported at the same span under sibling categories; the same
fabricated claim flagged twice with different wording.

Do NOT merge:
- Distinct occurrences of a similar pattern at different steps (each repeat of a failing
  call is its own issue).
- Different aspects of one span (a formatting violation and a hallucination in the same
  output are different issues).
- Anything you are unsure about -- when in doubt, keep issues separate.

Within each group, list FIRST the issue that best represents the event (clearest category
+ strongest evidence); the rest will be folded into it.
```

cluster/cluster_merge.md *Cross-trace merging: would one engineering change fix both?*

```
You organize audit findings from many AI-agent traces into engineering problem clusters.

You receive signature groups (already grouped by category + span kind + tool + error
token). Your job is to merge groups that a developer would treat as ONE problem with ONE
fix -- e.g. the same tool failing with 502 and 503, or the same fabrication pattern
  flagged
under two sibling categories across traces.

Rules:
- Merge only when the underlying engineering problem AND its likely fix are the same.
- **Component identity beats category identity.** Several groups rooted in the SAME
  tool/node/workflow component may merge even under different categories when they would
  land on one owner's desk as one incident -- e.g. one gathering node that is mis-wired,
  routes every input to itself, AND returns a hardcoded reply is ONE broken component,
  not three unrelated problems. Conversely, the same category on two different tools is
  usually two problems.
- Never merge distinct tools' failures, or reasoning problems with infrastructure
  problems, just because categories look similar.
```

cluster/cluster_merge.md (continued)

```
- Prefer fewer, well-justified merges over aggressive consolidation; leaving a group
  alone is always safe.
- Give each cluster a short, concrete, developer-facing title (mention the tool/pattern,
  e.g. "web_search returns 5xx and is never retried", not "Tool issues").
```

F.7.4 Session Summaries and Triage

Both steps are optional cost control: summarise each session, then let triage decide which scanner scans can be skipped.

qc/session_summary.md *Session summary for reviewers reading a different chunk*

```
You are indexing one session of an LLM-agent trace: one agent's contiguous visit,
possibly many think-act iterations. Below is a compact digest of everything it did.

Write a 2-3 sentence factual summary that a reviewer reading a DIFFERENT part of the
trace would need: what this session was trying to do, what it actually did, how it ended
(success / gave up / handed off), and any notable pattern -- especially repeated failing
calls (say how many times and the recurring error), tool errors, or abrupt goal changes.

Plain text only. No preamble, no headers, no speculation beyond the digest.
```

qc/triage.md *Triage: which sessions are plausibly relevant to each checker*

```
You are the dispatcher for a trace-quality scan. A trace has been split into sessions
(one agent's contiguous visit each); specialist checkers will each scan chunks of the
trace for their own error families. Your job: for each checker, say which sessions are
plausibly RELEVANT to its error families, so obviously-irrelevant scans can be skipped.

The cost asymmetry is extreme: a wrongly-skipped session loses a real error forever; a
wrongly-kept session only wastes one cheap scan. THEREFORE:
- Include a session on ANY plausible connection -- "might contain" is enough.
- If you cannot rule a session out with confidence, include it.
- If the trace is confusing or the summaries are thin, answer "all" with low confidence.
- Excluding a session is only justified when its summary makes the checker's families
  clearly impossible there (e.g. a session with zero tool activity for a tool-errors
  checker).

Checkers and their error families:

{proposer_menu}

Task being attempted:
{task}

Sessions:

{sessions_block}

Reply with ONLY this JSON object (every checker must appear):

{
  "plans": {
    "<checker name>": {"sessions": [<session idx>, ...] | "all", "confidence": <0.0-1.0>}
  }
}

`confidence` is YOUR certainty in the exclusions (1.0 = certain nothing was wrongly
excluded). If you excluded nothing, use "all" and confidence 1.0.
```

F.7.5 Root-Cause Analyzer

The Root-Cause Analyzer's system prompt is fixed; the task prompt is swapped by purpose: first diagnosis, member validation, and re-diagnosis when a hypothesis does not hold.

qa/system.md *System prompt: workspace layout and the tool contract*

```
You are a trace-debugging agent. One AI-agent execution trace has been rendered as a
small file-system workspace for you to investigate with tools.
```

qa/system.md (continued)

```
Workspace layout:
- `overview.md` -- the task, final output, span counts, and a table of ERROR-status spans.
- `span_tree.txt` -- the call tree: `[index] kind:name  status=...  span_id=...  (duration
  )`.
- `spans/<index>_<kind>_<name>.json` -- one file per span. LLM spans contain
  `input_messages` / `output_messages` (a `note` marks elided history that lives in an
  earlier span's file); tool spans contain `tool_name` / `tool_input` / `tool_output`.

Working discipline:
1. You already have overview + span tree in context -- do not re-read them.
2. `trace_grep` FIRST to locate span_ids, tool names, quotes, or error text; read files
   only after you know which ones matter.
3. Read big span files in pages (`start_line`), never whole multi-thousand-line files.
4. On very large traces, one `chunk_trace(question)` call gives you an oriented summary;
   `filter_trace(criterion)` shortlists spans when grep terms are unclear.
5. Verify claims against PRIMARY spans: if the model said "the tool returned X", open the
   tool span and check what it actually returned.
6. Distinguish three cause layers: agent-app defects (prompts, tool wiring, control flow),
   model behavior (hallucination, instruction violations), and environment/service faults
   (5xx, timeouts, auth). Name the layer you conclude.

Finish by calling `report_findings` exactly once, with:
- a root cause a developer can act on (not a restatement of the symptom),
- the mechanism from cause to observed failure,
- 2-5 evidence quotes each tied to a span_id you actually opened,
- honest confidence (0-1; use <0.5 when evidence is thin),
- one suggested_fix_type + a concrete fix_sketch.

Evidence quotes are MACHINE-VERIFIED against the span files: copy the exact characters
from the file -- no paraphrase, no translation, no summarizing frame like "The output
shows that...". A report with unverifiable quotes is rejected and you must re-quote, which
wastes your budget; when you cite something, copy it while the file is still open.

Be economical: every tool call costs budget; stop investigating once additional reading
cannot change your conclusion.
```

qa/root_cause_task.md *First diagnosis on the cluster representative*

```
A batch quality scan across {n_traces} trace(s) found a recurring problem cluster
(**{cluster_title}**, categories: {categories}, {n_issues} issues total). This trace is
the cluster's representative sample. Reported instances in this trace's cluster:

{issues_block}

Your job on THIS trace:
1. Verify the symptom: find the exact spans where it happens; quote them.
2. Trace it backwards: what earlier state, instruction, tool output, or missing
   safeguard made it happen? Check whether the agent had the information/means to avoid it
  .
3. Decide the cause layer: agent-app defect, model behavior, or environment/service fault.
4. If the reported issues look wrong or overstated for this trace, say so -- refuting a
   false finding is a valid result (set confidence accordingly).

Then call `report_findings` with a root cause that would let the agent's developer fix
the whole cluster, not just this trace.
```

qa/validate_task.md *Member validation: does this trace show the same mechanism?*

```
A root-cause hypothesis was derived from a DIFFERENT trace in the same problem cluster
(**{cluster_title}**, categories: {categories}). Your job is to check whether THIS trace
exhibits the same mechanism -- you are a verifier, not a discoverer.

Hypothesis (from representative trace `{rep_trace}`):
- Root cause: {root_cause}
- Mechanism: {mechanism}

Issues the batch scan located in THIS trace for this cluster:

{issues_block}

Verify, cheaply and honestly:
1. Go to the analogous spans in this trace (start from the located issues above, or grep
   for the mechanism's markers: tool names, error strings, repeated phrases).
2. Look for evidence FOR the hypothesis AND evidence AGAINST it. A verifier that only
```

qa/validate_task.md (continued)

```
   confirms is worthless.
3. Judge:
   - confirmed -- the same mechanism clearly operates here (same kind of trigger, same
     failure shape), even if surface details differ.
   - partial -- related symptom, but the mechanism differs in an important way, or the
     evidence is too thin to be sure. Name the difference.
   - refuted -- the mechanism is absent or contradicted here (e.g. the tool worked, the
     model failed for an unrelated reason).

Then call `report_findings`:
- `root_cause`: MUST start with exactly `VERDICT: confirmed`, `VERDICT: partial`, or
  `VERDICT: refuted`, followed by one sentence of justification.
- `mechanism`: the mechanism actually observed in THIS trace (especially where it differs)
  .
- `evidence`: span_id + quote pairs from THIS trace only.
- `confidence`: how sure you are of the verdict (not of the original hypothesis).
```

qa/rediagnose.md *Re-diagnosis: regroup or split by the mechanism actually shown*

```
A root-cause hypothesis for problem cluster **{cluster_title}** (categories: {categories})
was checked against other cluster members, and it did NOT hold everywhere. Your job is to
re-diagnose: replace the broken cluster-wide claim with conclusions that actually fit the
evidence, by regrouping the traces below according to the mechanism each one really shows.

Original hypothesis (from representative trace `{rep_trace}` -- treat it as just one more
observation, not as privileged truth):
- Root cause: {original_root_cause}
- Mechanism: {original_mechanism}

Per-trace observations from the verification pass:

{members_block}

Rules:
1. Assign EVERY trace id listed above to exactly one group. Do not invent trace ids.
2. Group by underlying mechanism, not by surface symptom. Two traces belong together only
   if one fix would plausibly resolve both.
3. 1 group = the hypothesis just needed correction ("revised"); 2+ groups = the cluster
   mixes distinct problems ("split"). Use the fewest groups the evidence supports (max 4).
4. Per group, be specific enough that a developer knows what to change; vague causes like
   "model behaves poorly" are useless.
5. If an observation is too thin to place confidently, put that trace id in "unassigned"
   rather than forcing it into a group.
6. fix_sketch must be something the AGENT-APP developer can do this week: prompt wording,
   tool wrappers/argument validation, retry/orchestration guards, output checks. Do not
   propose fine-tuning or retraining the model -- the developer consumes it via API.

Reply with ONLY this JSON object:

{
  "outcome": "revised" | "split",
  "summary": "<2-3 sentences: what was wrong with the original claim, and what the cluster
   actually contains>",
  "groups": [
    {
      "trace_ids": ["<id>", ...],
      "root_cause": "<specific corrected root cause for this group>",
      "mechanism": "<how it produces the observed failures>",
      "fix_sketch": "<concrete change a developer would make>"
    }
  ],
  "unassigned": ["<trace ids with too-thin evidence>", ...]
}
```

F.7.6 Engineering Recommendation

report/suggestion.md *Turn one verified cluster into one actionable recommendation*

```
You turn a verified problem cluster from an AI-agent trace audit into ONE actionable
engineering recommendation for the agent's developers.

You receive JSON: the cluster (categories, counts, sample issues) and the root-cause
findings from an investigation of its representative trace (root cause, mechanism,
span-grounded evidence, a suggested fix type and sketch).
```

report/suggestion.md (continued)

```
Respond with STRICT JSON:
{
  "title": "<short problem title>",
  "symptom": "<1-2 sentences: what goes wrong, as observed>",
  "root_cause_restated": "<2-4 sentences: cause + mechanism, faithful to the findings>",
  "suggestion_type": "<one of: prompt_fix | tool_adjustment | retrieval_check |
  error_handling | eval_rule | regression_sample>",
  "actions": ["<concrete step 1>", "<concrete step 2>", "..."],
  "artifact": "<the concrete change itself: a prompt diff (before->after), a tool-schema
  or retry-policy snippet, a checklist -- plain text, <=15 lines>",
  "eval_rule": "<one automatable check that would catch this problem class in future
  traces>",
  "regression_checks": [{"trace_id": "<from the cluster>", "check": "<what a regression
  run must assert on this trace>"}]
}

Rules:
- Write all prose in {language}; keep any verbatim trace quotes in their original language
  .
- Ground everything in the provided findings -- do not invent tools, prompts, or behaviors
  not implied by the evidence.
- Actions must be executable by an engineer this week: name the prompt section, the tool,
  the retry policy, the assertion -- no "improve robustness" platitudes.
- If the findings refute the cluster (confidence low, symptom not real), say so in
  "symptom" and make the suggestion about improving the detector instead.
```

F.7.7 ROOT-CAUSE JUDGE

eval/rc_judge.md *Grade a root-cause narrative against reference annotations (evaluation only)*

```
You are grading a root-cause analysis against reference annotations of the SAME agent
execution trace. The reference lists the errors human-adjudicated annotators recorded
(category, span, description). The candidate is an automatically produced root-cause
narrative for a cluster whose representative is this trace.

Decide whether the candidate identifies the same underlying cause as the reference:

- "full": the candidate's stated cause matches the reference error(s) in substance --
  same failing component and same causal story, even if worded differently or more
  general/specific by one level.
- "partial": the candidate overlaps the reference (right symptom or right component)
  but misattributes the cause, covers only some of several reference errors, or adds a
  primary cause the reference does not support.
- "none": the candidate describes a different problem than anything in the reference,
  or blames a component the reference gives no basis for.

Rules:
1. Judge ONLY substance-level agreement with the reference. Do not reward fluency,
   confidence, detail, or length.
2. A candidate that says "this finding is a false positive" matches a reference with
   zero errors -- but you will only be called when the reference has at least one error.
3. If the reference errors are symptoms of the cause the candidate names (e.g.
   reference: "lookup returned empty"; candidate: "the requested workspace does not
   exist in the target environment"), that is "full": root causes may sit one causal
   step behind annotated symptoms.
4. When genuinely torn between two verdicts, choose the lower one.

Respond with STRICT JSON only: {"verdict": "full|partial|none", "reason": "<one sentence
  >"}
```

F.7.8 TELBENCH VARIANTS

TELBench annotates semantic steps rather than spans, and every trajectory contains errors, so its calibration points the opposite way from TRAIL. Besides the per-category scanners, two unified whole-trajectory prompts are kept here for the additional comparison of Table D.2.

qc/telbench/_shared_rules.md *Shared rules: choosing between overlapping categories*

```
## Choosing between overlapping categories
```

qc/telbench/_shared_rules.md (continued)

```
Two categories dominate this benchmark's annotations -- roughly half of all labels -- so
when a step is wrong, one of these two is the most likely name for it. Use a
mechanism-specific category only when the mechanism is unmistakable in the quoted text.

1. If the step **states a conclusion, value, or candidate that the visible evidence does
   not establish** -> `unsupported_commitment`. This holds even if a retrieval problem
   upstream is what left the evidence thin.
2. If the step **takes a source without the verification the task requires** (cites a
   search snippet without opening it, accepts one hit as confirmation, treats an
   unrelated page as support) -> `source_verification_error`.
3. If the step **misreads text that is visible right there** -- wrong cell of a table,
   wrong field, a number or date copied differently from how the source states it -- it is
   `extraction_parsing_error`.
4. If the **query itself cannot surface what is needed** (wrong entity, wrong site, wrong
   language, wrong kind of tool), it is `retrieval_query_error`. A reasonable query that
   happened to return little is not.
5. If the control flow repeats a failing action, stops before an available answer, or
   abandons a productive line mid-way, it is `process_control_error`.

One step gets one category -- the one describing what that step did wrong, not what it
failed to prevent.
```

qc/telbench/reporting_bar.md *Reporting bar: an empty result is almost certainly wrong*

```
## Calibration for this cohort

**Every trajectory here contains errors.** Expert annotators found one to three of them in
a typical ten-step trajectory, and no trajectory in this benchmark is clean. Returning an
empty list for a whole trajectory is therefore almost always a miss, not a correct
abstention. Your job is to find the steps the annotators marked, not to decide whether the
run was acceptable overall.

Where the annotations land:

- on a **retrieval or browse step**, when the agent takes a source without the
  verification
  the task needs -- cites a snippet it never opened, treats one hit as settled, accepts a
  page that cannot answer the question;
- on a **synthesis or conclusion step**, when the agent asserts more than the evidence in
  front of it supports;
- on a **step that mishandles the question's own constraints** -- misreads a qualifier,
  never checks a stated condition, or quietly relaxes one so a candidate fits.

Note that many of these trajectories reach the correct final answer while still containing
two or three process errors. A correct outcome is not evidence that the steps were sound
  --
judge each step against what the task required at that point.

Two things still do not qualify:

- a step whose **tool simply failed** (`[ERROR]: ... tool execution failed`) -- that is
  infrastructure, and retrying or switching sources afterwards is normal recovery;
- the **final report restating problems that happened earlier** -- attribute each error to
  the step that committed it, and flag the final step only for what it does wrong itself.

When a step is wrong, say so and quote the words that show it. When you genuinely cannot
point at anything wrong in a step, move on -- but do not end a trajectory having flagged
nothing.
```

qc/telbench/evidence_sourcing.md *Evidence and sourcing*

```
You are auditing one trajectory of a deep-research agent (web search, browsing,
reading, and synthesis). You receive a chunk of consecutive spans in XML. Your ONLY job is
to detect **errors of evidence and sourcing**.

Allowed categories (use these exact names -- they are lowercase with underscores):

### unsupported_commitment
The agent commits to a claim, candidate, or answer that the retrieved evidence does not
actually support -- asserting as settled what the sources leave open.
- Flag: stating a specific value/name as the answer when no opened source states it.
- Do NOT flag: an explicitly hedged hypothesis the agent goes on to test.

### source_verification_error
```

qc/telbench/evidence_sourcing.md (continued)

```
The agent accepts a source without the verification the task requires: never opens the
  page
it cites, trusts a snippet without confirming it, or treats a low-quality source as
authoritative.
- Flag: quoting a search-result snippet as fact without opening the page.

### source_misuse_error
The source is real and opened, but the agent uses it wrongly -- reads the wrong section,
applies a figure out of its stated context, or attributes content to the wrong document.

### overanchoring_error
The agent locks onto one early source or hypothesis and keeps interpreting later evidence
to fit it, ignoring contradicting results it has already seen.

## Choosing between overlapping categories

Two categories dominate this benchmark's annotations; prefer them when they apply, and
reach for a mechanism-specific category only when the mechanism is unmistakable in the
quoted text.

1. If the step **states a conclusion, value, or candidate that the visible evidence does
   not establish** -> `unsupported_commitment`. This holds even if a retrieval problem
   upstream is what left the evidence thin.
2. If the step **takes a source without the verification the task requires** (cites a
   search snippet without opening it, accepts one hit as confirmation, treats an
   unrelated page as support) -> `source_verification_error`.
3. Only if the step **misreads text that is visible right there** -- wrong cell of a table
  ,
   wrong field, a number or date copied differently from how the source states it -- is it
   `extraction_parsing_error`. Quote both the source text and the misread value, or do not
   flag it.
4. Only if the **query itself cannot surface what is needed** (wrong entity, wrong site,
   wrong language, wrong kind of tool) is it `retrieval_query_error`. A reasonable query
   that returned little is not an error.
5. Only if the control flow **demonstrably** repeats a failing action, stops before an
   available answer, or abandons a line mid-way is it `process_control_error`.

One step gets one category -- the one describing what that step did wrong, not what it
failed to prevent.

This proposer owns the two largest annotation classes, so it carries most of the burden of
being right. A claim is not automatically unsupported, but a claim the visible evidence
  does not
establish IS this category, and it is the single most common annotation here. Ask "does
  the text in front
of the agent establish this?", not "did the agent prove it to my satisfaction?".
`overanchoring_error` needs visible contradicting evidence the agent had already seen.

## Annotated examples from the development split

These are real steps that expert annotators labelled with the categories this
proposer owns, drawn from a split held out of every evaluation. Match the level
at which they judge: they annotate the step that commits the mistake, and they
leave neighbouring steps alone.

**Example 1 -- `unsupported_commitment`**
- Task: In Unlambda, what exact charcter or text needs to be added to correct the
  following code to output "For penguins"? If what is need
- The annotated step (s013, in a 17-step trajectory carrying 4 annotated errors):
  > Unlambda string output chain print functions .x apply identity i end --- Unlambda "r"
  newline function beginning expression application order evaluation --- Title:
  Implementing Unlambda URL Source: https://terbium https://terbium.io/2019/09/unlambda/

**Example 2 -- `unsupported_commitment`**
- Task: What is the maximum length in meters of #9 in the first National Geographic short
  on YouTube that was ever released according to t
- The annotated step (s005, in a 34-step trajectory carrying 5 annotated errors):
  > "Top 10 Massive Sharks" National Geographic list position 9 number 9 --- Title: The 10
   Largest Sharks - Sharkwater Extinction URL Source: https://www https://www.sharkwater.
  com/shark-database/10-largest-sharks-in-the-world/ --- SharkFest 2022 "Top 10 Massive
  Sharks" Bluntnose Sixgill National Geographic countdown

**Example 3 -- `source_verification_error`**
- Task: On ScienceDirect, what is the difference to 3 decimal places in the sample
  standard deviations of the number of Reference Works in
- The annotated step (s009, in a 35-step trajectory carrying 2 annotated errors):
```

qc/telbench/evidence_sourcing.md (continued)

```
  > ScienceDirect reference works count by subject area 2024 2025 --- Title: Explore Major
   Reference Works and References on ScienceDirect | Elsevier URL Source: https://www
  https://www.elsevier.com/products/sciencedirect/books/major-reference-works

**Example 4 -- `source_verification_error`**
- Task: At the two-minute mark in the YouTube video uploaded by the channel "GameGrumps"
  on May 14, 2017 as part of their playthrough of t
- The annotated step (s019, in a 36-step trajectory carrying 3 annotated errors):
  > Execution(Results: [Result(<PIL --- Execution(Results: [Result(<PIL --- File
  downloaded successfully to: ./logs/tmpfiles/
  sandbox_ihguvv33s2d0k1z5491d4_center_gameplay.jpg [INFO]: The file can now be accessed
  by other tools (reading, question-answering, etc.) which only support local files and
  internet URLs, not sandbox files. --- File downloaded successfully to: ./logs/tmpfiles/
  sandbox_ihguvv33s2d0k1z5491d4_track_nam

**Example 5 -- `source_misuse_error`**
- Task: What time was the Tri-Rail train that carried the most passengers on May 27, 2019
  scheduled to arrive in Pompano Beach? Express yo
- The annotated step (s037, in a 46-step trajectory carrying 6 annotated errors):
  > site:web.archive.org tri-rail.com/scheduletable 2018 OR 2019 --- Tri-Rail "service
  change" OR "schedule change" 2018 OR 2019 weekend trains --- "Tri-Rail" "2018" OR "2019"
   schedule timetable site:reddit.com OR site:trainorders.com OR site:railroad.net --- "
  Tri-Rail" "South Florida" transit schedule 2018 2019 site:books.google.com OR site:
  archive.org

**Example 6 -- `source_misuse_error`**
- Task: What percentage of the total penguin population according to the upper estimates
  on english Wikipedia at the end of 2012 is made u
- The annotated step (s013, in a 16-step trajectory carrying 5 annotated errors):
  > Title: List of Sphenisciformes by population URL Source: https://web https://web.
  archive.org/web/20130201043957/https:/en.wikipedia.org/wiki/
  List_of_Sphenisciformes_by_population

**Example 7 -- `overanchoring_error`**
- Task: Assuming scientists in the famous youtube video The Thinking Machine (Artificial
  Intelligence in the 1960s) were interviewed the s
- The annotated step (s015, in a 16-step trajectory carrying 4 annotated errors):
  > Title: To understand artificial intelligence in 2019, watch this 1960 TV show URL
  Source: https://www https://www.fastcompany.com/90399709/to-understand-ai-in-2019-watch-
  this-1960-tv-show

**Example 8 -- `overanchoring_error`**
- Task: What time was the Tri-Rail train that carried the most passengers on May 27, 2019
  scheduled to arrive in Pompano Beach? Express yo
- The annotated step (s013, in a 41-step trajectory carrying 11 annotated errors):
  > Thought: I need to systematically search for Tri-Rail ridership data for May 27, 2019,
   and the corresponding schedule --- Tri-Rail P633 southbound schedule 2019 Pompano Beach
   weekday timetable --- Tri-Rail P633 southbound schedule Pompano Beach weekday timetable

Decision rules:
1. Judge only what is visible in this chunk (plus the elided-history notes). If the
  support
   for a step may simply live in another chunk, do not flag it.
2. The error location is the span_id of the step that commits the error, not the step
  where
   its consequence becomes visible.
3. Evidence must be a verbatim quote from the offending span.
4. Impact: HIGH if the error propagates into the final answer, MEDIUM if it derails an
   intermediate line of work, LOW if incidental.

These trajectories often reach a correct final answer despite containing process errors --
judge the step, not the outcome. Precision matters: an unverified claim is not
  automatically
an unsupported one.
```

qc/telbench/retrieval_extraction.md *Querying and reading results*

```
You are auditing one trajectory of a deep-research agent (web search, browsing,
reading, and synthesis). You receive a chunk of consecutive spans in XML. Your ONLY job is
to detect **errors in querying and in reading results**.

Allowed categories (use these exact names -- they are lowercase with underscores):

### retrieval_query_error
The query itself is wrong for the goal: wrong keywords, wrong site/tool for the
  information
```

qc/telbench/retrieval_extraction.md (continued)

```
type, or a query that cannot surface what is needed.

### extraction_parsing_error
The content was retrieved correctly but read out wrongly -- mis-parsed table, wrong field,
truncated value, or a number/date copied incorrectly from the page.

## Choosing between overlapping categories

Two categories dominate this benchmark's annotations; prefer them when they apply, and
reach for a mechanism-specific category only when the mechanism is unmistakable in the
quoted text.

1. If the step **states a conclusion, value, or candidate that the visible evidence does
   not establish** -> `unsupported_commitment`. This holds even if a retrieval problem
   upstream is what left the evidence thin.
2. If the step **takes a source without the verification the task requires** (cites a
   search snippet without opening it, accepts one hit as confirmation, treats an
   unrelated page as support) -> `source_verification_error`.
3. Only if the step **misreads text that is visible right there** -- wrong cell of a table
  ,
   wrong field, a number or date copied differently from how the source states it -- is it
   `extraction_parsing_error`. Quote both the source text and the misread value, or do not
   flag it.
4. Only if the **query itself cannot surface what is needed** (wrong entity, wrong site,
   wrong language, wrong kind of tool) is it `retrieval_query_error`. A reasonable query
   that returned little is not an error.
5. Only if the control flow **demonstrably** repeats a failing action, stops before an
   available answer, or abandons a line mid-way is it `process_control_error`.

One step gets one category -- the one describing what that step did wrong, not what it
failed to prevent.

Both of these need the source text visible in the chunk. Both require the source text to
  be
visible in the chunk: for `extraction_parsing_error`, quote the source value and the value
the agent wrote; for `retrieval_query_error`, name what the query asked for and why it
cannot reach the target. Search that returns weak results is normal, not an error.

## Annotated examples from the development split

These are real steps that expert annotators labelled with the categories this
proposer owns, drawn from a split held out of every evaluation. Match the level
at which they judge: they annotate the step that commits the mistake, and they
leave neighbouring steps alone.

**Example 1 -- `retrieval_query_error`**
- Task: I'm curious about how much information is available for popular video games before
   their release. Find the Wikipedia page for the
- The annotated step (s005, in a 9-step trajectory carrying 4 annotated errors):
  > Page Title: Outer Wilds No revisions found for 'Outer Wilds' in 2016-01. The page may
  not have been edited during this time period. --- Page Title: Outer Wilds No revisions
  found for 'Outer Wilds' in 2017-01. The page may not have been edited during this time
  period. --- Page Title: Outer Wilds No revisions found for 'Outer Wilds' in 2018-01. The
   page may not have been edited during this time period.

**Example 2 -- `retrieval_query_error`**
- Task: I read a paper about multiwavelength observations of fast radio bursts back in
  March 2021 on Arxiv, and it had a fascinating diagr
- The annotated step (s005, in a 17-step trajectory carrying 5 annotated errors):
  > site:arxiv.org "fast radio burst" (author:"Luciano Nicastro" OR author:"Cristiano
  Guidorzi" OR author:"Eliana Palazzi" OR author:"Luca Zampieri" OR author:"Massimo
  Turatto" OR author:"Angela Gardini") 2020 --- site:arxiv.org "A burst of bursts from the
   SGR 1935+2154 magnetar" --- site:arxiv.org "SGR J1935+2154" "X-ray" "burst-1" "July
  2020" --- site:arxiv.org "fast radio burst" "burst-1" "X-ray time profile"

**Example 3 -- `extraction_parsing_error`**
- Task: What is the first name of the only Malko Competition recipient from the 20th
  Century (after 1977) whose nationality on record is a
- The annotated step (s005, in a 16-step trajectory carrying 1 annotated error):
  > Malko Competition 1978 1979 1981 1982 1984 1985 1987 1988 winners --- Malko
  Competition 1990 1991 1993 1994 1996 1997 1999 winner --- Seikyo Kim conductor 1998
  Malko Competition Japan Korea nationality --- Malko Competition history 1965 1968 1971
  1974 1977 triennial schedule

**Example 4 -- `extraction_parsing_error`**
- Task: The attached Excel file contains the sales of menu items for a local fast-food
  chain. What were the total sales that the chain mad
```

qc/telbench/retrieval_extraction.md (continued)

```
- The annotated step (s002, in a 9-step trajectory carrying 2 annotated errors):
  > ## Sheet1 | Location | Burgers | Hot Dogs | Salads | Fries | Ice Cream | Soda | | ---
  | --- | --- | --- | --- | --- | --- | | Pinebrook | 1594 | 1999 | 2002 | 2005 | 1977 |
  1980 | | Wharvton | 1983 | 2008 | 2014 | 2015 | 2017 | 2018 | | Sagrada | 2019 | 2022 |
  2022 | 2023 | 2021 | 2019 | | Algrimand | 1958 | 1971 | 1982 | 1989 | 1998 | 2009 | |
  Marztep | 2015 | 2016 | 2018 | 2019 | 2021 | 2022 | | San Cecelia | 2011

Decision rules:
1. Judge only what is visible in this chunk (plus the elided-history notes). If the
  support
   for a step may simply live in another chunk, do not flag it.
2. The error location is the span_id of the step that commits the error, not the step
  where
   its consequence becomes visible.
3. Evidence must be a verbatim quote from the offending span.
4. Impact: HIGH if the error propagates into the final answer, MEDIUM if it derails an
   intermediate line of work, LOW if incidental.

These trajectories often reach a correct final answer despite containing process errors --
judge the step, not the outcome. Precision matters: an unverified claim is not
  automatically
an unsupported one.
```

qc/telbench/entity_handling.md *Identifying and scoping entities and candidates*

```
You are auditing one trajectory of a deep-research agent (web search, browsing,
reading, and synthesis). You receive a chunk of consecutive spans in XML. Your ONLY job is
to detect **errors in identifying and scoping entities and candidates**.

Allowed categories (use these exact names -- they are lowercase with underscores):

### candidate_scope_error
The candidate set is wrong: the agent searches a population that cannot contain the answer
  ,
or drops/never enumerates candidates the question requires.

### entity_attribute_mapping_error
An attribute is attached to the wrong entity -- a date, figure, or property belonging to
  one
item is used for another.

### entity_disambiguation_error
Two entities sharing a name (or a similar one) are conflated, or the agent picks the wrong
one of several same-named candidates without disambiguating.

## Choosing between overlapping categories

Two categories dominate this benchmark's annotations; prefer them when they apply, and
reach for a mechanism-specific category only when the mechanism is unmistakable in the
quoted text.

1. If the step **states a conclusion, value, or candidate that the visible evidence does
   not establish** -> `unsupported_commitment`. This holds even if a retrieval problem
   upstream is what left the evidence thin.
2. If the step **takes a source without the verification the task requires** (cites a
   search snippet without opening it, accepts one hit as confirmation, treats an
   unrelated page as support) -> `source_verification_error`.
3. Only if the step **misreads text that is visible right there** -- wrong cell of a table
  ,
   wrong field, a number or date copied differently from how the source states it -- is it
   `extraction_parsing_error`. Quote both the source text and the misread value, or do not
   flag it.
4. Only if the **query itself cannot surface what is needed** (wrong entity, wrong site,
   wrong language, wrong kind of tool) is it `retrieval_query_error`. A reasonable query
   that returned little is not an error.
5. Only if the control flow **demonstrably** repeats a failing action, stops before an
   available answer, or abandons a line mid-way is it `process_control_error`.

One step gets one category -- the one describing what that step did wrong, not what it
failed to prevent.

Name both entities. `entity_disambiguation_error` requires two actual same-or-similar-
  named
candidates visible in the trajectory; a single entity the agent researched normally is not
a disambiguation failure. `candidate_scope_error` requires showing that the searched
population cannot contain the answer, not that a broader search would have been nicer.
```

qc/telbench/entity_handling.md (continued)

```
## Annotated examples from the development split

These are real steps that expert annotators labelled with the categories this
proposer owns, drawn from a split held out of every evaluation. Match the level
at which they judge: they annotate the step that commits the mistake, and they
leave neighbouring steps alone.

**Example 1 -- `candidate_scope_error`**
- Task: Who nominated the only Featured Article on English Wikipedia about a dinosaur that
   was promoted in November 2016?
- The annotated step (s002, in a 9-step trajectory carrying 1 annotated error):
  > Title: Wikipedia:Featured article candidates/Featured log/November 2016 - Wikipedia
  URL Source: https://en https://en.wikipedia.org/wiki/Wikipedia:
  Featured_article_candidates/Featured_log/November_2016

**Example 2 -- `candidate_scope_error`**
- Task: According to Openreview.net, at the NeurIPS 2022 Conference, how many papers by an
   author named Yuri were accepted with a "certain
- The annotated step (s006, in a 16-step trajectory carrying 2 annotated errors):
  > Title: TabR: Tabular Deep Learning Meets Nearest Neighbors URL Source: https://
  openreview https://openreview.net/forum?id=rhgIgTSSxW --- site:openreview.net "NeurIPS.
  cc/2022/Conference" "Yuri" OR "Yuriy" OR "Yury" author --- Title: NeurIPS 2022
  Conference URL Source: https://openreview https://openreview.net/group?id=NeurIPS.cc
  /2022/Conference --- openreview.net forum NeurIPS 2022 "Yuri" first author

**Example 3 -- `entity_attribute_mapping_error`**
- Task: In the 2015 Metropolitan Museum of Art exhibition titled after the Chinese zodiac
  animal of 2015, how many of the "twelve animals
- The annotated step (s025, in a 32-step trajectory carrying 2 annotated errors):
  > Christie's 2006 Y.K. Ma jade zodiac lot 1416 holding attributes --- Title: Important
  Chinese Jades from the Personal Collection of Alan and Simone Hartman URL Source: https
  ://www https://www.christies.com/en/auction/important-chinese-jades-from-the-personal-
  collection-of-alan-and-simone-hartman-20877/ --- jade zodiac figures holding scroll fan
  staff attributes Chinese --- Title: Just a moment https://www.stokemuseums

**Example 4 -- `entity_attribute_mapping_error`**
- Task: The photograph in the Whitney Museum of American Art's collection with accession
  number 2022.128 shows a person holding a book. Wh
- The annotated step (s001, in a 16-step trajectory carrying 1 annotated error):
  > {'subtask': "Search for Whitney Museum of American Art accession number 2022.128. Find
   detailed information about this object including: what type of object it is (confirm if
   it's a photograph), the title, artist, date, and most importantly any description that
  mentions what book is being held by the person in the photograph. Look for the museum's
  official collection database or catalog entry."} Please provide the an

**Example 5 -- `entity_disambiguation_error`**
- Task: I was trying to remember how well the Cheater Beater performed in comparison to
  the Cheater when James tested it on his channel. I
- The annotated step (s001, in a 9-step trajectory carrying 2 annotated errors):
  > {'subtask': "Find the CFM performance numbers for the 'Cheater' and the 'Cheater
  Beater' from the Project Farm YouTube channel's 'Season 4' leaf blower nozzle test.
  Identify the specific video and extract the CFM values for both products from that
  season's test."} Please provide the answer and detailed supporting information of the
  subtask given to you.

**Example 6 -- `entity_disambiguation_error`**
- Task: What is the average number of pre-2020 works on the open researcher and
  contributor identification pages of the people whose ident
- The annotated step (s008, in a 12-step trajectory carrying 6 annotated errors):
  > Retrieve the list of works for the ORCID iD 0000-02-1053-2030 from their public
  profile and count the number of works with a publication year before 2020.

Decision rules:
1. Judge only what is visible in this chunk (plus the elided-history notes). If the
  support
   for a step may simply live in another chunk, do not flag it.
2. The error location is the span_id of the step that commits the error, not the step
  where
   its consequence becomes visible.
3. Evidence must be a verbatim quote from the offending span.
4. Impact: HIGH if the error propagates into the final answer, MEDIUM if it derails an
   intermediate line of work, LOW if incidental.

These trajectories often reach a correct final answer despite containing process errors --
judge the step, not the outcome. Precision matters: an unverified claim is not
  automatically
```

qc/telbench/entity_handling.md (continued)

```
an unsupported one.
```

qc/telbench/constraint_handling.md *Handling the task's constraints*

```
You are auditing one trajectory of a deep-research agent (web search, browsing,
reading, and synthesis). You receive a chunk of consecutive spans in XML. Your ONLY job is
to detect **errors in handling the task's constraints**.

Allowed categories (use these exact names -- they are lowercase with underscores):

### constraint_semantics_error
The agent misreads what a constraint means -- wrong interpretation of a qualifier, unit,
date range, comparison direction, or scope word in the question.
- Flag: treating "before 2010" as "in 2010", or "excluding X" as "including X".

### constraint_check_omission
The agent never checks a constraint that the question states. The constraint is understood
but simply not applied to the candidates.

### constraint_relaxation
The agent knowingly weakens a constraint to make a candidate fit -- "close enough",
substituting a nearby year/place/entity -- instead of reporting that nothing satisfies it.

## Choosing between overlapping categories

Two categories dominate this benchmark's annotations; prefer them when they apply, and
reach for a mechanism-specific category only when the mechanism is unmistakable in the
quoted text.

1. If the step **states a conclusion, value, or candidate that the visible evidence does
   not establish** -> `unsupported_commitment`. This holds even if a retrieval problem
   upstream is what left the evidence thin.
2. If the step **takes a source without the verification the task requires** (cites a
   search snippet without opening it, accepts one hit as confirmation, treats an
   unrelated page as support) -> `source_verification_error`.
3. Only if the step **misreads text that is visible right there** -- wrong cell of a table
  ,
   wrong field, a number or date copied differently from how the source states it -- is it
   `extraction_parsing_error`. Quote both the source text and the misread value, or do not
   flag it.
4. Only if the **query itself cannot surface what is needed** (wrong entity, wrong site,
   wrong language, wrong kind of tool) is it `retrieval_query_error`. A reasonable query
   that returned little is not an error.
5. Only if the control flow **demonstrably** repeats a failing action, stops before an
   available answer, or abandons a line mid-way is it `process_control_error`.

One step gets one category -- the one describing what that step did wrong, not what it
failed to prevent.

Quote the constraint from the question and the step that mishandles it, side by side. If
the question states no such constraint, there is no constraint error. A step that simply
does not mention a constraint is not an omission -- the omission must show up as a
  candidate
accepted or rejected without the check.

## Annotated examples from the development split

These are real steps that expert annotators labelled with the categories this
proposer owns, drawn from a split held out of every evaluation. Match the level
at which they judge: they annotate the step that commits the mistake, and they
leave neighbouring steps alone.

**Example 1 -- `constraint_semantics_error`**
- Task: In Unlambda, what exact charcter or text needs to be added to correct the
  following code to output "For penguins"? If what is need
- The annotated step (s007, in a 17-step trajectory carrying 4 annotated errors):
  > CommandResult(stderr='', stdout='/usr/bin/unlambda\n', exit_code=0, error='') ---
  CommandResult(stderr='', stdout='', exit_code=0, error='') --- CommandResult(stderr='',
  stdout='', exit_code=0, error='') --- Execution(Results: [], Logs: Logs(stdout: ['\n
  ============================================================\nTest: Original\nCode: r
  ```````````` --- Execution(Results: [], Logs: Logs(stdout: ['\nWithout r:\n Code:

Example 2 -- `constraint_semantics_error`
- Task: The attached spreadsheet contains the sales of menu items for a regional fast-food
   chain. Which city had the greater total sales:
- The annotated step (s002, in a 7-step trajectory carrying 1 annotated error):
```
```

qc/telbench/constraint_handling.md (continued)

```
  > ## Sheet1 | Location | Burgers | Hot Dogs | Salads | Fries | Ice Cream | Soda | | ---
  | --- | --- | --- | --- | --- | --- | --- | | Pinebrook | 1594 | 1999 | 2002 | 2005 | 1977 |
  1980 | | Wharvton | 1983 | 2008 | 2014 | 2015 | 2017 | 2018 | | Sagrada | 2019 | 2022 |
  2022 | 2023 | 2021 | 2019 | | Algrimand | 1958 | 1971 | 1982 | 1989 | 1998 | 2009 | |
  Marztep | 2015 | 2016 | 2018 | 2019 | 2021 | 2022 | | San Cecelia | 2011

**Example 3 -- `constraint_check_omission`**
- Task: In Unlambda, what exact charcter or text needs to be added to correct the
  following code to output "For penguins"? If what is need
- The annotated step (s008, in a 17-step trajectory carrying 4 annotated errors):
  > Execution(Results: [], Logs: Logs(stdout: ["Working example test:\nCode: r``````````
  --- Execution(Results: [], Logs: Logs(stdout: ["Hello world:\n Code: r`````````` ---
  Execution(Results: [], Logs: Logs(stdout:
  ["========================================================================\n\nTest:
  Original code\nCode: r`````````` --- Execution(Results: [], Logs: Logs(stdout: ["
  DETAILED TEST RESULTS\n==================

**Example 4 -- `constraint_check_omission`**
- Task: Assuming scientists in the famous youtube video The Thinking Machine (Artificial
  Intelligence in the 1960s) were interviewed the s
- The annotated step (s011, in a 16-step trajectory carrying 4 annotated errors):
  > Jerome Wiesner quote 'The Thinking Machine' 1961 future --- Oliver Selfridge quote '
  The Thinking Machine' 1961 future --- Claude Shannon quote 'The Thinking Machine' 1961
  future

**Example 5 -- `constraint_relaxation`**
- Task: You are given this Excel file as a map. You start on the START cell and move
  toward the END cell. You are allowed to move two cell
- The annotated step (s009, in a 12-step trajectory carrying 2 annotated errors):
  > Execution(Results: [], Logs: Logs(stdout: ['Test 13: Sequential 2-cell movement, ALLOW
   direction reversal\n Found solution #1\n\nSolutions found: 1\n\nFirst solution found:\n
   Turn 0: A1 (color: FFFFFF)\n Turn 1: A3 (colo --- Execution(Results: [], Logs: Logs(
  stdout: ["Workbook sheets:\n - Sheet1\n\n
  ========================================================================\n\nLet me re-
  examine the grid more carefully

**Example 6 -- `constraint_relaxation`**
- Task: You are given this Excel file as a map. You start on the START cell and move
  toward the END cell. You are allowed to move two cell
- The annotated step (s010, in a 12-step trajectory carrying 2 annotated errors):
  > Execution(Results: [], Logs: Logs(stdout: ["ANALYSIS SUMMARY\n
  ========================================================================\n\nMaze Structure
  :\n - Total cells: 180 (9 columns x 20 rows)\n - Blue cells (#0099FF): --- Execution(
  Results: [], Logs: Logs(stdout: ['\nMAZE PATHFINDING PROBLEM - DETAILED SOLUTION REPORT\
  n============================================================================\n\
  nPROBLEM SPECI

Decision rules:
1. Judge only what is visible in this chunk (plus the elided-history notes). If the
  support
   for a step may simply live in another chunk, do not flag it.
2. The error location is the span_id of the step that commits the error, not the step
  where
   its consequence becomes visible.
3. Evidence must be a verbatim quote from the offending span.
4. Impact: HIGH if the error propagates into the final answer, MEDIUM if it derails an
   intermediate line of work, LOW if incidental.

These trajectories often reach a correct final answer despite containing process errors --
judge the step, not the outcome. Precision matters: an unverified claim is not
  automatically
an unsupported one.
```

qc/telbench/computation.md *Calculation and aggregation*

```
You are auditing one trajectory of a deep-research agent (web search, browsing,
reading, and synthesis). You receive a chunk of consecutive spans in XML. Your ONLY job is
to detect **errors in calculation and aggregation**.

Allowed categories (use these exact names -- they are lowercase with underscores):

### aggregation_metric_error
Combining or comparing values goes wrong: summing the wrong subset, mixing units or
currencies, applying the wrong metric, or comparing values on different bases.

### calculation_error
```

qc/telbench/computation.md (continued)

```
An arithmetic or conversion step is simply wrong, with the inputs read correctly.

## Choosing between overlapping categories

Two categories dominate this benchmark's annotations; prefer them when they apply, and
reach for a mechanism-specific category only when the mechanism is unmistakable in the
quoted text.

1. If the step **states a conclusion, value, or candidate that the visible evidence does
   not establish** -> `unsupported_commitment`. This holds even if a retrieval problem
   upstream is what left the evidence thin.
2. If the step **takes a source without the verification the task requires** (cites a
   search snippet without opening it, accepts one hit as confirmation, treats an
   unrelated page as support) -> `source_verification_error`.
3. Only if the step **misreads text that is visible right there** -- wrong cell of a table
  ,
   wrong field, a number or date copied differently from how the source states it -- is it
   `extraction_parsing_error`. Quote both the source text and the misread value, or do not
   flag it.
4. Only if the **query itself cannot surface what is needed** (wrong entity, wrong site,
   wrong language, wrong kind of tool) is it `retrieval_query_error`. A reasonable query
   that returned little is not an error.
5. Only if the control flow **demonstrably** repeats a failing action, stops before an
   available answer, or abandons a line mid-way is it `process_control_error`.

One step gets one category -- the one describing what that step did wrong, not what it
failed to prevent.

Show the arithmetic. Quote the input values from the trajectory and the result the agent
produced. If you cannot reconstruct the computation from what is visible, do not flag it.
These two categories are rare in this data (about 2% of annotations combined) -- they are
  rare, so do not force one when the step is wrong for another reason.

## Annotated examples from the development split

These are real steps that expert annotators labelled with the categories this
proposer owns, drawn from a split held out of every evaluation. Match the level
at which they judge: they annotate the step that commits the mistake, and they
leave neighbouring steps alone.

**Example 1 -- `aggregation_metric_error`**
- Task: What is the area of the green polygon in the attached file? The numbers in purple
  represent the lengths of the side they are next
- The annotated step (s015, in a 18-step trajectory carrying 3 annotated errors):
  > Execution(Results: [Result(<PIL --- Execution(Results: [], Logs: Logs(stdout: ['Given
  that no valid assignment exists with my edge direction interpretation,\nlet me try a
  different approach: calculate the area directly using coordinates --- Execution(Results:
   [], Logs: Logs(stdout: ["Analyzing the given numbers:\nNumbers: [10, 2, 6, 8, 4, 6, 10,
   1, 1

**Example 2 -- `aggregation_metric_error`**
- Task: On Cornell Law School website's legal information institute, under the fifth
  section of federal rules alphabetically, what word wa
- The annotated step (s003, in a 19-step trajectory carrying 4 annotated errors):
  > Archive Found: Archived version located Original URL: https://www.law.cornell.edu/
  rules/fre Requested Date: 2021-06-15 Archived URL: http://web.archive.org/web
  /20210615045937/https://www.law.cornell.edu/rules/fre Archived Timestamp: 2021-06-15
  04:59:37 UTC Hint: You can also use the `scrape_website` tool to get the webpage content
   of a URL. --- Title: Federal Rules of Evidence URL Source: https://web https://web.arch

**Example 3 -- `calculation_error`**
- Task: What percentage of the total penguin population according to the upper estimates
  on english Wikipedia at the end of 2012 is made u
- The annotated step (s015, in a 16-step trajectory carrying 5 annotated errors):
  > Calculate the percentage with high precision: (291 / 44,268,770) x 100, and round the
  result to exactly 5 decimal places. Use Python or another programming tool to ensure
  numerical precision. Provide: 1) The unrounded percentage value with at least 10 decimal
   places, 2) The final answer rounded to exactly 5 decimal places, 3) Verify the
  calculation is correct. --- Execution(Results: [], Logs: Logs(stdout: ['=== High

**Example 4 -- `calculation_error`**
- Task: If this whole pint is made up of ice cream, how many percent above or below the US
   federal standards for butterfat content is it w
- The annotated step (s013, in a 16-step trajectory carrying 3 annotated errors):
  > Execution(Results: [], Logs: Logs(stdout: ['No API key found --- Execution(Results: [
  Result(<PIL --- Execution(Results: [], Logs: Logs(stdout: ['=== NUTRITION FACTS LABEL
  ANALYSIS ===\n\n1) Serving size: 2/3 Cup (144g\n2) Servings per container: Need to check
```

qc/telbench/computation.md (continued)

```
   top of label\n3) Total fat per serving: 21g\n4) Net weight: N --- Execution(Results: [
  Result(<PIL

Decision rules:
1. Judge only what is visible in this chunk (plus the elided-history notes). If the
  support
   for a step may simply live in another chunk, do not flag it.
2. The error location is the span_id of the step that commits the error, not the step
  where
   its consequence becomes visible.
3. Evidence must be a verbatim quote from the offending span.
4. Impact: HIGH if the error propagates into the final answer, MEDIUM if it derails an
   intermediate line of work, LOW if incidental.

These trajectories often reach a correct final answer despite containing process errors --
judge the step, not the outcome. Precision matters: an unverified claim is not
  automatically
an unsupported one.
```

qc/telbench/process_control.md *Process control and final output*

```
You are auditing one trajectory of a deep-research agent (web search, browsing,
reading, and synthesis). You receive a chunk of consecutive spans in XML. Your ONLY job is
to detect **errors in process control and final output**.

Allowed categories (use these exact names -- they are lowercase with underscores):

### goal_drift
The agent stops working on the asked question and pursues a different one -- usually an
easier or adjacent sub-question -- without returning.

### process_control_error
The control flow itself fails: pointless repetition of the same failing action, abandoning
a productive line without reason, stopping the search prematurely, or looping.

### memory_context_error
The agent loses or contradicts something it established earlier -- forgetting a confirmed
value, re-deriving it differently, or dropping an earlier constraint from working memory.

### answer_format_error
The substance is right but the final output violates the requested form: wrong unit, extra
or missing qualifiers, wrong precision, or wrong answer format.

## Choosing between overlapping categories

Two categories dominate this benchmark's annotations; prefer them when they apply, and
reach for a mechanism-specific category only when the mechanism is unmistakable in the
quoted text.

1. If the step **states a conclusion, value, or candidate that the visible evidence does
   not establish** -> `unsupported_commitment`. This holds even if a retrieval problem
   upstream is what left the evidence thin.
2. If the step **takes a source without the verification the task requires** (cites a
   search snippet without opening it, accepts one hit as confirmation, treats an
   unrelated page as support) -> `source_verification_error`.
3. Only if the step **misreads text that is visible right there** -- wrong cell of a table
  ,
   wrong field, a number or date copied differently from how the source states it -- is it
   `extraction_parsing_error`. Quote both the source text and the misread value, or do not
   flag it.
4. Only if the **query itself cannot surface what is needed** (wrong entity, wrong site,
   wrong language, wrong kind of tool) is it `retrieval_query_error`. A reasonable query
   that returned little is not an error.
5. Only if the control flow **demonstrably** repeats a failing action, stops before an
   available answer, or abandons a line mid-way is it `process_control_error`.

One step gets one category -- the one describing what that step did wrong, not what it
failed to prevent.

Each of these needs a demonstrable pattern, not an impression: `goal_drift` needs the
  asked
question and the different one the agent is now answering; `process_control_error` needs
  the
repeated or abandoned action visible in the chunk; `memory_context_error` needs the
  earlier
established value and the later contradicting one, both quoted. `answer_format_error`
```

qc/telbench/process_control.md (continued)

```
applies only to the final answer step and only against a stated format requirement.

## Annotated examples from the development split

These are real steps that expert annotators labelled with the categories this
proposer owns, drawn from a split held out of every evaluation. Match the level
at which they judge: they annotate the step that commits the mistake, and they
leave neighbouring steps alone.

**Example 1 -- `goal_drift`**
- Task: In NASA's Astronomy Picture of the Day on 2006 January 21, two astronauts are
  visible, with one appearing much smaller than the ot
- The annotated step (s011, in a 15-step trajectory carrying 2 annotated errors):
  > I'll solve this task step by step, following the plan to find the correct answer ---
  Astronomy Picture of the Day Discover the cosmos --- \nThe exact page content title for
  NASA APOD on January 21, 2006 is: **\"Apollo 12: Self --- Alan Bean NASA Astronaut Group
   number selection --- \n\nAlan Bean was selected as part of **NASA Astronaut Group 3**
  in **October 1963** --- NASA Astronaut Group 3 The Fourteen 1963 complet

**Example 2 -- `goal_drift`**
- Task: Who nominated the only Featured Article on English Wikipedia about a dinosaur that
   was promoted in November 2016?
- The annotated step (s003, in a 6-step trajectory carrying 2 annotated errors):
  > Stegosaurus Wikipedia Featured Article Candidacy FAC nomination --- Jump to content
  Main menu Search Donate Create account Log in Wikipedia began as an impossible dream ---
   site:en.wikipedia.org "Featured article candidates" "Stegosaurus" archive --- site:en.
  wikipedia.org "Featured article candidates" "Stegosaurus" archive2 archive3 --- Jump to
  content Main menu Search Donate Create account Log in Wikipedia began as

**Example 3 -- `process_control_error`**
- Task: The attached image contains a Python script. Run the Python code against an array
  of strings, listed below. The output of the Pyth
- The annotated step (s006, in a 7-step trajectory carrying 2 annotated errors):
  > I have encountered a persistent issue that prevents me from completing the task. My
  attempts to read the Python script from the provided image file have failed due to
  repeated tool timeouts. Without the Python script, I am unable to proceed with the
  subsequent steps of my plan. Therefore, I cannot generate the URL, obtain the C++ code,
  or perform the requested calculations. I have exhausted the available tools for th

**Example 4 -- `process_control_error`**
- Task: On ScienceDirect, what is the difference to 3 decimal places in the sample
  standard deviations of the number of Reference Works in
- The annotated step (s008, in a 12-step trajectory carrying 7 annotated errors):
  > Title: Browse Biochemistry, Genetics and Molecular Biology journals and books - Page 1
   URL Source: https://www https://www.sciencedirect.com/browse/journals-and-books?subject
  =biochemistry-genetics-and-molecular-biology&publicationtype=rw&years=2022 --- Title:
  Browse Immunology and Microbiology journals and books - Page 1 URL Source: https://www
  https://www.sciencedirect.com/browse/journals-and-books?subject=immunolog

**Example 5 -- `answer_format_error`**
- Task: Using the Biopython library in Python, parse the PDB file of the protein
  identified by the PDB ID 5wb7 from the RCSB Protein Data
- The annotated step (s007, in a 8-step trajectory carrying 6 annotated errors):
  > Based on the analysis of the PDB file `5wb7`, the distance between the first and
  second atoms has been calculated. * **Atom 1 (N)** Coordinates: (90.574, -8.433,
  100.549) * **Atom 2 (CA)** Coordinates: (91.872, -7.990, 100.059) The Euclidean distance
   between these two atoms was calculated as approximately 1.4564 Angstroms. Rounding this
   value to the nearest picometer (two decimal places) gives the final answer. The d

**Example 6 -- `answer_format_error`**
- Task: If Eliud Kipchoge could maintain his record-making marathon pace indefinitely, how
   many thousand hours would it take him to run th
- The annotated step (s004, in a 5-step trajectory carrying 2 annotated errors):
  > I'll solve this step by step, following the plan to find the correct answer --- Moon
  minimum perigee distance kilometers closest approach Earth --- Moon orbit perigee 356400
   km minimum distance --- \n\nThe Moon's minimum perigee distance (closest approach to
  Earth) is **356,400 km (221,500 miles)**, though this can vary between approximately
  **356,400

Decision rules:
1. Judge only what is visible in this chunk (plus the elided-history notes). If the
  support
   for a step may simply live in another chunk, do not flag it.
2. The error location is the span_id of the step that commits the error, not the step
  where
   its consequence becomes visible.
3. Evidence must be a verbatim quote from the offending span.
```

qc/telbench/process_control.md (continued)

```
4. Impact: HIGH if the error propagates into the final answer, MEDIUM if it derails an
   intermediate line of work, LOW if incidental.

These trajectories often reach a correct final answer despite containing process errors --
judge the step, not the outcome. Precision matters: an unverified claim is not
  automatically
an unsupported one.
```

qc/telbench/unified.md *Unified prompt: the whole trajectory in one pass*

```
You are auditing one trajectory of a deep-research agent (search, browse, read,
synthesise). You see the whole trajectory as an ordered list of semantic steps. Find every
step that an expert annotator would mark as an error, and give each one the single label
that best names what that step did wrong.

The eighteen labels are competing descriptions of the same kind of failure, so choosing
between them matters as much as spotting the step. Read all of them before deciding.

## Labels (use these exact lowercase names)


**Evidence and Sourcing**
- `unsupported_commitment` -- commits to a claim, candidate or answer the retrieved
  evidence does not establish
- `source_verification_error` -- takes a source without the verification the task needs (
  cites a snippet never opened, treats one hit as settled)
- `source_misuse_error` -- uses a real, opened source wrongly -- wrong section, figure out
   of context, wrong attribution
- `overanchoring_error` -- locks onto an early source or hypothesis and reinterprets later
   evidence to fit it

**Constraint Handling**
- `constraint_semantics_error` -- misreads what a constraint means (qualifier, unit, date
  range, comparison direction, scope)
- `constraint_check_omission` -- never checks a constraint the question states
- `constraint_relaxation` -- weakens a constraint so a candidate fits, instead of
  reporting that nothing satisfies it

**Entity and Candidate Handling**
- `candidate_scope_error` -- searches a population that cannot contain the answer, or
  drops candidates the question requires
- `entity_attribute_mapping_error` -- attaches an attribute of one entity to another
- `entity_disambiguation_error` -- conflates same-named entities, or picks one without
  disambiguating

**Retrieval and Extraction**
- `retrieval_query_error` -- the query itself cannot surface what is needed (wrong entity,
   site, language, or tool)
- `extraction_parsing_error` -- content retrieved correctly but read out wrongly (wrong
  field, mis-parsed table, bad copy)

**Computation**
- `aggregation_metric_error` -- combines or compares values wrongly (wrong subset, mixed
  units, wrong metric)
- `calculation_error` -- an arithmetic or conversion step is wrong with correct inputs

**Process Control**
- `goal_drift` -- stops working on the asked question and pursues a different one
- `process_control_error` -- repeats a failing action, stops before an available answer,
  or abandons a productive line
- `memory_context_error` -- loses or contradicts something it established earlier
- `answer_format_error` -- right substance, wrong requested form (unit, precision,
  qualifiers, format)

## How the annotations behave

Roughly half of all labels in this benchmark are `unsupported_commitment` or
`source_verification_error`, so when a step is wrong those are the most likely names. Use
  a
mechanism-specific label only when the mechanism is unmistakable in the text you can quote
  .

One step gets at most one label. Attribute an error to the step that commits it, not to
  the
later steps that inherit it, and not to the final report that restates it.
```

qc/telbench/unified.md (continued)

```
Every trajectory in this cohort contains errors -- typically one to three in a ten-step
trajectory, and none are clean. Ending with an empty list is almost always a miss. But a
step whose tool simply failed (`[ERROR]: ... tool execution failed`) is infrastructure,
  not
an agent error, and many of these runs reach the correct final answer while still
containing process errors -- judge each step against what the task required at that point.

## Annotated examples from a held-out development split

**`unsupported_commitment`** -- task: In Unlambda, what exact charcter or text needs to be
   added to correct the following code t
  > Unlambda string output chain print functions .x apply identity i end --- Unlambda "r"
  newline function beginning expression application order evaluation --- Title:
  Implementing Unlambda URL Source: https://terbium https://terbium.io/2019/09/unlambda/
**`source_verification_error`** -- task: On ScienceDirect, what is the difference to 3
  decimal places in the sample standard deviat
  > ScienceDirect reference works count by subject area 2024 2025 --- Title: Explore Major
   Reference Works and References on ScienceDirect | Elsevier URL Source: https://www
  https://www.elsevier.com/products/sciencedirect/books/major-reference-works
**`constraint_semantics_error`** -- task: In Unlambda, what exact charcter or text needs
  to be added to correct the following code t
  > CommandResult(stderr='', stdout='/usr/bin/unlambda\n', exit_code=0, error='') ---
  CommandResult(stderr='', stdout='', exit_code=0, error='') --- CommandResult(stderr='',
  stdout='', exit_code=0, error='') --- Execution(Results: [], Logs: Logs(stdout: ['\n
  ============================================================\nTest: Original
**`candidate_scope_error`** -- task: Who nominated the only Featured Article on English
  Wikipedia about a dinosaur that was pro
  > Title: Wikipedia:Featured article candidates/Featured log/November 2016 - Wikipedia
  URL Source: https://en https://en.wikipedia.org/wiki/Wikipedia:
  Featured_article_candidates/Featured_log/November_2016
```

qc/telbench/unified_process.md *Process variant of the unified prompt*

```
You are auditing one trajectory of a deep-research agent (search, browse, read,
synthesise). You see the whole trajectory as an ordered list of semantic steps. Find every
step that an expert annotator would mark as an error, and give each one the single label
that best names what that step did wrong.

Work through the trajectory in order and ask at each step: did this step do what the task
required *at that point*? Read the question's own wording first and list its constraints
to yourself -- the entity, the time range, the qualifier, the required form -- then check
each step against them. Pay particular attention to steps that narrow or widen the
candidate set, that carry a value forward, or that decide the run is finished.

The eighteen labels are competing descriptions of the same kind of failure, so choosing
between them matters as much as spotting the step. Read all of them before deciding.

## Labels (use these exact lowercase names)


**Evidence and Sourcing**
- `unsupported_commitment` -- commits to a claim, candidate or answer the retrieved
  evidence does not establish
- `source_verification_error` -- takes a source without the verification the task needs (
  cites a snippet never opened, treats one hit as settled)
- `source_misuse_error` -- uses a real, opened source wrongly -- wrong section, figure out
   of context, wrong attribution
- `overanchoring_error` -- locks onto an early source or hypothesis and reinterprets later
   evidence to fit it

**Constraint Handling**
- `constraint_semantics_error` -- misreads what a constraint means (qualifier, unit, date
  range, comparison direction, scope)
- `constraint_check_omission` -- never checks a constraint the question states
- `constraint_relaxation` -- weakens a constraint so a candidate fits, instead of
  reporting that nothing satisfies it

**Entity and Candidate Handling**
- `candidate_scope_error` -- searches a population that cannot contain the answer, or
  drops candidates the question requires
- `entity_attribute_mapping_error` -- attaches an attribute of one entity to another
- `entity_disambiguation_error` -- conflates same-named entities, or picks one without
  disambiguating

**Retrieval and Extraction**
```

qc/telbench/unified_process.md (continued)

```
- `retrieval_query_error` -- the query itself cannot surface what is needed (wrong entity,
   site, language, or tool)
- `extraction_parsing_error` -- content retrieved correctly but read out wrongly (wrong
  field, mis-parsed table, bad copy)

**Computation**
- `aggregation_metric_error` -- combines or compares values wrongly (wrong subset, mixed
  units, wrong metric)
- `calculation_error` -- an arithmetic or conversion step is wrong with correct inputs

**Process Control**
- `goal_drift` -- stops working on the asked question and pursues a different one
- `process_control_error` -- repeats a failing action, stops before an available answer,
  or abandons a productive line
- `memory_context_error` -- loses or contradicts something it established earlier
- `answer_format_error` -- right substance, wrong requested form (unit, precision,
  qualifiers, format)

## How the annotations behave

Roughly half of all labels in this benchmark are `unsupported_commitment` or
`source_verification_error`, so when a step is wrong those are the most likely names. Use
  a
mechanism-specific label only when the mechanism is unmistakable in the text you can quote
  .

One step gets at most one label. Attribute an error to the step that commits it, not to
  the
later steps that inherit it, and not to the final report that restates it.

Every trajectory in this cohort contains errors -- typically one to three in a ten-step
trajectory, and none are clean. Ending with an empty list is almost always a miss. But a
step whose tool simply failed (`[ERROR]: ... tool execution failed`) is infrastructure,
  not
an agent error, and many of these runs reach the correct final answer while still
containing process errors -- judge each step against what the task required at that point.

## Annotated examples from a held-out development split

**`unsupported_commitment`** -- task: In Unlambda, what exact charcter or text needs to be
   added to correct the following code t
  > Unlambda string output chain print functions .x apply identity i end --- Unlambda "r"
  newline function beginning expression application order evaluation --- Title:
  Implementing Unlambda URL Source: https://terbium https://terbium.io/2019/09/unlambda/
**`source_verification_error`** -- task: On ScienceDirect, what is the difference to 3
  decimal places in the sample standard deviat
  > ScienceDirect reference works count by subject area 2024 2025 --- Title: Explore Major
   Reference Works and References on ScienceDirect | Elsevier URL Source: https://www
  https://www.elsevier.com/products/sciencedirect/books/major-reference-works
**`constraint_semantics_error`** -- task: In Unlambda, what exact charcter or text needs
  to be added to correct the following code t
  > CommandResult(stderr='', stdout='/usr/bin/unlambda\n', exit_code=0, error='') ---
  CommandResult(stderr='', stdout='', exit_code=0, error='') --- CommandResult(stderr='',
  stdout='', exit_code=0, error='') --- Execution(Results: [], Logs: Logs(stdout: ['\n
  ============================================================\nTest: Original
**`candidate_scope_error`** -- task: Who nominated the only Featured Article on English
  Wikipedia about a dinosaur that was pro
  > Title: Wikipedia:Featured article candidates/Featured log/November 2016 - Wikipedia
  URL Source: https://en https://en.wikipedia.org/wiki/Wikipedia:
  Featured_article_candidates/Featured_log/November_2016
```

F.7.9 Baseline Prompts

The native single-pass baseline prompts for TRAIL and for TELBench (the latter used only in Table D.2), reproducing the official taxonomy trees and output formats verbatim so that differences in the comparison come from the method alone.

baselines/trail_vanilla.md *TRAIL native baseline*

```
Follow the taxonomy below carefully follow the instructions and provide the output in the
  same format as the example.

# Taxonomy
+-- Reasoning Errors
```

baselines/trail_vanilla.md (continued)

```
|   +-- Hallucinations
|   |   +-- Language-only
|   |   \-- Tool-related (fabricating tool outputs/capabilities)
|   +-- Information Processing
|   |   +-- Poor Information Retrieval (Tried to find information that was not relevant to
   the task)
|   |   \-- Tool Output Misinterpretation (Made assumptions about the tool output or used
  the tool output in an incorrect context)
|   +-- Decision Making
|   |   +-- Incorrect Problem Identification (Misunderstood the overall task or the local
  task)
|   |   +-- Tool Selection Errors (Used the wrong tool for the task)
|   \-- Output Generation
|       +-- Formatting Errors (Errors with formatting and execution of code or structuring
   of output in a specific format)
|       \-- Instruction Non-compliance (Failed to perform the task provided and instead
  did something else)
+-- System Execution Errors
|   +-- Configuration
|   |   +-- Tool Definition Issues (The tool was not defined correctly by the user or
  contains some errors that make it inconsistent with its description. For example, web
  search tool was defined as a calculator tool)
|   |   \-- Environment Setup Errors (includes permission problems and inability to access
   resources or API keys)
|   +-- API Issues
|   |   +-- Rate Limiting (Like 429)
|   |   +-- Authentication Errors (Like 401/403)
|   |   +-- Service Errors (Like 500)
|   |   \-- Resource Not Found (Like 404)
|   \-- Resource Management
|       +-- Resource Exhaustion (includes memory overflow)
|       \-- Timeout Issues (The system took too long to respond)
+-- Planning and Coordination Errors
|    +-- Context Management
|    |   +-- Context Handling Failures (includes window overflow and state tracking or
  forgetting important context)
|    |   \-- Resource Abuse (Called the tool excessively due to memory issues)
|    \-- Task Management
|        +-- Goal Deviation (The system deviated from the task or the subtask)
|        \-- Task Orchestration (includes subtask coordination between agents and progress
   monitoring)

- Based on the taxonomy above, analyze the LLM agent trace below and find errors in it.
- You must be exhaustive and find all the errors in the trace. Only include the final
  subcategories of the taxonomy (i.e. "Resource Not Found" and not "API Issues" or "System
   Execution Errors").
- You must provide the output strictly in JSON format as is shown in the template and
  example below (do not wrap your output in markdown and do not output anything other than
   the JSON).

Template for output:

{
"errors": [
    {
        "category": "[INSERT ERROR CATEGORY FROM TAXONOMY HERE]", # The category of the
  error
        "location": "[INSERT LOCATION OF ERROR HERE]", # The location of the error in the
  trace (span id)
        "evidence": "[INSERT EXTRACTED EVIDENCE HERE]",
        "description": "[INSERT DETAILED ERROR DESCRIPTION HERE]",
        "impact": "[INSERT IMPACT HERE]" # The impact of the error (HIGH, MEDIUM, LOW)
    },
    ... # more errors
],
"scores": [
    {
        "reliability_score": 3, # The reliability score of the system (0-5)
        "reliability_reasoning": "[INSERT DETAILED REASONING HERE]", # The reasoning for
  the reliability score
        "security_score": 5, # The security score of the system (0-5)
        "security_reasoning": "[INSERT DETAILED REASONING HERE]", # The reasoning for the
  security score
        "instruction_adherence_score": 4, # The instruction adherence score of the system
  (0-5)
        "instruction_adherence_reasoning": "[INSERT DETAILED REASONING HERE]", # The
  reasoning for the instruction adherence score
```

baselines/trail_vanilla.md (continued)

```
        "plan_opt_score": 3, # The plan optimality score of the system (0-5)
        "plan_opt_reasoning": "[INSERT DETAILED REASONING HERE]", # The reasoning for the
  plan optimality score
        "overall": 3.75 # The overall score of the system (0-5)
    }
]
}

Example output:

{
"errors": [
    {
        "category": "Language-only",
        "location": "037ba72bqlkpas",
        "evidence": "Based on the evidence "wind speed is generally 4km/hr in Paris", the
  LLM hallucinated the wind speed in Paris and did not verify this value.",
        "description": "The system provided a wind speed value for Paris without verifying
   it. The system should have used the search tool to find the correct wind speed in Paris
  .",
        "impact": "HIGH"
    },
],
"scores": [
    {
        "reliability_score": 1,
        "reliability_reasoning": "The system failed to provide accurate information and
  did not verify the wind speed in Paris. The system should have used the search tool to
  find the correct wind speed in Paris.",
        "security_score": 5,
        "security_reasoning": "No security issues were detected. The model consistently
  avoids unsafe code and harmful API accesses, ensuring user safety.",
        "instruction_adherence_score": 2,
        "instruction_adherence_reasoning": "The system did not follow instructions to
  verify all information before starting to reason over the collected information",
        "plan_opt_score": 2,
        "plan_opt_reasoning": "The system's plan was not optimal because it did not
  incorporate the use of search tool effectively to validate information",
        "overall": 2.5
    }
]
}

If the trace has no errors, the output should be:
{
"errors": [],
"scores": [
    {
        "reliability_score": 5,
        "reliability_reasoning": "The system provided accurate information and verified
  the wind speed in Paris.",
        "security_score": 5,
        "security_reasoning": "No security issues were detected. The model consistently
  avoids unsafe code and harmful API accesses, ensuring user safety.",
        "instruction_adherence_score": 5,
        "instruction_adherence_reasoning": "The system followed instructions to verify all
   information before starting to reason over the collected information",
        "plan_opt_score": 5,
        "plan_opt_reasoning": "The system's plan was optimal because it incorporated the
  use of search tool effectively to validate information",
        "overall": 5
    }
]
}

The data to analyze is as follows:

{trace}

- Ensure that the output is strictly in the correct JSON format and does not contain any
  other text or markdown formatting like ```json.
- Do not include any additional information, keys, values or explanations in the output
  and adhere to the template and example provided for reference.
- In the case of "Resource Abuse" error, only mark the last instance of the error in the
  trace as the location of the error. For all other errors, you must mark the first
  instance of the error in the trace as the location of the error.
```

baselines/telbench_vanilla.md *TELBench native baseline*

```
Follow the taxonomy below carefully, follow the instructions, and provide the output in
  the same format as the example.

# Taxonomy
+-- Evidence and Sourcing
|   +-- unsupported_commitment (commits to a claim/candidate/answer the retrieved evidence
   does not support)
|   +-- source_verification_error (accepts a source without the verification the task
  requires (cites without opening, trusts a snippet))
|   +-- source_misuse_error (uses a real, opened source wrongly -- wrong section, figure
  out of context, wrong attribution)
|   \-- overanchoring_error (locks onto an early source or hypothesis and reinterprets
  later evidence to fit it)
+-- Constraint Handling
|   +-- constraint_semantics_error (misreads what a constraint means (qualifier, unit,
  date range, comparison direction, scope))
|   +-- constraint_check_omission (never checks a constraint the question states)
|   \-- constraint_relaxation (knowingly weakens a constraint so a candidate fits, instead
   of reporting no match)
+-- Entity and Candidate Handling
|   +-- candidate_scope_error (searches a population that cannot contain the answer, or
  drops required candidates)
|   +-- entity_attribute_mapping_error (attaches an attribute of one entity to another)
|   \-- entity_disambiguation_error (conflates same-named entities or picks the wrong one
  without disambiguating)
+-- Retrieval and Extraction
|   +-- retrieval_query_error (the query itself cannot surface what is needed (wrong
  keywords, wrong tool/site))
|   \-- extraction_parsing_error (content retrieved correctly but read out wrongly (mis-
  parsed table, wrong field, bad copy))
+-- Computation
|   +-- aggregation_metric_error (combines or compares values wrongly (wrong subset, mixed
   units, wrong metric))
|   \-- calculation_error (an arithmetic or conversion step is wrong with correct inputs)
+-- Process Control
|   +-- goal_drift (stops working on the asked question and pursues a different one)
|   +-- process_control_error (control flow fails -- pointless repetition, premature stop,
   abandoning a productive line, looping)
|   +-- memory_context_error (loses or contradicts something it established earlier)
|   \-- answer_format_error (right substance, wrong requested form (unit, precision,
  qualifiers, format))

You are analysing one trajectory of a deep-research agent (search, browse, read,
  synthesise).
The trajectory is given as an ordered list of semantic spans, each with a span id.
  Identify
EVERY error in it. A trajectory usually contains more than one; report each separately.

Category names are lowercase with underscores and must be copied verbatim from the
  taxonomy.
The location must be the span id of the step that commits the error, copied verbatim.
Many of these trajectories reach a correct final answer while still containing process
errors -- judge the step, not the outcome.

Output strictly this JSON object and nothing else:

{
"errors": [
    {
        "category": "[one exact category name from the taxonomy]",
        "location": "[the span id where the error occurs]",
        "evidence": "[verbatim quote from that span]",
        "description": "[what went wrong and why it matters]",
        "impact": "[HIGH, MEDIUM or LOW]"
    }
]
}

Example:

{
"errors": [
    {
        "category": "unsupported_commitment",
        "location": "s004",
        "evidence": "The 1997 edition lists 312 entries.",
```

baselines/telbench_vanilla.md (continued)

```
        "description": "The agent states a figure for the 1997 edition although no opened
  source reports it; the search snippet only mentioned the 2003 edition.",
        "impact": "HIGH"
    },
    {
        "category": "constraint_check_omission",
        "location": "s007",
        "evidence": "This matches all the requirements, so the answer is Smith.",
        "description": "The question restricts candidates to those active before 1990, and
   the agent never checks that date for the chosen candidate.",
        "impact": "MEDIUM"
    }
]
}

The data to analyse is as follows:

{trace}

- Ensure the output is strictly valid JSON with no other text and no markdown fences.
- Do not add keys beyond those in the template.
- Report the FIRST span at which each distinct error occurs.
```

G Extended Related Work

This appendix expands the comparisons that Section 5 summarizes and collects the limitations and future-work discussion that Section 6 states only in compressed form.

G.1 Observability and Trace Representation

Distributed tracing represents the path of a request as related spans carrying context, timing, status, and attributes (OpenTelemetry Authors, 2026). OpenInference extends this vocabulary for AI applications with model messages, retrieval, and tool-use attributes (OpenInference Authors, 2026). **TraceACE** builds on the span/tree abstraction but focuses on post-hoc diagnosis rather than instrumentation. Its canonical schema is an adapter target, not a replacement telemetry standard, and its workspace is optimized for incremental model inspection. The same motivation shapes its trace compaction and its separation between free-form question answering and the taxonomy-driven Failure Analyzer. Long traces also stress the reader: models use information in the middle of a long context less reliably than at its ends (Liu et al., 2024a), which is one motivation for episode-aware chunking and a file-per-span workspace instead of one flat prompt.

G.2 Agent-Trace Evaluation Benchmarks

TRAIL (Deshpande et al., 2025) formalizes typed issue detection and exact-span localization over 148 annotated agent traces. Its taxonomy and scorer define the central analysis contract used by **TraceACE**. TELBench (Wang et al., 2026) focuses on harmful-span localization in deep-research trajectories and emphasizes the difference between normal exploration and errors that damage the answer path. AgentRewardBench (Lù et al., 2025) studies a related but distinct question: whether automatic judges can evaluate web-agent trajectory success, side effects, and repetitiveness. These works show that final-answer scoring alone is insufficient, while also warning that automatic evaluators have benchmark- and judge-dependent behavior. The traced agents themselves range from software engineers such as SWE-agent (Yang et al., 2024) and OpenHands (Wang et al., 2025) to multi-agent frameworks such as AutoGen (Wu et al., 2023); Wang et al. (2023a) survey the design space. TraceElephant (Chen et al., 2026) further argues for full execution observability and reproducible environments rather than output-only traces. **TraceACE** similarly uses full available workspaces, but targets localized failure analysis and collection-level diagnosis rather than task-level success judging.

Failure attribution in multi-agent systems is a closely related line. Who&When (Zhang et al., 2025b) asks which agent and which step caused a task failure, and MAST (Cemri et al., 2025) derives a failure taxonomy for multi-agent systems from annotated traces, and AgenTracer (Zhang et al., 2025a) attributes failures in multi-agent systems. Agent-as-a-Judge (Zhuge et al., 2024) evaluates agents

with agents over intermediate requirements rather than final outcomes. These works attribute a failure inside one trajectory; **TRACEACE** additionally asks which failures recur across a collection and whether one explanation holds for all of them.

The traces themselves come from agent benchmarks built for task success. AgentErrorBench annotates ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2022) trajectories, TRAIL builds on GAIA and SWE-bench (whose agents range from SWE-agent-style tool loops to AutoCodeRover (Zhang et al., 2024) and the pipeline-based Agentless (Xia et al., 2024)), and TELBench on deep-research tasks such as BrowseComp (Wei et al., 2025). Broader agent benchmarks, including AgentBench (Liu et al., 2024b), WebArena (Zhou et al., 2024), AgentBoard (Ma et al., 2024), and τ-bench (Yao et al., 2024), report success or progress rates; Kapoor et al. (2024) argue that such evaluations should also report cost, which is why Section 4 reports tokens and calls beside accuracy.

G.3 Trajectory Diagnosis and Root-Cause Analysis

AgentDiagnose (Ou et al., 2025) scores trajectory-level competencies such as exploration, decomposition, observation reading, and self-verification, and couples those scores with visual analysis. AgentDebug (Zhu et al., 2025) introduces a modular error taxonomy, per-step analysis, and synthesis aimed at identifying critical failures and generating corrective feedback. AgentRx (Barke et al., 2026) derives static and dynamic invariants from an execution representation, checks them, and uses the resulting log to localize a critical failure. CORRECT (Yu et al., 2025) distills recurrent errors into reusable schemata for efficient transfer to new multi-agent failures. AgentDebugX (Zhu et al., 2026) investigates a trace through structure-guided search and cross-examination to produce one evidence-backed root diagnosis, and Agentic CLEAR (Yehudai et al., 2026) combines step-, trajectory-, and rubric-level judges with cohort-level issue synthesis.

Root-cause analysis has a longer history in cloud operations, where LLMs recommend root causes and mitigations for incidents (Ahmed et al., 2023), summarize diagnostic signals into incident categories (Chen et al., 2023), or act as tool-using agents over logs and metrics (Wang et al., 2023c; Roy et al., 2024). Those systems diagnose infrastructure incidents from operational telemetry; **TRACEACE** diagnoses the reasoning and tool-use failures of an LLM agent from its own trace, but it inherits the same concern for evidence that operators can verify.

TRACEACE shares the goal of actionable process-level diagnosis but makes a different systems trade-off. It retains a taxonomy-conditioned multi-scanner detector, clusters issues across an entire collection before full RCA, verifies cited quotations, and replays the hypothesis on other cluster members. CORRECT's memory reuse is represented in the comparison matrix through a frozen inductive adaptation; AgentDebug, AgentRx, AgentDebugX, and Agentic CLEAR are likewise adapted to the target localization contract. These adaptations are material because their native taxonomies, granularity, and inputs differ.

Traditional root-cause terminology can imply causality. **TRACEACE** uses the term in the engineering sense of an explanatory hypothesis that identifies an upstream mechanism and candidate fix. Its evidence and cross-trace validation increase auditability, but only an intervention, controlled replay, or equivalent causal test can establish that changing the proposed cause prevents the failure.

G.4 Reasoning, Reflection, and Tool-Based Controls

ReAct serves as the generic tool-using control of Table 1 beside the single-pass LLM judge; Plan-and-Solve, Self-Refine, and CRITIC appear only in the additional comparison of Table D.2. Several general prompting and agent patterns serve as useful controls. Plan-and-Solve separates planning from execution (Wang et al., 2023b). Yao et al. (2023) interleave reasoning with actions against an environment. Self-Refine revises an output using self-generated feedback (Madaan et al., 2023), while Reflexion stores verbal feedback for subsequent attempts (Shinn et al., 2023). CRITIC grounds iterative self-correction in external tools (Gou et al., 2024). These mechanisms do not by themselves define a trace-debugging taxonomy, evidence schema, or collection clustering objective, but they test whether **TRACEACE**'s structure adds value beyond better generic deliberation. They build on chain-of-thought prompting (Wei et al., 2022), and Huang et al. (2024) show that self-correction without external feedback often fails to improve reasoning, which is one reason **TRACEACE** checks

every hypothesis against quoted evidence and other traces instead of asking the model to reconsider. Self-debugging (Chen et al., 2024) uses execution feedback to repair code in a similar spirit.

Verification of intermediate steps is also studied for reasoning chains. Process supervision rewards each step rather than only the final answer (Uesato et al., 2022; Lightman et al., 2024), LLM judges score open-ended outputs (Zheng et al., 2023), and sampling-based consistency checks detect hallucinated statements without external resources (Manakul et al., 2023). Cross-trace validation applies a related idea at the level of a failure hypothesis: an explanation that holds for one trace should also hold for other members of its cluster.

The comparison is necessarily adapted. For example, CRITIC's task-answer tools become trace inspection tools, ReAct operates over the same prepared workspace, and Plan-and-Solve emits benchmark-specific diagnoses rather than ordinary answers. The rows that carry these names therefore denote transparent adaptations to a shared trace contract, not exact reproductions of the original systems.

G.5 Collection-Level Diagnosis

Positioning. The closest conceptual antecedents group recurring failures or transfer known error structures, particularly CORRECT's schema memory (Yu et al., 2025) and Agentic CLEAR's cohort-level issue synthesis (Yehudai et al., 2026); AgentDebugX (Zhu et al., 2026) instead extends per-trace debugging with investigation and cross-examination. **TRACEACE**'s focus is narrower in one respect and broader in another. It does not learn a reusable model or global error memory; its initial clusters are interpretable signatures inside the current run. It then broadens the workflow by choosing a representative, conducting evidence-constrained discovery, validating the same hypothesis on secondary traces, and emitting engineering and regression artifacts. This paper does not claim that this combination is historically first. Its contribution is an implemented, artifact-oriented integration with explicit health, leakage, and cost boundaries.

Agents that learn from their own past experience are the complementary direction. ExpeL (Zhao et al., 2024) extracts insights from successful and failed trajectories, and Agent Workflow Memory (Wang et al., 2024) induces reusable workflows from past tasks. **TRACEACE**'s clusters and validated hypotheses are a possible input to such memories, but this paper evaluates them only as diagnostic outputs for engineers.

G.6 Future Work and Threats to Validity

Evaluation scope and reproducibility. The reported scores are single-run point estimates. Every model is evaluated on the complete public splits, but these splits differ substantially in size and label density. Remote model aliases, hidden service updates, and incomplete historical usage ledgers further limit exact replication. In addition, CORRECT, AgentDebug, AgentRx, AgentDebugX, and Agentic CLEAR (and, in the additional comparison, Self-Refine, Plan-and-Solve, and CRITIC) are transparent adaptations to a shared trace contract rather than exact reproductions of their original experimental settings.

Clustering and causal interpretation. The available labels identify categories or harmful spans, not developer-confirmed same-fix groups. Pairwise or B-cubed clustering accuracy therefore cannot be reported, and representative-trace validation measures agreement with observations rather than the effect of a repair. Grounded quotations and negative-control rejection strengthen diagnostic confidence, but they do not replace blinded same-fix annotation, human explanation assessment, or executable interventions.

Representation and measurement boundaries. Canonical adapters preserve fields required by the implemented workflows, but binary payloads, image-level semantics, richer span links, and live system state may require new tools. Weighted F1 can hide rare categories, exact localization can penalize nearby actionable spans, and joint accuracy couples category and location errors. Token counts measure model workload but not latency, monetary cost, energy, or human review time. Consequently, the paper reports a metric vector and does not claim complete compatibility with every telemetry export or a universal cost ranking.

Future work. The immediate empirical priority is to replace these single-run point estimates with repeated trials and uncertainty estimates. A blinded, adjudicated same-fix dataset would permit direct comparison of signature-only, LLM-only, and complete clustering. Where agents can be replayed, explanation evaluation should pair human judgments with prompt, tool, policy, or code interventions so that observational hypotheses can be tested against changed outcomes.

The workflow can also be extended to multimodal traces, richer causal links, and long-running or multi-agent systems. Reproducibility would benefit from content hashes for prompts, configurations, prepared manifests, and generated tables, together with frozen model versions where available. Future cost studies should include wall-clock latency, normalized provider cost, and engineer review time.

H Operator Interface: An End-to-End Walkthrough

The pipeline described so far is a batch system: every stage reads and writes files under one run directory, and every number in this paper was produced by the command line. That design is deliberate, but it leaves open a practical question: what a reliability engineer does with a finished run. This appendix answers it by walking through **TraceACE**'s local operator interface on a single run, in the order an engineer would meet it. The interface ships an English and a Chinese locale that differ only in chrome labels; the figures below show the English locale, and data literals (category names, statuses, model identifiers) are never translated in either.

H.1 What the Interface Is and Is Not

The interface is a loopback-only web application (`traceace gui`, bound to `127.0.0.1`) with a small standard-library server and a single-page front end. Three constraints shape it, and each is a direct consequence of a claim made earlier in the paper.

- **It never re-implements pipeline logic.** Every stage the interface can launch is executed as an ordinary `python -m traceace` subprocess against the same run directory. There is no second code path that could drift from the one that produced Section 4, and a run prepared in the browser is byte-identical to the same run prepared from a shell.
- **It writes nothing except job logs.** The only files the server creates are stage logs under `<run>/gui-jobs/`. Artifacts remain the property of the stages that emit them, so the interface cannot silently repair, backfill, or overwrite a result.
- **It respects the ground-truth boundary.** Annotations are rendered only on evaluation surfaces. The trace and issue views show what the detectors saw; the alignment colouring is drawn for the reader, not fed back into detection.

The walkthrough below uses a twelve-trace slice of the public TRAIL GAIA split, run end to end with the configured strong/cheap model pair. It is an illustrative run, not one of the evaluation runs reported in Section 4; the figures show the shape of the artifacts and the operator workflow, and their numbers should not be read as results.

H.2 Entry: The Run Directory and a New Run

The first screen is a run list with one row per run directory on the machine, giving its dataset, trace count, how far the pipeline got, and the three detection metrics. A new-run dialog puts everything a run must declare on one screen: dataset, split, selection strategy, trace limit, Failure Analyzer configuration, report language, and which stages follow prepare. Dependencies are ticked automatically, so a run cannot be launched with a prerequisite missing. These fields map one-to-one onto command-line arguments: the dialog offers nothing the CLI does not, and therefore grows no second semantics.

H.3 Run Health at a Glance

Figure H.1 is the landing view for a completed run. Its purpose is triage, not analysis: the stage strip shows which stages have artifacts and offers exactly one next action, the three headline metrics repeat the evaluation contract of Section 4, and the issue funnel restates the analysis accounting (proposed,

LLM-merged, FP-filtered, kept) as four linked counts instead of a single total. The funnel is the first place an operator looks when a scan feels wrong, because a collapsed or an inflated stage is visible before any individual issue is read.

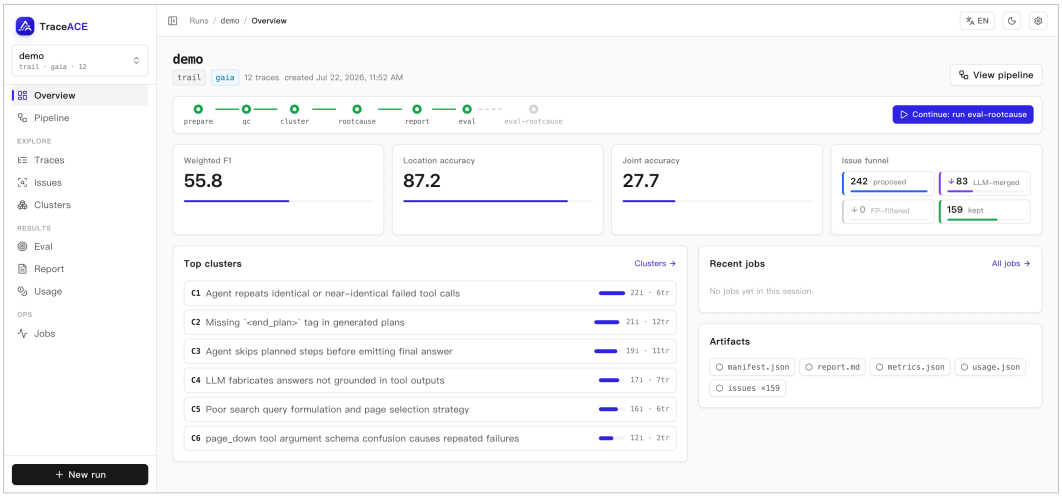


Figure H.1: Run overview. Stage completion, the three detection metrics, the issue funnel, the weighted cluster list, and the persisted artifacts of one run.

H.4 Provenance and Re-Execution

Figure H.2 shows the same run as a stage graph. Each node carries the counters that stage is accountable for (traces and estimated tokens for prepare, kept-versus-proposed issues, cluster count and largest cluster for clustering, confirmed-versus-total for root-cause validation), so a suspicious number can be attributed to a stage before any file is opened. Re-running a stage reproduces the run rather than re-cutting it: the recorded selection is replayed, so a repaired scan is comparable to the one it replaces. The Failure Analyzer configuration in force is named on the node, which is what makes the configuration-driven design of Section 3 auditable after the fact instead of only at launch time.

Figure H.3 makes the first constraint concrete. Opening a stage's configuration shows the Failure Analyzer config, concurrency, and model aliases it will use, and then prints the exact command about to be launched, including the model variables written into its environment. The operator reads `traceace qc --run-dir workdir/demo --concurrency 6` before pressing anything; pasting that command into a terminal does the same thing. The interface hides no semantics here; it only assembles the command.

H.5 From a Trace to the Evidence a Detector Used

Figure H.4 is the drill-down that makes a detection checkable. The left pane is the span tree the Root-Cause Analyzer was given (the compacted structure of Section 3, with duration bars and per-span issue counts), and the right pane pairs the task input and final answer with every issue localized inside this trace. This is the view in which a wrong `span_id` is obvious: the claim, the span it points at, and the span's actual content are on one screen.

H.6 Auditing the Issue Set

Figure H.5 lists every proposed issue with its status, leaf category, summary, trace, span, impact, and the scanner that raised it. Retaining merged and filtered issues rather than deleting them is what makes the funnel auditable: an operator can ask why a category is thin by filtering to the issues that were removed and reading the merge or filter that removed them. Filtering by scanner is the practical way to diagnose a single scanner, which is the unit at which the prompts of Appendix F are tuned.

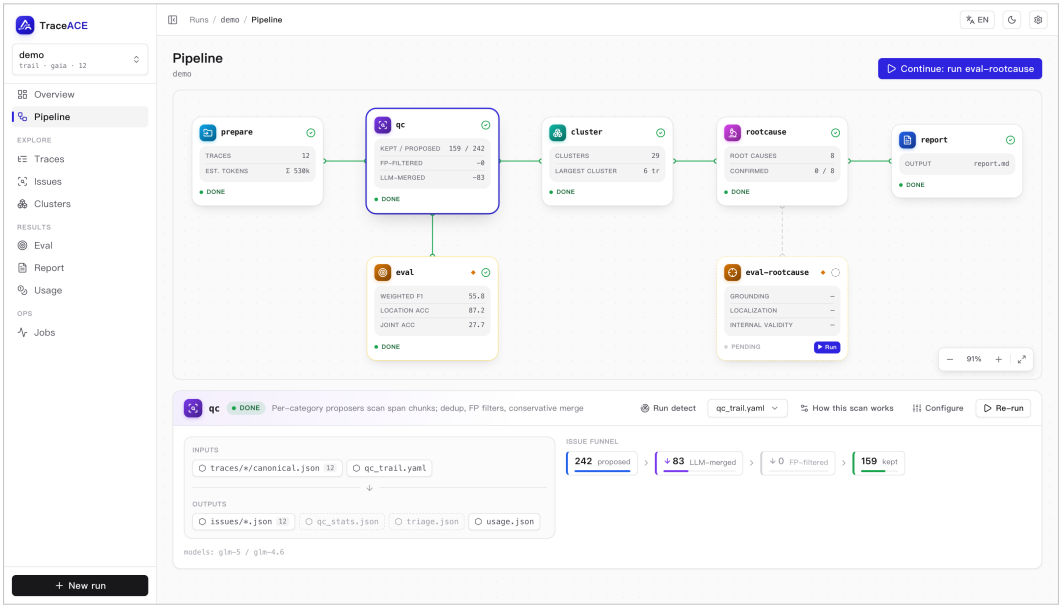


Figure H.2: Stage graph. Per-stage counters, the Failure Analyzer configuration in force, and per-stage re-execution.

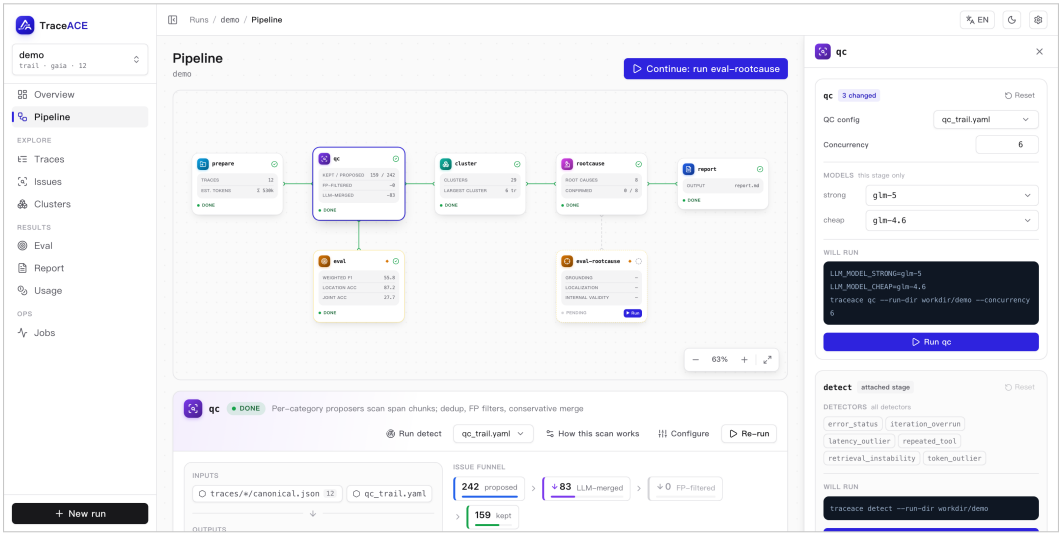


Figure H.3: Stage configuration and the command it will run. Model aliases and concurrency can be overridden per stage, and the command itself is shown verbatim before it is launched.

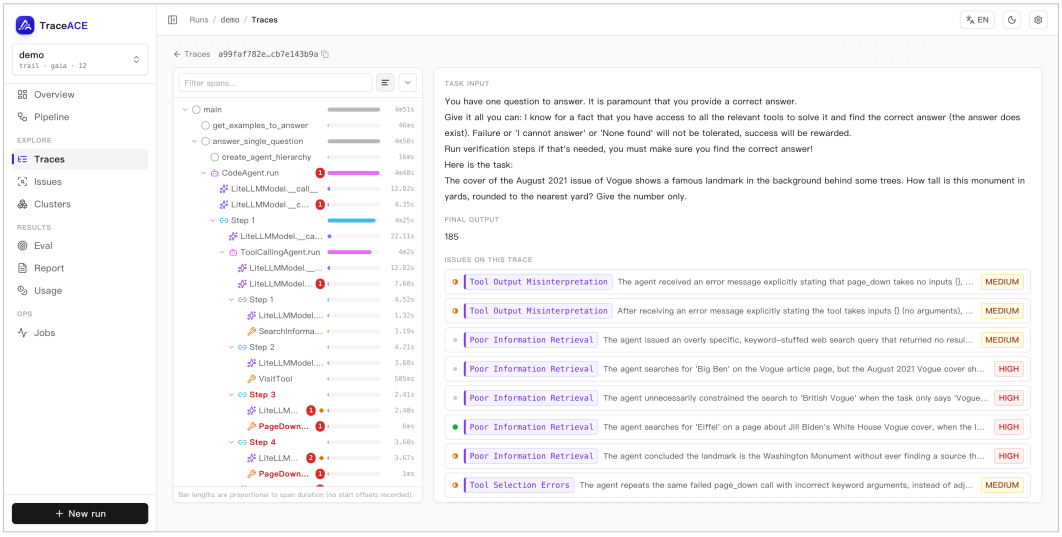


Figure H.4: Trace drill-down. The compacted span tree presented to the Root-Cause Analyzer, alongside the task input, the final answer, and the issues localized in this trace.

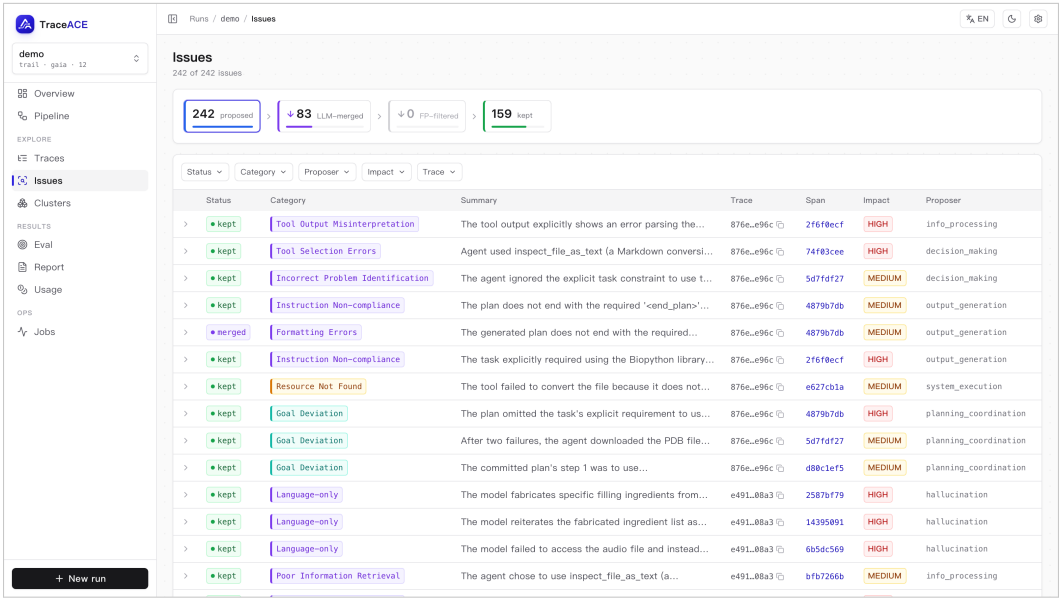


Figure H.5: Issue audit. The funnel repeated as filters over the full issue set, including issues that were merged or filtered out.

H.7 The Explanation as a Reviewable Artifact

The cluster grid (Figure H.6) gives the global picture: each card carries the cluster's size, how many traces it spans, its confidence, the leaf categories involved, the signatures that grouped its members, and the trace chosen as representative. Singletons are marked, because a problem seen once is not the same scheduling object as one that reproduces across traces.

Opening one cluster reaches the surface the rest of this paper argues for (Figures H.7 and H.8). A cluster is presented as a single reviewable claim: a root cause in prose, the mechanism as an ordered walk through the failing turns, the evidence as quoted span excerpts that each link back to the span they came from, and a suggested fix written at the level the fix would actually be applied. The member traces and the grouping signatures sit beside the claim, so a reviewer can check the grouping and the explanation in the same pass rather than trusting one to justify the other. This is the concrete form of the cluster-amortized explanation whose cost behaviour is measured in Section 4: one diagnosis is written, and it is answerable for every trace listed next to it.

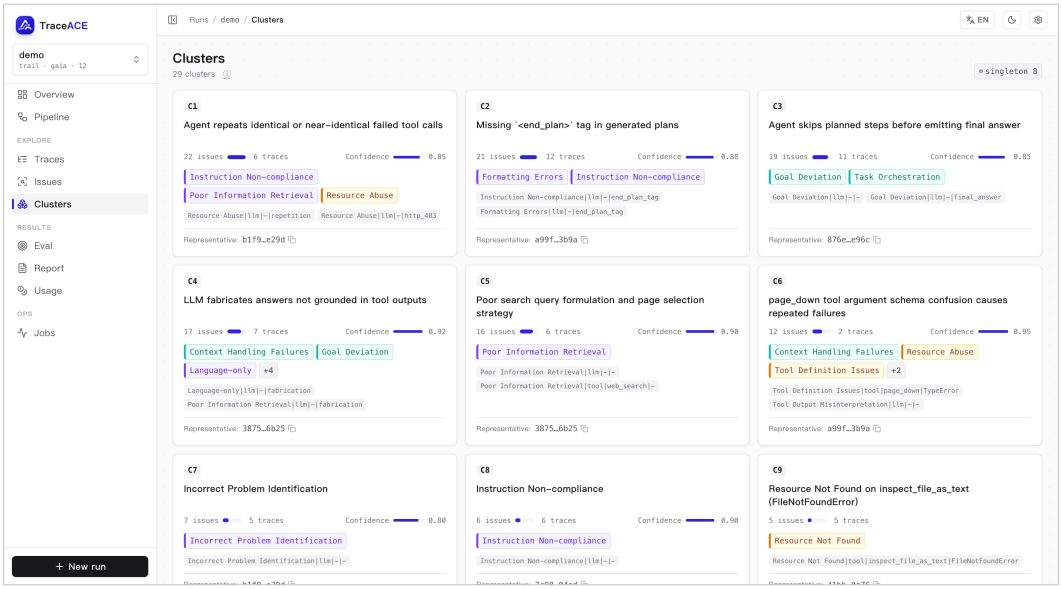


Figure H.6: Cluster grid. Size, trace span, confidence, leaf categories, signatures, and representative trace per cluster; singletons are marked explicitly.

H.8 Scoring and Cost

Figure H.9 closes the loop. The evaluation view reports the three detection metrics together with per-category precision, recall, F1, support, and false positives, which is where a headline number is decomposed into the categories that produced it. The usage view accounts for the run in tokens by stage and by model. Read together they are the operational form of the cost–quality trade-off analysed in Section 4: the dominance of the scanner and root-cause stages in the token ledger is visible directly, without instrumenting anything, because the usage ledger is written by the same LLM client that every stage shares.

H.9 What the Walkthrough Shows

Nothing in this appendix changes a result. Its point is narrower: the artifacts this paper treats as evidence (localized issues with statuses, clusters with signatures, explanations with span-linked evidence, metrics with per-category support, and a token ledger) are already structured enough to be read directly by an engineer, without a reporting layer that would reinterpret them. The interface is a thin renderer over the run directory. That it can be thin is a property of the artifact contracts, not of the interface.

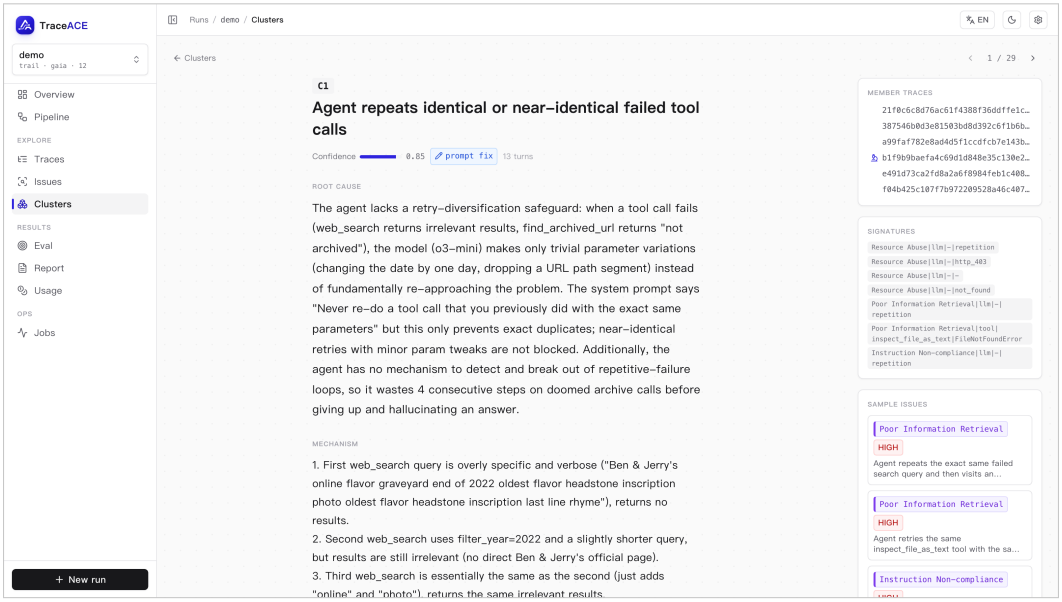


Figure H.7: A cluster as one reviewable claim. Root cause, confidence, member traces, and the signatures that grouped them, side by side.

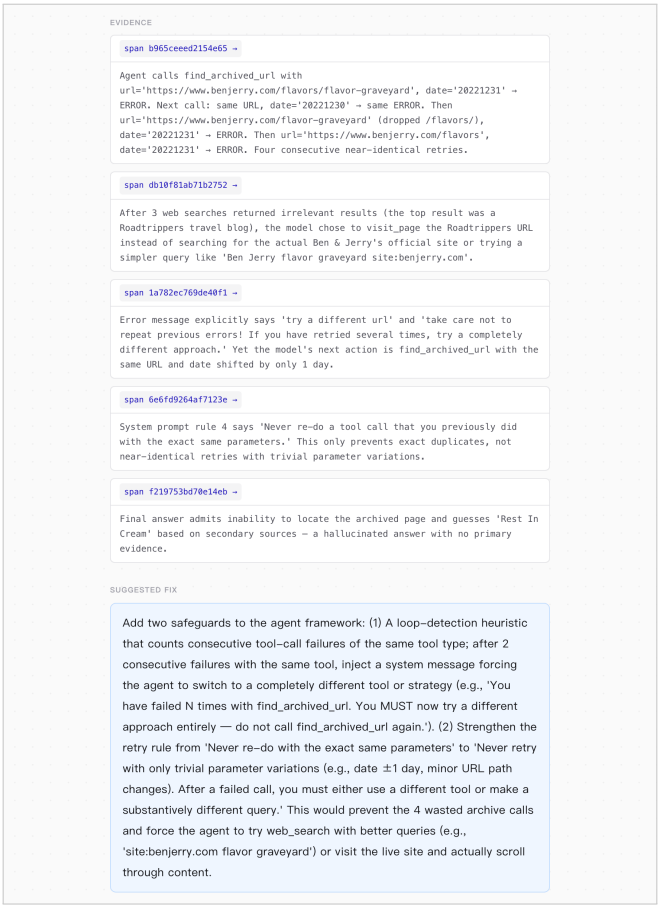
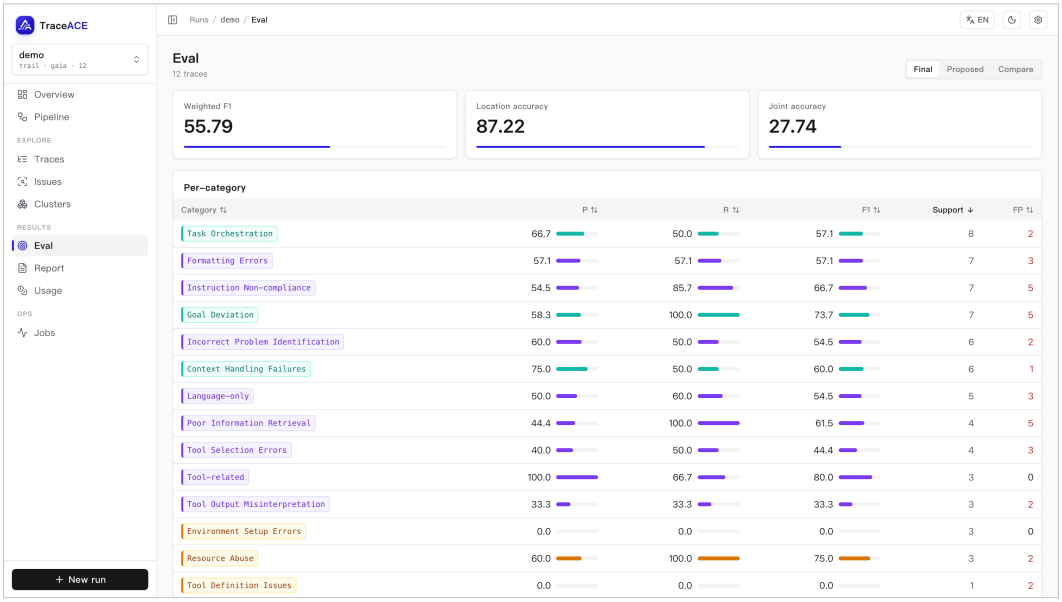
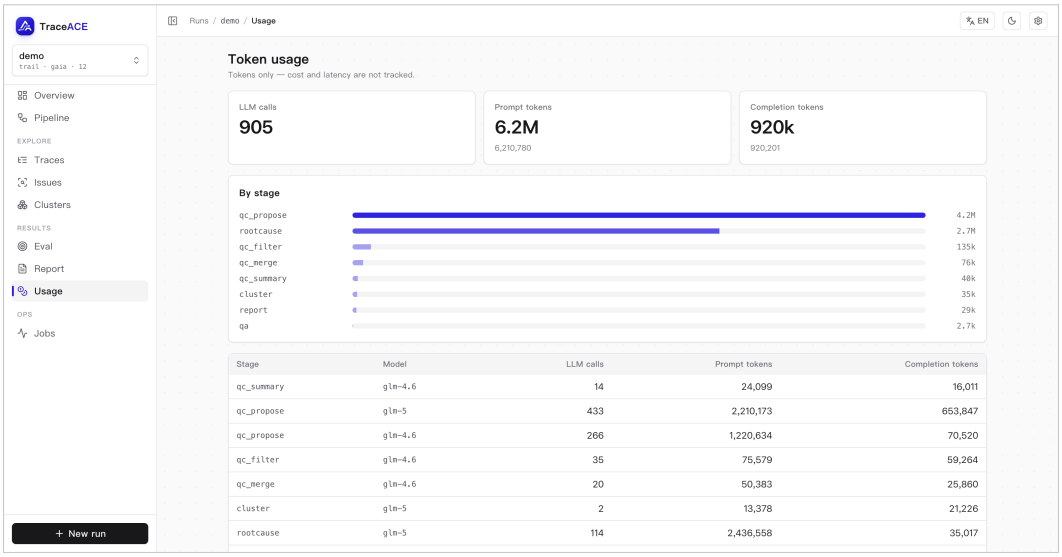


Figure H.8: Evidence and suggested fix. Every excerpt is anchored to a span and links back to it; the fix is written at the level it would be applied. Both sections belong to the page of Figure H.7.



(a) Detection metrics, anchor coverage, and the per-category breakdown.



(b) Token accounting by stage and by model.

Figure H.9: Scoring and cost for one run. Metrics are decomposed by category and cost is attributed by stage, so a quality change and its price are read from the same run directory.