TRACESIR: A MULTI-AGENT CRITIQUE FRAMEWORK FOR STRUCTURED ANALYSIS AND REPORTING OF AGENTIC EXECUTION TRACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic systems augment large language models with external tools and iterative decision making, enabling complex tasks such as deep research, function calling, and coding. However, their long and intricate execution traces make failure diagnosis and root cause analysis extremely challenging. Manual inspection does not scale, and applying LLMs directly to raw traces suffers from length limits and unreliable reasoning. Focusing solely on final task outcomes further discards critical behavioral information required for accurate issue localization. To address these limitations, we introduce **Trace Critique** as the task of producing structured analyses and reports over agentic execution traces, and propose **TraceSIR**, a multi-agent framework that operationalizes this task. TraceSIR coordinates three specialized agents: (1) *StructureAgent*, which introduces a novel abstraction format, `TraceFormat`, to compress execution traces while preserving essential behavioral information; (2) *InsightAgent*, which performs fine-grained diagnosis including issue localization, root cause analysis, and optimization suggestions; (3) *ReportAgent*, which aggregates insights across task instances and generates comprehensive analysis reports. To evaluate TraceSIR, we construct **TraceBench**, covering three real-world agentic scenarios, and introduce `ReportEval`, an evaluation protocol for assessing the quality of analysis reports aligned with industry needs. Experiments show that TraceSIR consistently produces coherent, informative, and actionable reports, significantly outperforming existing approaches across all evaluation dimensions. We further show that TraceSIR provides correct and actionable diagnoses, matching human-annotated root causes, achieving higher scores in a format-independent factual-grounding audit, and enabling substantial recovery of originally failed tasks. Our code is publicly available [1].

1 INTRODUCTION

Agentic systems, in which large language models (LLMs) are augmented with external tools and iterative decision making, have emerged as a powerful paradigm for real-world applications such as ClaudeCode and OpenClaw. As a result, improving the effectiveness of agentic systems has become an important research topic. This requires the identification of the limitations of existing agentic models and the root causes of the issues they exhibit. However, the dominant practice of manually inspecting agent outputs does not scale to the long and complex execution traces of agentic systems, where a single task may involve thousands of tool invocations and sub-agent interactions spanning large numbers of tokens, making

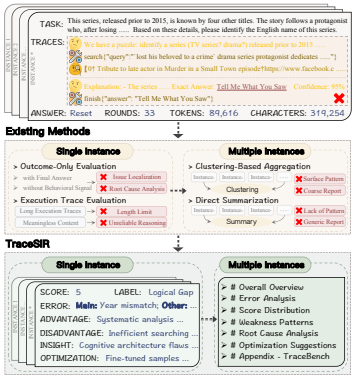

Figure 1: Comparison of existing methods and **TraceSIR** for agentic trace analysis and reporting.

[1] https://anonymous.4open.science/r/TraceSIR-8888

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.

comprehensive understanding and root cause analysis prohibitively difficult for humans (Deshpande et al., 2025). This difficulty is further compounded by the fact that valuable issue localization and root cause analysis often require coordinated comparison across multiple task instances. Such cross-instance analysis substantially increases the analytical burden and underscores the urgent need for automated analysis and reporting frameworks.

However, automated analysis of agentic systems faces a fundamental bottleneck, as long execution traces can easily exceed the input length limits of LLMs and substantially interfere with their ability to reliably analyze agent behavior, often leading to meaningless or hallucinatory content (Chen et al., 2026; Jia et al., 2026), as shown in Figure 1. One possible alternative is to focus solely on final task outcomes, but relying only on outcomes discards large amounts of valuable information embedded in the execution traces, which directly hinders accurate issue localization and prevents root cause analysis that depends on behavioral context. Accordingly, a central challenge is how to abstract agentic execution traces in a way that preserves core information necessary for accurate issue localization and root cause analysis. In addition, presenting researchers with comprehensive and coherent analysis reports poses another important challenge. To date, little prior work has focused on generating and evaluating analysis reports that align with practical industry needs.

Motivated by these challenges, we formulate **Trace Critique** as the task of generating structured analyses and reports over agentic execution traces to support issue localization and root-cause analysis. We operationalize this task with **TraceSIR**, a multi-agent critique framework that coordinates three specialized agents: *StructureAgent*, *InsightAgent*, and *ReportAgent*. Specifically, *StructureAgent* introduces a novel abstraction representation, referred to as `TraceFormat`, which structurally abstracts execution traces to substantially reduce redundancy while preserving core information from the original traces. Built on the structured traces, *InsightAgent* performs fine-grained analysis, including overall assessment, issue localization, weakness identification, root cause analysis, and optimization suggestions. Finally, *ReportAgent* summarizes critical errors observed in *InsightAgent* and determines whether report generation is warranted based on the number of task instances. When triggered, it conducts targeted statistical analysis and produces comprehensive analysis reports that provide researchers and engineers with detailed insights to support decisions.

To evaluate the effectiveness of TraceSIR, we collect task instances from three representative real-world agentic benchmarks, including BrowseComp (Deep Research) (Wei et al., 2025), Tau2Bench (Function Calling) (Barres et al., 2025), and SWE-bench (Agentic Coding) (Jimenez et al., 2024), and construct a unified benchmark named **TraceBench** for systematic experimental analysis. Specifically, we select 50 failure cases from each benchmark using GLM-4.6 as the target model, and retain execution traces in the standard OpenAI message format, which serve as inputs to our TraceSIR for structured analysis and reporting of agentic execution traces. To reflect the practical needs of researchers and engineers in real industry settings, we further introduce a report evaluation protocol, referred to as `ReportEval`. ReportEval defines a set of gold standard principles for assessing analysis reports and adopts LLM-as-a-judge to assist human annotation, enabling end-to-end evaluation of the quality and usability of reports generated by TraceSIR.

Experiments under the `ReportEval` protocol show that TraceSIR's Trace Critique reports on TraceBench consistently outperform those produced by the strong baseline ClaudeCode. On average, TraceSIR improves overall report quality by 9.7% under human evaluation by expert researchers in LLM Agent and 7.5% under LLM-as-a-judge evaluation, with the largest relative gain reaching 26.0%. These results indicate that TraceSIR produces analysis reports that better align with practical industry needs and deliver more reliable and informative diagnostic insights. More importantly, TraceSIR yields clear and robust advantages in issue localization, error interpretation, root cause analysis, and optimization suggestions, demonstrating its effectiveness in trace structuring and analysis across different underlying LLMs and its actionable value for downstream task recovery.

2 Related Work

Most existing approaches (Wei et al., 2025; Barres et al., 2025) for evaluating agentic systems remain outcome-oriented, which limits their ability to support issue localization and root cause analysis that depend on fine-grained behavioral context. Although some recent work attempts to incorporate partial intermediate signals, such as selected states (Avraham et al., 2026; Chuang et al., 2026) or summarized steps (Chen et al., 2026; Deshpande et al., 2025), these abstractions inevitably discard

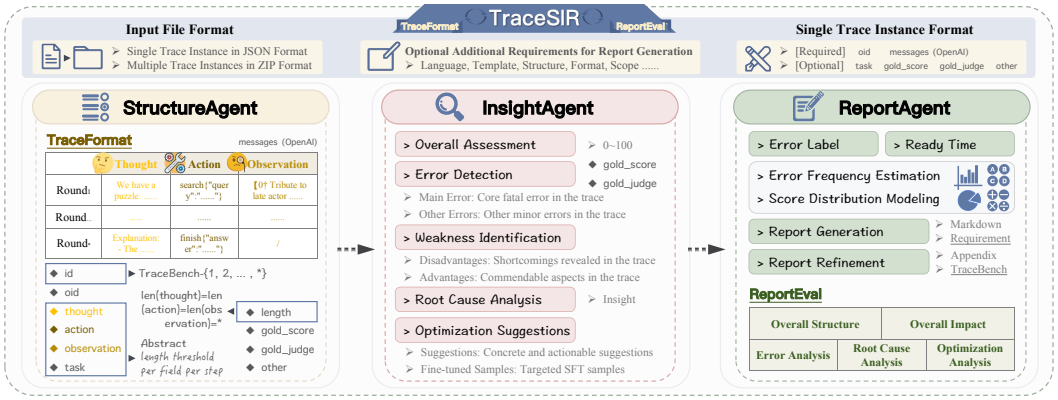

Figure 2: The overall architecture of **TraceSIR** for **Trace Critique**.

substantial contextual information embedded in long execution traces and fail to support in-depth, insight-driven root cause analysis, making systematic critique of agent behavior difficult.

In practice, understanding agentic systems often requires analyzing multiple task instances and producing consolidated analysis reports, yet such settings have received limited attention in prior work. Existing approaches typically evaluate cases in isolation and aggregate results only at a superficial level (Chen et al., 2026; Deshpande et al., 2025; Avraham et al., 2026), which obscures recurring failure patterns across execution traces. Moreover, techniques based on naive summarization (Aranya & Desai, 2026; Aubreville et al., 2025; Gan et al., 2025) or clustering-style aggregation (Rivera et al., 2026; Salmè et al., 2026; Zhao et al., 2026) struggle to preserve trace-level evidence, often resulting in overly coarse or weakly grounded reports.

In contrast, TraceSIR is designed specifically for structured analysis and reporting of agentic execution traces. By abstracting long traces into structured representations and coordinating specialized agents for critique and aggregation, TraceSIR enables scalable issue localization, root cause analysis, and actionable reporting across multiple cases. While individual components relate to prior work on agent trace structuring and error attribution, TraceSIR contributes a new, principled system-level formulation of Trace Critique, integrating length-aware abstraction, per-case diagnosis, cross-case aggregation, and dedicated benchmarking and evaluation into an end-to-end framework.

3 METHOD

We propose **TraceSIR**, a multi-agent critique framework for structured analysis and reporting of agentic execution traces. As illustrated in Figure 2, TraceSIR coordinates three specialized components, i.e., StructureAgent, InsightAgent, and ReportAgent. Each component operates autonomously with tool invocation over structured traces, while the overall framework is designed to support both fine-grained per-case analysis and scalable cross-case reporting for **Trace Critique**.

3.1 TRACEFORMAT

The input to TraceSIR can be either a single execution trace in JSON format or multiple trace cases in ZIP format. Each trace case is represented as an object with a required `oid` field for trace identification, a `messages` field containing the execution trace in the standard OpenAI format, and several optional auxiliary fields. Formally, for each trace case, the core execution trace stored in the `messages` field is represented as a sequence of messages $\mathcal{M} = \{m_1, m_2, \ldots, m_K\}$. Each message m_k consists of a role (e.g., `user`, `assistant`, `tool`) and the corresponding content. This representation is widely adopted by modern agentic systems and naturally captures iterative reasoning, tool invocation, and environment feedback. In addition to the `messages` field and an arbitrary trace identifier `oid`, a trace case may optionally include the following fields:

- `task`: a textual description of the task;
- `gold_score`: an automated evaluation score provided by external evaluators;
- `gold_judge`: textual feedback or error messages from automated assessment;

- `other`: auxiliary or miscellaneous metadata.

TraceSIR also supports an optional additional `requirement` specification that constrains or guides the report generation process, such as preferences over language. In the absence of explicit specifications, reports are generated in Chinese by default, considering practical deployment settings. All optional fields are treated as auxiliary signals and are not required for TraceSIR.

To enable scalable and reliable analysis, TraceSIR introduces `TraceFormat`, a structured abstraction of OpenAI messages that preserves essential behavioral information while providing a compact and analyzable representation. Given an execution trace $\mathcal{M}$, we define a deterministic parsing function $\Phi : \mathcal{M} \rightarrow \mathcal{T} = \{(t_i, a_i, o_i)\}_{i=1}^{N}$, which transforms the original message sequence into a structured trace $\mathcal{T}$ consisting of N interaction rounds. In each round, t_i means the agent's **Thought**, capturing its intermediate reasoning expressed in assistant messages; a_i denotes the **Action** selected by the agent, such as a tool invocation or explicit operation; and o_i represents the resulting **Observation** returned by the environment or tool.

`TraceFormat` strictly preserves the temporal order and causal alignment among reasoning, action, and observation, and can be rendered as a three-column table to support both human inspection and automated analysis. Each trace case is assigned a standardized identifier with a `TraceBench` prefix through a newly introduced `id` field, ensuring consistent indexing and tool-based cross-case comparison throughout the analysis and reporting process.

3.2 StructureAgent

While `TraceFormat` standardizes execution traces, long-horizon agent behaviors often contain verbose reasoning, long code snippets, or extensive tool outputs that exceed context limits for downstream analysis. StructureAgent addresses this bottleneck by performing trace-level abstraction.

Formally, given a structured trace $\mathcal{T}$, StructureAgent applies a length-aware abstraction operator parameterized by a threshold θ to produce a compressed trace $\mathcal{T}' = \mathcal{A}_\theta(\mathcal{T})$, where $\mathcal{A}_\theta$ selectively abstracts overlong elements while preserving essential behavioral information through automatic tool invocation. The threshold θ is a fixed dual criterion (100 words or 1,000 characters) applied *per field and per step*, rather than a global budget tied to total trace length: an element is abstracted only if it individually exceeds θ, so short steps are left verbatim and only verbose reasoning, code, or tool outputs are compressed, without obscuring critical signals required for subsequent analysis.

When necessary, StructureAgent also generates a compact abstraction of the `task` field. The resulting structured trace $\mathcal{T}'$ substantially reduces redundancy while remaining faithful to the original execution semantics, thereby enabling reliable and scalable downstream diagnosis.

3.3 InsightAgent

Built on the structured traces produced by StructureAgent, InsightAgent performs fine-grained, instance-level diagnostic analysis. Rather than relying solely on final task outcomes, it reasons over the entire structured execution trace to uncover behavioral issues and performance limitations.

Formally, given a structured trace $\mathcal{T}'$ and the associated `task` field q, InsightAgent produces a set of structured diagnostic outputs, denoted as $\mathcal{D} = \{s, E, W, R, O\}$, by autonomously invoking analysis tools over the structured trace. Here, s represents an overall task completion assessment ranging from 0 to 100. E captures detected errors in the execution trace and distinguishes between the primary, task-critical error and other secondary errors. W denotes identified weaknesses revealed by the trace, while also optionally noting notable strengths when relevant. R corresponds to root cause analysis, providing in-depth and insight-driven explanations of why the observed failures or limitations occur, and serves as a central diagnostic signal in our framework. Finally, O consists of optimization suggestions, including concrete textual recommendations and fine-tuning samples to address the identified issues and support actionable agent improvement.

When available, `gold_score` and `gold_judge` are incorporated as auxiliary reference signals to support task completion assessment and error identification and interpretation. In their absence, InsightAgent derives all diagnostic outputs solely from trace-level behavioral evidence. All diagnostic outputs for each trace case are returned to the user in a structured format, supporting fine-grained, per-case analysis and enabling reliable aggregation and comparison across multiple cases.

Scenario	Method	Backbone	Overall Score	OS	EA	RCA	OA	OI	Ranking
Deep Research	ClaudeCode	GLM-5	55.0	5.5	4.5	5.5	6.0	6.0	4
	ClaudeCode	Claude-4.6	66.0	6.5	7.5	5.5	**7.0**	6.5	2
	TraceSIR	GLM-5	65.0	7.5	6.5	5.5	6.0	7.0	3
	TraceSIR	Claude-4.6	**81.0**	**8.5**	**8.5**	**8.0**	**7.0**	**8.5**	1
Function Calling	ClaudeCode	GLM-5	40.0	5.0	3.5	4.5	3.0	4.0	4
	ClaudeCode	Claude-4.6	74.0	**7.5**	**7.5**	**8.5**	6.5	7.0	2
	TraceSIR	GLM-5	53.0	6.0	6.0	5.0	5.0	4.5	3
	TraceSIR	Claude-4.6	**77.0**	**7.5**	**7.5**	**8.5**	**7.5**	**7.5**	1
Agentic Coding	ClaudeCode	GLM-5	57.0	6.5	5.5	5.5	6.0	5.0	4
	ClaudeCode	Claude-4.6	77.0	7.5	8.0	7.0	7.5	8.5	2
	TraceSIR	GLM-5	62.0	5.5	6.5	6.5	6.5	6.0	3
	TraceSIR	Claude-4.6	**89.0**	**9.0**	**9.0**	**9.0**	**8.5**	**9.0**	1

Table 1: Comparison of report quality under the `ReportEval` protocol using human evaluation. TraceSIR is our system while ClaudeCode is the baseline; the same convention applies below.

3.4 REPORTAGENT

While InsightAgent focuses on individual task instances, practical diagnosis of agentic systems often requires coordinated analysis across multiple cases. ReportAgent operates at this higher level.

Formally, given a collection of diagnostic outputs $\{\mathcal{D}_1, \ldots, \mathcal{D}_M\}$ produced by InsightAgent for M trace cases, ReportAgent autonomously invokes analysis tools to determine whether report generation is warranted based on the number of available cases and predefined triggering criteria. When activated, it performs targeted statistical analysis over the aggregated diagnostics, including error frequency estimation and score distribution modeling, to uncover recurring issues and patterns.

For error analysis, ReportAgent summarizes the detected errors E_i of each case into a canonical error label ℓ_i, and estimates the frequency of each error type across M cases as $P(\ell) = \frac{1}{M}\sum_{i=1}^{M} \mathbb{I}(\ell_i = \ell)$, where ℓ_i denotes the error label derived from E_i for case i. Importantly, $P(\ell)$ is deterministically computed from structured per-case labels ℓ_i, ensuring exactly reproducible rather than model-hallucinated statistics. For performance analysis, ReportAgent models the distribution of task completion scores over predefined disjoint score intervals $\mathcal{B} = \{b_1, \ldots, b_L\}$. The score distribution is estimated as $P(b) = \frac{1}{M}\sum_{i=1}^{M} \mathbb{I}(\hat{s}_i \in b)$, where $\hat{s}_i$ denotes the score used for aggregation, taking `gold_score` when available and otherwise falling back to the score s_i produced by InsightAgent.

ReportAgent integrates quantitative statistics with qualitative insights to generate comprehensive analysis reports in Markdown format by autonomously invoking reporting and analysis tools. When user-specified `requirement` fields are provided, they are incorporated during report generation; otherwise, reports are produced according to a default analysis schema. After report generation, ReportAgent further refines the report by automatically identifying referenced trace cases through standardized `id` fields with the `TraceBench` prefix using a matching tool, and appending the corresponding structured trace data to the appendix. This process ensures transparency and traceability of the reported analyses. By aggregating trace-level evidence across cases, TraceSIR produces coherent, evidence-grounded reports that support practical research and engineering decision-making.

4 EXPERIMENTS

4.1 TRACEBENCH

To evaluate the effectiveness of TraceSIR, we construct **TraceBench**, a unified benchmark of agentic execution traces collected from three representative real-world agentic benchmarks: BrowseComp (Deep Research) (Wei et al., 2025), Tau2Bench (Function Calling) (Barres et al., 2025), and SWE-bench (Agentic Coding) (Jimenez et al., 2024). TraceBench includes 150 failed task instances of GLM-4.6 [2], distributed across three scenarios. Construction details are provided in Appendix A.

[2] https://huggingface.co/zai-org/GLM-4.6

Scenario	Method	Backbone	Overall Score	OS	EA	RCA	OA	OI	Ranking
Deep Research	ClaudeCode	GLM-5	82.7	8.7	8.0	7.3	**9.0**	8.3	4
	ClaudeCode	Claude-4.6	90.0	**9.3**	8.7	9.0	**9.0**	**9.0**	2
	TraceSIR	GLM-5	88.0	9.0	8.0	9.0	**9.0**	**9.0**	3
	TraceSIR	Claude-4.6	**91.3**	**9.3**	**9.0**	**9.3**	**9.0**	**9.0**	1
Function Calling	ClaudeCode	GLM-5	80.7	8.3	7.3	7.7	9.0	8.0	4
	ClaudeCode	Claude-4.6	88.0	8.7	8.3	**9.0**	9.0	**9.0**	3
	TraceSIR	GLM-5	89.3	**9.0**	8.7	**9.0**	9.0	**9.0**	2
	TraceSIR	Claude-4.6	**91.3**	**9.0**	**9.0**	**9.0**	**9.7**	**9.0**	1
Agentic Coding	ClaudeCode	GLM-5	58.7	6.0	4.0	5.7	8.0	5.7	4
	ClaudeCode	Claude-4.6	90.0	9.0	**9.0**	**9.0**	**9.0**	**9.0**	2
	TraceSIR	GLM-5	84.7	8.3	7.7	8.7	**9.0**	8.7	3
	TraceSIR	Claude-4.6	**90.7**	**9.3**	**9.0**	**9.0**	**9.0**	**9.0**	1

Table 2: Comparison of report quality under the `ReportEval` protocol using LLM-as-a-judge.

4.2 REPORTEVAL

To evaluate the quality and usefulness of reports generated by TraceSIR, we propose `ReportEval`, a report-centric evaluation protocol for agentic execution trace critique. `ReportEval` assesses each report along five dimensions with equal weight, scoring each part and overall.

The five evaluation dimensions of Trace Critique are defined as follows. **Overall Structure** (OS) evaluates whether the report is well organized, coherent, and clearly grounded in the analyzed execution traces. **Error Analysis** (EA) measures the correctness and trace support of the identified agent errors. **Root Cause Analysis** (RCA) assesses whether the report provides insightful and well-justified explanations of the underlying causes of the observed errors. **Optimization Analysis** (OA) evaluates the relevance, feasibility, and actionability of the proposed optimization suggestions. **Overall Impact** (OI) provides a holistic assessment of the report's usefulness from a practitioner's perspective, reflecting its value for understanding agent behavior and supporting decision making. Each dimension is scored on a scale ranging from 0 to 10. The overall score is computed as the equally weighted sum of the five dimension scores and normalized to a range from 0 to 100.

Each report is evaluated through two independent channels, including six blinded human experts and an LLM judge using GPT-5 with scores averaged over three runs. We release the rubric, judge prompt, per-rater scores, and agreement statistics for transparency. We assign equal weights to all dimensions to reflect their shared importance in practical trace analysis. Beyond `ReportEval`, we further conduct format-independent factual-grounding and gold-based diagnostic-correctness evaluations to assess factual support and diagnostic accuracy beyond presentation quality.

4.3 GOLD ANNOTATION

To evaluate whether TraceSIR's diagnoses are correct beyond presentation quality, we construct an expert-annotated gold set on half of TraceBench, covering 75 traces across the three scenarios. For each trace, annotators identify the final task failure, key trace-level error, and underlying root cause with supporting evidence. These annotations serve as the reference for evaluating issue localization, root-cause identification, and factual grounding. The gold set is produced independently of both `ReportEval` and TraceSIR's output schema. Annotation follows a dual-annotator protocol with an independent quality-control reviewer, yielding 90.7% inter-annotator agreement and Cohen's $\kappa = 0.66$ for localization criteria, indicating a reliable reference for diagnostic evaluation.

4.4 SETTINGS

TraceSIR is instantiated with two backbone LLMs, GLM-5 (GLM-5-Team et al., 2026) and Claude-4.6[3], and we use ClaudeCode[4] as the main baseline. Report quality is evaluated through a hybrid setting with six blinded expert LLM-agent engineers/researchers and an automated LLM-based

[3]https://www.anthropic.com/news/claude-opus-4-6

[4]https://github.com/anthropics/claude-code

judge. For fair comparison, all systems receive identical inputs: the same execution traces, the same requested report structure, and, when available, the same optional `gold_score`/`gold_judge` fields. Since `gold_judge` may contain evaluator feedback that hints at corrective actions, we provide it equally to all systems and additionally assess diagnostic quality with trace-grounded metrics checked against the execution traces and independent human gold annotations. Appendix B gives additional experimental details, while Appendix C details the baseline setup and released artifacts.

4.5 RESULTS

From Table 1 and Table 2, it can be observed that TraceSIR consistently and substantially outperforms ClaudeCode across all three agentic scenarios. From the perspective of human evaluation, TraceSIR yields consistent and sizable improvements over ClaudeCode with both backbone models. When using the weaker GLM-5 backbone, TraceSIR improves the overall score by 10.0%, 13.0%, and 5.0% in Deep Research, Function Calling, and Agentic Coding, respectively. With the stronger Claude-4.6 backbone, TraceSIR further improves performance by 15.0%, 3.0%, and 12.0% across these scenarios. Averaged across scenarios and backbones, TraceSIR achieves a 9.7% improvement under human evaluation, indicating a clear expert preference for its reports. Averaged across the five 10-point evaluation dimensions, TraceSIR further achieves mean improvements of 0.9, 1.3, 1.2, 0.8, and 0.9 points, respectively, with particularly pronounced gains in error analysis and root cause analysis, demonstrating its strengthened diagnostic capability and its effectiveness in supporting systematic analysis and report generation.

Results from LLM-as-a-judge evaluation exhibit a highly consistent trend. With the GLM-5 backbone, TraceSIR improves the overall score by 5.3%, 8.6%, and 26.0%, respectively. When paired with Claude-4.6, the corresponding improvements are 1.3%, 3.3%, and 0.7%. Overall, TraceSIR yields a 7.5% improvement over ClaudeCode under LLM-based evaluation. Notably, the largest gains appear in harder settings, especially Agentic Coding with weaker backbones, highlighting TraceSIR's robustness under constrained model capacity. The strong rank agreement between human and LLM evaluations, with Spearman $\rho = 0.81$ and 83.3% consistent rankings, further confirms that LLM-as-a-judge reliably preserves the relative performance trends observed in human evaluation, indicating the quality of TraceSIR reports is consistently recognized by experts and LLMs. We next analyze whether these report gains correspond to faithful, correct, and actionable diagnoses.

5 ANALYSIS

5.1 GOLD-GROUNDED CORRECTNESS

Beyond report-level quality, we evaluate whether TraceSIR identifies the actual issues and root causes in individual traces. Using expert gold annotations, an LLM judge determines whether each critique matches the annotated issue or the core root-cause error. Across both backbones, TraceSIR performs well above a 50% random baseline, showing that its critiques reliably capture trace-level diagnostic signals. Panel b of Figure 4 further shows that the stronger Claude-4.6 backbone performs best, while the weaker GLM-5 backbone remains competitive; the two backbones are numerically ordered but not significantly separated under a paired McNemar test.

We further decompose diagnostic correctness into two criteria in Table 3: *issue localization*, which measures whether the critique identifies where the trace goes wrong, and *root-cause identification*, which measures whether it pinpoints the true underlying cause. TraceSIR identifies the true root cause in 81.3% of cases with Claude-4.6 and 64.0% with GLM-5. Since both criteria are checked against a human gold standard independent of `ReportEval`, these results show that TraceSIR captures real diagnostic signals rather than merely producing well-structured reports.

5.2 FACTUAL GROUNDING

We also evaluate whether the generated reports are factually supported by the traces and gold annotations. Unlike `ReportEval`, which measures report quality, this factual-grounding audit checks verifiable report claims against the human gold annotations using an independent judge from a different model family. Each claim is labeled as supported, contradicted, or unverifiable. We compute factual grounding as $\mathrm{FG} = 100 \cdot \frac{2PC}{P+C}$, where P is faithfulness precision, measuring whether report

Criterion	Backbone	Deep Research	Function Calling	Agentic Coding	Overall
Issue localization	GLM-5	56.3	85.2	25.0	53.3
	Claude-4.6	75.0	88.9	65.6	76.0
Root-cause identification	GLM-5	81.3	66.7	53.1	64.0
	Claude-4.6	75.0	92.6	75.0	**81.3**

Table 3: Diagnostic correctness of TraceSIR against expert gold on the annotated half of TraceBench. Scores are reported in percent with n=75.

claims are supported rather than contradicted, and C is root-cause coverage, measuring whether the report captures important gold root causes. Scores are averaged over three runs.

As shown in Figure 3, TraceSIR achieves higher factual grounding in every paired setting, with mean FG 94.1 compared with 74.5 for ClaudeCode. The improvement is largest with the weaker GLM-5 backbone, where TraceSIR obtains 88.2 versus 52.4, and remains positive with Claude-4.6, where TraceSIR obtains 100.0 versus 96.5. The largest gap appears on Function Calling with GLM-5, where ClaudeCode often attributes failures to the environment while the gold annotations identify omitted agent actions. This example illustrates the value of grounding report claims against independent evidence: errors that may appear plausible in a quality-only evaluation are exposed when checked against trace-level diagnostic truth. Overall, these results indicate that TraceSIR's reports are not only higher quality under `ReportEval`, but also more faithful to independently annotated diagnostic evidence.

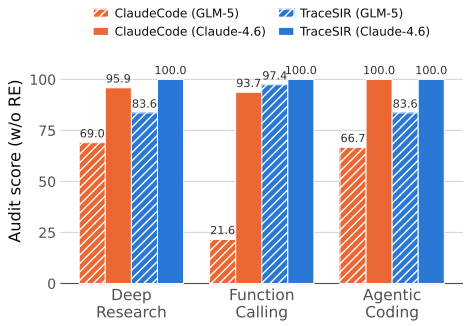

Figure 3: Factual-grounding audit of report faithfulness against the gold annotations.

5.3 Additional Baselines

We further compare TraceSIR with Codex and OpenCode under the `ReportEval` protocol using LLM-as-a-judge. These baselines are general-purpose agentic systems capable of reading trace files and producing analysis reports, making them strong practical alternatives in the absence of a dedicated trace-critique system. To keep the comparison controlled, all systems receive the same traces, the same task instructions, and the same requested report structure. TraceSIR, ClaudeCode, and OpenCode use Claude-4.6 as the backbone, while Codex uses GPT-5.5. As shown in panel a of Figure 4, TraceSIR achieves the highest scores across all three scenarios, followed by ClaudeCode, whereas Codex and OpenCode perform lower. This comparison is particularly informative because Codex uses a stronger underlying model but still does not match TraceSIR's report quality. The gap is especially clear for OpenCode, whose reports are concise but provide weaker root-cause and optimization analysis. These results suggest that TraceSIR's advantage comes primarily from its structured trace-analysis framework, including length-aware abstraction, per-case diagnosis, and cross-case aggregation, rather than raw backbone capability or model scale alone.

5.4 Report-Guided Task Recovery

Beyond producing accurate diagnoses, we evaluate whether TraceSIR's reports can help recover originally failed tasks. We use a report-guided re-solve protocol where every case is an original failure by the target agent GLM-4.6, so the initial pass rate is 0 by construction. We then re-run the same agent in the real task environment under two guidance conditions: `+Generic`, a content-free retry hint, and `+TraceSIR`, the diagnosed root cause and fix from TraceSIR's report. Both conditions use the same environment, tools, and task budget; the only difference is the guidance injected into the agent. We measure pass@1 as the fraction of originally failed tasks that are solved after re-running. Function Calling and Deep Research re-run the full agent loop with tools, while Agentic Coding is scored with the official SWE-bench test harness.

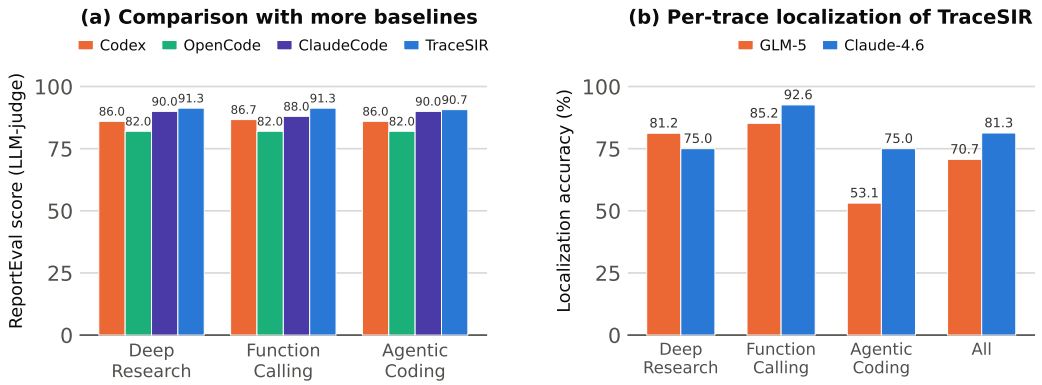

Figure 4: Additional analysis. Panel a compares TraceSIR with Codex, OpenCode, and ClaudeCode under `ReportEval`. Panel b reports issue-identification accuracy across scenarios and backbones.

Scenario	+ Generic	+ TraceSIR
Deep Research (BrowseComp)	24.0	26.0
Function Calling (Tau2Bench)	84.0	**96.0**
Agentic Coding (SWE-bench)	14.0	16.0

Table 4: Report-guided task recovery on 50 originally failed tasks per scenario.

As shown in Table 4, TraceSIR guidance recovers more failures than a generic retry hint in every scenario. The gain is largest on Function Calling, where `+TraceSIR` reaches 96.0% pass@1 compared with 84.0% for `+Generic`. In these cases, the report often identifies the specific missing action, such as an omitted tool call, that caused the original failure, allowing the agent to correct the failed trajectory rather than simply retrying. Ablating the report to root-cause analysis alone retains almost the entire gain on Function Calling, reaching 94.0%, indicating that diagnostic content, not generic advice or verbosity, drives the improvement. The gains are smaller on Deep Research and Agentic Coding, where recovery often requires new external evidence or extensive repository exploration that a post-hoc report cannot fully replace. Nevertheless, TraceSIR guidance still improves performance in both scenarios, showing that the reports provide useful corrective signals even when the remaining task difficulty is dominated by external search or complex code execution. Overall, these results show that TraceSIR's diagnoses are actionable and can translate trace critique into concrete task recovery, rather than serving only as retrospective descriptions.

5.5 Ablation and Efficiency

Appendix D and Appendix E further analyze TraceSIR's abstraction mechanism and computational efficiency. StructureAgent reduces trace length by 84% on average, allowing long-horizon traces that would otherwise exceed context limits to be analyzed in a single pipeline while preserving diagnostic signals. The full system costs approximately $1.09 per trace, showing that TraceSIR offers scalable cross-case diagnosis with practical overhead for real debugging workflows.

6 Conclusion

We presented **TraceSIR**, a framework for **Trace Critique** that enables structured analysis and reporting of agentic execution traces. By introducing `TraceFormat`, TraceSIR abstracts long traces while preserving critical behavioral information, enabling scalable issue localization and root-cause analysis. Its coordinated StructureAgent, InsightAgent, and ReportAgent further support both instance-level critique and cross-case aggregation, producing actionable analysis reports. Experiments on **TraceBench** show that TraceSIR generates high-quality reports aligned with practical research and engineering needs. Beyond report quality, gold-grounded evaluations confirm that its diagnoses are correct rather than merely well formatted, while report-guided re-solving demonstrates their actionability by recovering markedly more failed tasks than a generic retry.

AI USE STATEMENT

In this work, we used generative AI tools as part of the studied system and experimental pipeline. Specifically, TraceSIR and the evaluated baselines rely on large language models to process agentic execution traces, generate structured diagnostic outputs, and produce analysis reports used in our experiments. We also used generative AI tools to assist with writing polish, including grammar correction, wording refinement, and improving clarity of presentation.

We did not use generative AI tools for literature retrieval or discovery, research ideation, mathematical proof generation, or drafting scientific claims beyond language-level editing. We also did not use generative AI tools to create synthetic benchmark tasks; TraceBench is constructed from existing benchmarks and collected execution traces as described in the paper.

All AI-assisted experimental outputs were evaluated through the protocols described in the paper, including `ReportEval`, expert human assessment, gold-grounded diagnostic-correctness evaluation, factual-grounding audit, and report-guided re-solving. The authors reviewed the AI-assisted writing edits and verified the experimental procedures, reported results, and conclusions. We take responsibility for the final content of this work, including all text, claims, results, and artifacts produced with the aid of generative AI.

REFERENCES

OFM Riaz Rahman Aranya and Kevin Desai. Trace: Temporal radiology with anatomical change explanation for grounded x-ray report generation, 2026. URL https://arxiv.org/abs/2602.02963.

Marc Aubreville, Taryn A. Donovan, and Christof A. Bertram. Exploring general-purpose autonomous multimodal agents for pathology report generation, 2025. URL https://arxiv.org/abs/2601.11540.

Elad Ben Avraham, Changhao Li, Ron Dorfman, Roy Ganz, Oren Nuriel, Amir Dudai, Aviad Aberdam, Noah Flynn, Elman Mansimov, Adi Kalyanpur, and Ron Litman. Dream: Deep research evaluation with agentic metrics, 2026. URL https://arxiv.org/abs/2602.18940.

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

Yanyu Chen, Jiyue Jiang, Jiahong Liu, Yifei Zhang, Xiao Guo, and Irwin King. Trace: Trajectory-aware comprehensive evaluation for deep research agents, 2026. URL https://arxiv.org/abs/2602.21230.

Yun-Shiuan Chuang, Chaitanya Kulkarni, Alec Chiu, Avinash Thangali, Zijie Pan, Shivani Shekhar, Yirou Ge, Yixi Li, Uma Kona, Linsey Pang, and Prakhar Mehrotra. Toward scalable verifiable reward: Proxy state-based evaluation for multi-turn tool-calling llm agents, 2026. URL https://arxiv.org/abs/2602.16246.

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

Shuyu Gan, Renxiang Wang, James Mooney, and Dongyeop Kang. A2p-vis: an analyzer-to-presenter agentic pipeline for visual insights generation and reporting, 2025. URL https://arxiv.org/abs/2512.22101.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei

Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Jin Jia, Zhiling Deng, Zhuangbin Chen, Yingqi Wang, and Zibin Zheng. Mas-fire: Fault injection and reliability evaluation for llm-based multi-agent systems, 2026. URL https://arxiv.org/abs/2602.19843.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Juampablo E. Heras Rivera, Dickson T. Chen, Tianyi Ren, Daniel K. Low, Asma Ben Abacha, Alberto Santamaria-Pang, and Mehmet Kurt. Btreport: A framework for brain tumor radiology report generation with clinically relevant features, 2026. URL https://arxiv.org/abs/2602.16006.

Marco Salmè, Federico Siciliano, Fabrizio Silvestri, Paolo Soda, Rosa Sicilia, and Valerio Guarrasi. Concept-enhanced multimodal rag: Towards interpretable and accurate radiology report generation, 2026. URL https://arxiv.org/abs/2602.15650.

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, 2025. URL https://arxiv.org/abs/2504.12516.

Ruoqing Zhao, Runze Xia, and Piji Li. Hallucination mitigating for medical report generation, 2026. URL https://arxiv.org/abs/2601.15745.

A TRACEBENCH CONSTRUCTION

For BrowseComp, Tau2Bench, and SWE-bench, we run GLM-4.6[5] as the target agent model on the official test split and retain complete execution traces in the standard OpenAI message format. Task performance is evaluated with the official protocol of each benchmark, yielding a binary `gold_score`. To focus on diagnostic analysis of agent failures, we randomly sample 50 failed cases from each benchmark, where `gold_score`=0. TraceBench therefore contains 150 failed task instances across three agentic scenarios. These traces serve as the unified input set for evaluating structured trace analysis and reporting with TraceSIR.

[5]https://docs.bigmodel.cn/cn/guide/models/text/glm-4.6

B Experimental Settings

We evaluate TraceSIR under a controlled experimental setting. The length threshold θ for trace abstraction is set to 100 words or 1,000 characters. During processing, report generation is automatically triggered whenever the number of processed cases reaches a multiple of 10. For evaluation consistency, we compare only the final reports generated after all 50 cases in each scenario have been processed. Thus, all methods operate on the same 50 failed traces per scenario and produce one final report per scenario.

In practical deployment and in our demonstration system, TraceSIR additionally enforces final report generation regardless of the number of input cases. This ensures that users always receive a reference analysis report at the end of the workflow, even when only a single trace is submitted.

For report evaluation, we use a hybrid setting combining expert human assessment and automated LLM-based judging. Six domain experts with extensive experience in agentic systems evaluate the generated reports under the `ReportEval` protocol. The final human score is obtained by averaging expert ratings, with each scenario evaluated by at least two experts. We also conduct LLM-as-a-judge evaluation under `ReportEval` using GPT-5[6]. The judge is run three times and the scores are averaged to improve stability, providing a scalable complement to human evaluation.

C Baseline Setup and Fair-Comparison Protocol

We compare TraceSIR with general-purpose agentic systems, including ClaudeCode as the main baseline and Codex and OpenCode as additional baselines in Figure 4. At the time of writing, there is no established open system dedicated to cross-case agentic Trace Critique. These systems are therefore strong practical alternatives, since they can read trace files, use file-level tools, and produce analysis reports. To ensure a controlled and reproducible comparison, all baselines are evaluated under the same protocol:

- **Identical inputs.** Each baseline reads the same directory of 50 execution traces per scenario, stored as the same `TraceBench-*.json` files with the same `TraceFormat` fields and read-only access. No baseline receives any trace, field, or evaluator signal that TraceSIR does not receive, and vice versa.
- **Identical task and output contract.** All systems receive the same instruction to analyze 50 failed traces, ground claims in specific cases and steps, include quantitative error statistics, and produce a Markdown report containing an overall summary, failure patterns, root-cause analysis, and optimization suggestions.
- **Backbone control.** TraceSIR, ClaudeCode, and OpenCode use Claude-4.6 as the backbone where reported, while Codex uses GPT-5.5. This setup helps separate framework effects from backbone effects.

We release the exact prompts, tool configurations, and generated baseline reports. Under this protocol, the main experimental variable is the analysis framework itself, so the reported gains are attributable to TraceSIR's design, including structured abstraction, per-case diagnosis, and cross-case aggregation, rather than to input, prompt, or tool differences.

D Effect of Trace Abstraction

To isolate the contribution of StructureAgent, we measure the size of every structured trace before and after length-aware abstraction over all 300 processed traces. Abstraction reduces the analyzed trace length from 47.6K to 7.6K tokens on average, corresponding to an 84% reduction. The factual-grounding results in Figure 3 and the diagnostic-correctness results in Table 3 show that this compression preserves the diagnostic signal. This abstraction is also necessary for long-horizon analysis. Without abstraction, 24 out of 300 traces exceed a 128K-token context window and cannot be processed in a single pass. This issue is most pronounced in Deep Research, where traces average 111.7K tokens and reach 289.8K tokens. StructureAgent is therefore not incidental preprocessing,

[6] https://developers.openai.com/api/docs/models/gpt-5

Component	LLM calls	Tokens
StructureAgent reasoning	1,489	5.39M
InsightAgent reasoning	1,306	2.94M
ReportAgent reasoning	929	2.72M
Tool inputs and outputs	–	23.06M
Total for 150 traces	3,724	34.10M
Average per trace	24.8	0.23M

Table 5: End-to-end cost of TraceSIR on TraceBench.

but a prerequisite for reliable analysis of long execution traces. This also explains why raw-trace or outcome-only baselines degrade most on the longest cases.

E Cost and Efficiency

Table 5 reports the end-to-end cost of running TraceSIR on the full 150-trace TraceBench. The three agents issue 3,724 LLM calls and consume 34.1M tokens, including 11.0M reasoning tokens and 23.1M tool input-output tokens. This corresponds to 24.8 calls and 227K tokens per trace on average, with an estimated cost of $1.09 per trace and $164 in total. The full run takes approximately 33 hours of wall-clock time.

StructureAgent accounts for the largest share of reasoning tokens, consistent with its role in compressing long traces before downstream diagnosis and reporting. A three-agent critique pipeline is naturally more expensive than a single outcome-only judgment, but the cost is modest relative to the engineering time required for manual cross-case debugging. In deployment, this cost can be further reduced by assigning trace abstraction to a cheaper backbone.

F TraceSIR Demonstration

Figure 5 presents an end-to-end demonstration of TraceSIR, illustrating how the system supports structured analysis and reporting of agentic execution traces in practice. The interface supports model configuration, trace submission, progress monitoring, result retrieval, task rerunning, log inspection, interactive report refinement, and documentation access.

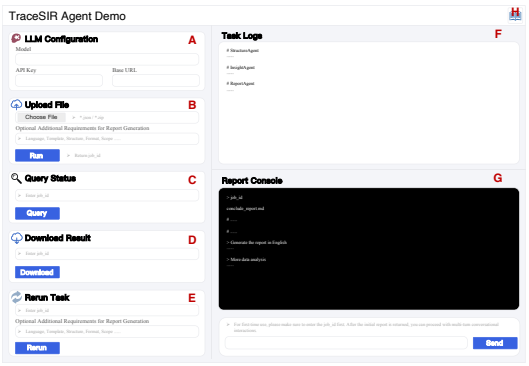

Figure 5: TraceSIR demonstration interface.

These components provide a complete workflow for practical Trace Critique. Users can configure LLM backends, submit single or batched trace files, monitor analysis progress, resume interrupted jobs, inspect execution logs, download generated reports, and refine reports through interactive follow-up. The demonstration is designed to support transparent, reproducible, and user-controllable trace analysis.

G Example Analysis Report

To illustrate the qualitative characteristics of TraceSIR reports, Figure 6 shows a representative excerpt from an analysis report. The report is generated in Chinese, reflecting practical deployment settings and user requirements. The complete reports and all related data are publicly available.

H Capability Comparison with ClaudeCode

To contextualize the design choices and empirical results of TraceSIR, we provide a capability-level comparison with ClaudeCode, a strong baseline for agentic analysis. Table 6 summarizes the key

	Capability	TraceSIR	ClaudeCode
System Design	1. Multi-agent architecture explicitly designed for analysis and reporting, with separated roles for trace structuring, diagnosis, and reporting	✓	△
	2. Fine-grained, per-case diagnostic outputs with structured fields	✓	△
	3. Native cross-case aggregation and statistical analysis over all traces	✓	△
	4. Incremental analysis with resume support for large trace collections	✓	×
	5. Configurable analysis and report constraints via explicit settings	✓	△
Methodology	1. Full execution trace modeling with explicit Thought–Action–Observation	✓	×
	2. Structured trace abstraction preserving causal and temporal dependencies	✓	×
	3. Insight-driven root cause analysis beyond surface-level error description	✓	△
	4. Cross-step causal reasoning over long-horizon agent behaviors	✓	×
	5. Failure-centric analysis designed for systematic diagnosis	✓	△
Reporting	1. Multi-dimensional analysis reports with explicit diagnostic categories	✓	△
	2. Trace-level evidence linking conclusions to specific execution steps	✓	×
	3. Actionable optimization suggestions grounded in diagnosed root causes	✓	△
	4. Generation of targeted SFT samples for downstream model improvement	✓	×
	5. Robust report quality under weaker backbone language models	✓	×

Table 6: Capability-level comparison between TraceSIR and ClaudeCode across system design, methodology, and reporting. ✓ denotes native and systematic support, △ denotes partial or non-systematic support, and × denotes lack of support.

differences between the two systems across system design, methodology, and reporting capabilities. The comparison highlights TraceSIR's advantages in trace-centric diagnosis, structured evidence preservation, and cross-case reporting.

I REPORTEVAL RUBRIC AND PROTOCOL

For reproducibility, we document the full `ReportEval` scoring rubric in Table 7. Each report is scored on five dimensions using an integer scale from 0 to 10. The overall score is the equally weighted sum normalized to a range from 0 to 100. Evaluation is performed through two independent channels: six domain experts and an LLM judge using GPT-5 with scores averaged over three runs. Each scenario is rated by at least two experts, and all experts are blinded to method and backbone identity. We assign equal weights to the five dimensions to reflect their shared importance in practical trace analysis. To avoid conflating format compliance with diagnostic correctness, we evaluate correctness separately through the gold-grounded factual-grounding audit in Figure 3 and the diagnostic-correctness study in Table 3. We release the full rubric, judge prompt, per-rater scores, and agreement statistics with the code.



```
# 模型智能体 Bad Case 深度分析报告
---
## 1. 全局概览
.....
## 2. 常见错误分析
### 2.1 错误类型分布总览

| 错误大类 | 涵盖的 key_error 标签 | 案例数 | 占比 |
|---|---|---|---|
| 理解偏差 | 理解偏差(4)、概念理解偏差(1)、概念混淆错误(1) | 6 | **12%** |
| 确认偏误/过早锁定 | 锁定错误(2)、锚定偏误(1)、错误定位(1) | 4 | **8%** |
| 搜索策略失效 | 策略混乱(2)、搜索失焦(1)、搜索不精准(1)、检索策略失败(1) | 6 | **12%** |
| 验证不足/逻辑断裂 | 验证不足逻辑混乱(1)、证据不足(1)、验证过度(1)、浅层规止(1) | 4 | **8%** |
| 时间推理错误 | 时间判断失误(1)、时间验证不足(1)、时间验证失误(1)、时间焦实失误(1)、时间逻辑混乱(1) | 5 | **10%** |
| 目标丢失/未闭环 | 目标丢失(1)、未提交答案(1)、未填参数(1)、半途而废(1)、偏离主线(1) | 5 | **10%** |
| 信息整合失败 | 整合推理不足(1)、搜索整合失败(1)、关键断裂(1)、检索逻辑断裂(1) | 4 | **8%** |
| 识别匹配错误 | 识别不准确(1)、身份识别错误(1)、目标识别错误(1)、识别笔名(1)、识别计算错误(1) | 5 | **10%** |
| 其他（谜题解码、距离误判等） | 谜题解码失败(1)、距离误判(1)、坐标选取错误(1)、平均当时点(1)等 | 11 | **22%** |

### 2.2 十大错误趋势深度分析
#### **趋势 1：硬约束条件被软化处理（占比 ~30%，15/50 案例）**
**模式描述**：当任务包含精确的数值、时间或逻辑约束时，智能体倾向于将这些"必须满足"的硬性条件降级为"尽量满足"的软性偏好。
**典型案例**：
- **TraceBench-25**：题目明确要求"2015 年前发行"，但智能体最终给出 2020 年播出的《Tell Me What You Saw》，并赋予 95% 置信度，完全忽视时间硬约束
- **TraceBench-12**：题目要求德州分店距 Town East Mall **5.1 英里**，实际验证为 7.6 英里，智能体试图通过改变测量方式"强行匹配"
- **TraceBench-23**：题目要求"1989-1992 年间组建乐队"，但 Loose Change 乐队官网明确标注 1988 年成立，智能体引用第三方平台"est. 1990"来合理化矛盾
**根因分析**：智能体缺乏约束分类机制，将所有条件视为等权重的"特征匹配项"，用累积评分替代了布尔逻辑的"全部满足"要求。当多数软条件匹配时，个别硬约束的违反被概率性地"稀释"。
.....
## 3. 分数分布分析
.....
## 4. 模型缺点模式
基于 disadvantages 字段的系统性分析，归纳出以下六大典型缺点模式：
### 模式 A：验证闭环缺失（出现在 ~80% 的案例中）
.....
## 5. 根因分析与洞察生成
### 洞察 1：概率匹配范式与逻辑验证需求的根本性冲突
.....
## 6. 结论与建议
### 6.1 核心发现总结
### 6.2 优化建议
#### 短期可执行措施（Prompt 工程层）
#### 中期架构改进
#### 长期训练策略
### 6.3 趋势预测
.....
## 附录：TraceBench 处理后的数据
### TraceBench-1
.....
```

Figure 6: Excerpt from an analysis report generated by TraceSIR using Claude-4.6 on the BrowseComp Deep Research scenario.

Dimension scored 0–10	Criteria for a high score
Overall Structure	Clear and coherent organization with essential sections present; descriptions are objective and grounded in the analyzed traces.
Error Analysis	Identified agent errors are accurate and explicitly supported by specific trace steps rather than fabricated.
Root-Cause Analysis	Explanations of why failures occur are insightful, well justified, and go beyond surface-level symptom description.
Optimization Analysis	Suggestions are relevant, feasible, and logically tied to the diagnosed causes, providing actionable guidance.
Overall Impact	The report provides practical value for understanding agent behavior and supporting research or engineering decisions.

Table 7: The five `ReportEval` dimensions and their scoring criteria. The overall score is computed as $\frac{100}{50}\sum$ over the five dimension scores.

J Qualitative Case Study

We illustrate the downstream mechanism with a representative Function Calling case from tau2 retail. The original agent scored 0, and the `+Generic` re-solve also scored 0, while `+TraceSIR` scored 1. The user asked to cancel the office items of a pending order. The agent found the item prices but, after concluding that the system did not support partial cancellation, pivoted to changing the address without communicating the refund amount of $1093.34. As a result, the tau2 evaluator's `communicate` check failed. TraceSIR diagnoses this core error precisely. The report identifies that the agent failed to tell the user the partial-cancellation refund amount, computed as $235.13 + $511.24 + $346.97 = $1093.34, and explains that the `communicate` check failed because the value 1093.34 was never conveyed. Given this diagnosis, the re-solved agent computes and states the refund amount, satisfying the communication criterion and resolving the task. The guidance succeeds not by revealing an answer key, but by identifying the specific behavioral omission that caused the failure. This is the type of actionable signal that a quality-only report rubric cannot fully capture. For contrast, downstream gains are smaller on Deep Research, as shown in Table 4. In that setting, diagnoses may be accurate, but recovery often requires gathering new external evidence that a post-hoc report cannot directly supply.

K Limitations

Despite its effectiveness, TraceSIR has several limitations that also motivate future work. First, TraceSIR relies on LLM-based agents, so analysis quality depends on the underlying models, especially in challenging domains such as complex coding. Structured abstraction and gold-grounded validation reduce this risk, but a fully model-agnostic guarantee remains open. Second, TraceBench currently contains failed traces generated by a single source model, GLM-4.6. Our gold-grounded correctness study evaluates whether diagnoses are correct, but it does not test false-positive behavior on successful traces, calibration on partially successful runs, or generalization to traces from other source models. Extending TraceBench with successful cases, partial-success cases, and multi-model traces is an important next step. TraceSIR already ingests standard OpenAI-format traces, so this is primarily a data-collection limitation rather than a methodological one. Third, the main report evaluation uses one generated report per condition scored by multiple raters. This characterizes evaluator agreement but not generation variance. The factual-grounding and LLM-judge scores are already averaged over three runs, and estimating multi-generation report-level variance is planned. Fourth, we provide direct downstream evidence through report-guided re-solving across all three scenarios. A stronger form of validation would be to fine-tune on TraceSIR's optimization samples and re-measure benchmark performance, which remains a natural next step built on the same actionable-diagnosis premise. Finally, cross-case reporting currently assumes a moderate number of instances and may require hierarchical or incremental aggregation to scale to thousands of traces. TraceSIR may also incur nontrivial latency, token cost, and output variability, as reflected in Table 5. Reducing cost, latency, and variance while integrating TraceSIR as a service, skill, hook, or CLI within existing agent harnesses remains important future work.