INSIGHTS GENERATOR: SYSTEMATIC CORPUS-LEVEL TRACE DIAGNOSTICS FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diagnosing failures in LLM agents remains largely manual. Practitioners inspect a small subset of execution traces, form ad-hoc hypotheses, and iterate. This process misses patterns that only emerge across trace populations and does not scale to production corpora where individual traces span tens of thousands of tokens. We formalize the problem of corpus-level trace diagnostics. Given a corpus of execution traces, the goal is to produce grounded natural-language insights that characterize systematic behavioral patterns across trace groups, each linked to supporting evidence. We present the **Insights Generator (IG)**, a multi-agent system that answers diagnostic questions by proposing and testing hypotheses across the trace corpus to produce an evidence-backed insights report. We evaluate IG across qualitative and objective dimensions, spanning rubric-based report assessment and downstream performance improvements achieved by implementing IG insights. Human experts using IG reports improve scaffold performance by 30.4 pp over the unmodified baseline scaffold, and coding agents leveraging IG-derived insights show consistent and stable gains. Across benchmarks, IG's *scout-investigator* architecture produces findings comparable in detection coverage to competing approaches, while domain experts rated IG reports as leading depth and evidence quality.

1 INTRODUCTION

Agentic systems produce long execution traces of reasoning steps, tool calls, and environment interactions. Debugging typically starts from evaluation results that flag failed runs; practitioners then inspect a few traces and form hypotheses (Deshpande et al., 2025; Nanda et al., 2026). Evaluations are defined over final task outcomes, so they yield aggregate signals but reveal neither intermediate behaviors nor patterns across the broader space of scenarios, and many errors, inefficiencies, and performance differences stay hard to identify (Deshpande et al., 2025; Ma et al., 2025). This is especially true for failures localized to specific subsets of inputs, task types, or execution contexts that are not individually prominent enough to noticeably shift aggregate metrics. Such patterns are invisible to standard evaluation unless the practitioner already knows to look for them.

Many of these signals emerge only at the cohort level. Systematic differences in tool usage, recurring reasoning structures, and patterns correlated with strong or weak performance become visible only when traces are analyzed collectively. Cohort comparison is a particularly powerful form of this analysis: examining groups of traces that vary along a single controlled axis, such as model variant, prompt version, or task category, isolates the behavioral differences attributable to that axis and provides a stronger targeted diagnostic signal than aggregate metrics alone. Beyond diagnosis of known failure modes, there is a distinct need for open-ended *discovery*. Silent failures, behavioral trends that reduce reliability without triggering explicit error signals, are a key example; they rarely appear in evaluation dashboards and require population-level analysis to detect. We refer to such findings as **trace insights**: grounded natural-language statements characterizing salient patterns across groups of traces, each supported by trace-level evidence. For example, a representative IG insight on HLE might read: `84% of incorrect Python-using traces exhibit 'silent computation failures': the code executes without errors but implements the wrong mathematical model. Cross-checking fails to catch these because it verifies internal consistency within the wrong framework rather than challenging the framework itself.`

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.

```
Observed in 94/112 incorrect Python-using traces (37.6% of all
traces); 0/19 cross-checking traces detected the modeling error.
```

Surfacing trace insights at corpus scale presents two challenges. **Scale and Complexity:** Trace corpora, the collections of traces gathered across many runs of an agent, routinely exceed any single model context window even when each individual trace fits within it. Individual traces interleave thousands of tokens of reasoning, tool calls, and environment state. The patterns of interest (silent failures, systematic inefficiencies, behavioral differences between cohorts) are semantic and not reducible to keyword matching or predefined taxonomies (Bertsch et al., 2025; Maekawa et al., 2025). Prior work addresses fragments of this problem through single-trajectory debugging and failure attribution (Zhu et al., 2025; Ma et al., 2026; Barke et al., 2026; Deshpande et al., 2025; Zhang et al., 2025c), classification under fixed failure taxonomies (Cemri et al., 2025; Ma et al., 2025), outcome-conditioned distillation of lessons into skills (Ni et al., 2026), harness optimization in which a proposer reads the raw traces of prior candidates (Lee et al., 2026), or recursive search over trace snippets (Context Labs, 2026); to our knowledge, none separates hypothesis proposal on sampled traces from statistical validation over the full corpus (Section 2). **No Evaluation Standard:** Trace insights are free-form claims about population-level behavior; whether such a claim is correct, grounded in cited evidence, and useful to a downstream practitioner cannot be reduced to a single existing metric, and no shared methodology has been proposed.

We present *Insights Generator (IG)*, which addresses the first challenge through iterative hypothesis-driven aggregation: it decomposes an open-ended diagnostic question into testable hypotheses, validates each at corpus scale, and synthesizes evidence-backed findings. Our contributions are:

- **Iterative corpus-level trace analysis.** IG separates hypothesis generation from corpus-scale validation and routes all trace access and processing through a stateful Python layer, enabling a decompose-hypothesize-validate loop that operates over corpora exceeding any single context window. This surfaces patterns, such as the silent computation failures above, that single-pass and per-trajectory methods miss (Section 2).
- **Evaluation framework for insights.** We introduce a four-setting framework that varies who evaluates (calibrated LLM judge vs. human expert) and what is measured (report quality vs. downstream scaffold impact), and apply it to a range of trace-analysis systems on multiple benchmarks. Rankings are consistent across the four settings and across LLM judges from different model families, providing the first systematic basis for comparing corpus-level trace diagnostic systems (Section 4).
- **Practitioner and automated impact.** Subject matter experts using IG reports improve scaffold performance by 30.4 pp over the unmodified baseline, nearly doubling the 16.2 pp gain from the strongest alternative system ($p = 0.016$). This gap is consistent with IG's higher scores on evidence depth and mechanism explanation in the automated evaluation, and establishes that diagnostic quality translates directly to practitioner effectiveness (Section 4.2). In an automated patching loop on APEX-Agents, IG reports produce the largest held-out gain of any report source, and IG report quality rises with analysis budget: citation validity and judged quality increase across the capped budget tiers under both judge families (Sections 4.4 and 4.5).

2 Related Work

Single-Trajectory Error Analysis. Most agent debugging work analyzes one trajectory at a time, often by counterfactual replay: AgentDebug (Zhu et al., 2025) reasons about alternative decision points, AgenTracer (Zhang et al., 2026) builds failure-attribution data through counterfactual replay and programmatic fault injection and trains a lightweight attribution model, and DoVer (Ma et al., 2026) replays trajectories with targeted interventions. AgentRx (Barke et al., 2026) infers constraints from tool schemas and checks whether agent actions satisfy them. Taxonomy-based methods (MAST (Cemri et al., 2025), AgentFail (Ma et al., 2025)) classify known failure types but cannot surface modes the taxonomy did not anticipate, and failure-attribution benchmarks such as Who&When (Zhang et al., 2025c) ask which agent and which step caused a given failure. Per-trace diagnosis remains hard: on TRAIL (Deshpande et al., 2025) the best models score only 11%. Some, such as DoVer, already pair failure hypotheses with active tests, but by intervention on one trajectory

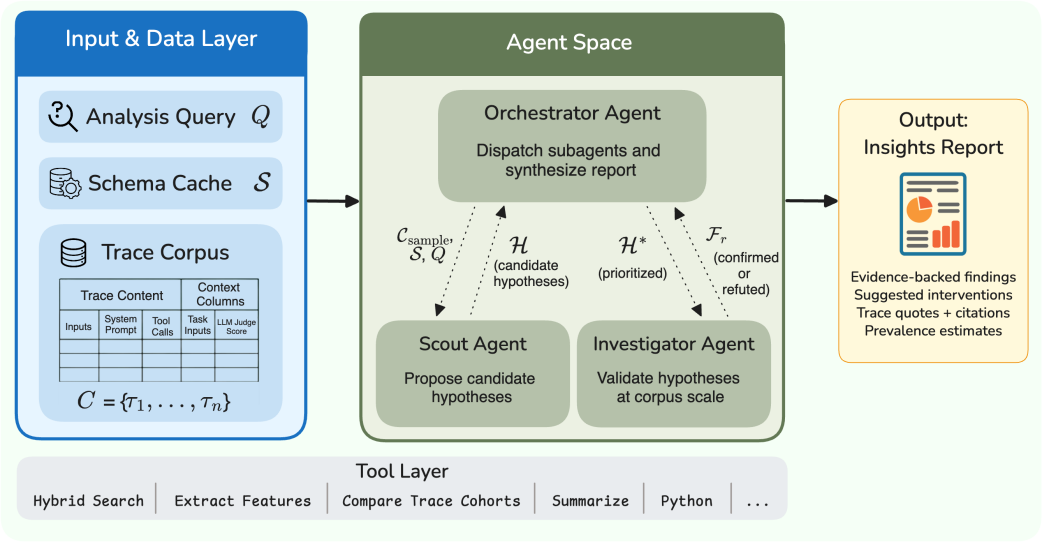

Figure 1: IG system overview. **Left:** inputs are a diagnostic question Q and a trace corpus $\mathcal{C}$, from which IG builds a processed trace store with schema cache $\mathcal{S}$. **Center:** the **Orchestrator** dispatches **Scouts** ($\mathcal{H}$: hypothesize over sampled traces) and **Investigators** ($\mathcal{H}^*$: validate prioritized hypotheses via corpus-scale cohort comparison). Investigators return findings $\mathcal{F}_r$, which the Orchestrator synthesizes into the final report. **Right:** an evidence-backed report with findings, fixes, citations, and prevalence estimates. **Bottom:** the shared tool layer. Algorithm 1 formalizes the loop.

at a time. These trajectory-focused methods excel at individual remediation, but miss systemic patterns that only emerge across corpora of agent runs, which is the gap our corpus-level approach addresses.

Agentic Insight Generation. More recent work has leveraged agent-based decompositions to surface insights from agent execution traces. Trace2Skill (Ni et al., 2026) dispatches a flat parallel fleet of analyst sub-agents over individual trajectories to identify skill patches, and then performs hierarchical consolidation to merge individual patches into a conflict-free skill directory consumed by the agent at inference time. HALO (Context Labs, 2026) employs a Recursive Language Model (Zhang et al., 2025a) that analyzes traces through tools like search, view, count, and code, and recursively invokes itself on the relevant snippets, producing a diagnostic report handed to a downstream coding agent for harness fixes. IG differs in two ways: it uses typed hypothesis-generation and validation roles rather than a flat or recursive decomposition, separating sample-level pattern proposal from corpus-scale statistical validation, and it routes all trace access through a stateful Python data processing layer, so findings reflect corpus-level distributions rather than per-trajectory observations.

Agent Optimization Frameworks. Several recent works frame agent improvement as an automated optimization task. VeRO (Ursekar et al., 2026) introduces an evaluation harness for *agent optimization*: a coding agent iteratively modifies a target agent's implementation, prompts, tools, and orchestration logic, and is evaluated on the lift it achieves against a fixed benchmark. AFlow (Zhang et al., 2025b) and ADAS (Hu et al., 2025) similarly explore automated search over agent-as-code representations. Meta-Harness (Lee et al., 2026) delegates both diagnosis and proposal to a single coding-agent proposer that reads the source code, scores, and execution traces of all prior candidates through a filesystem, rather than externalizing the improvement logic in a hand-designed search loop. These systems treat the agent as an artifact to be improved against a specific evaluation set. Our work is complementary: rather than optimizing an agent toward a fixed target, IG surfaces behavioral insights latent in a trace corpus, including patterns that no predefined objective would have targeted. These insights can then be consumed by an agent-optimization framework in a closed loop.

3 INSIGHTS GENERATOR

Overview. Insights Generator (IG) is a multi-agent analysis system for discovering and validating behavioral patterns across large trace corpora without requiring full-context loading of raw trajectories. The system is motivated by a practical bottleneck in agent debugging: manual inspection of small numbers of traces may be sufficient to form hypotheses, but is insufficient to validate them at scale. IG addresses this by separating hypothesis discovery from hypothesis validation, and by routing all analysis through a Python data processing layer rather than passing raw trace content to LLM context directly. Agents interact with traces exclusively through callable tools (summarization, extraction, cohort comparison) that return only aggregated results to model context. Figure 1 illustrates the system; Algorithm 1 (Appendix A.1) formalizes the multi-round loop.

Task definition. The input is a diagnostic question Q and a corpus of agent execution traces $\mathcal{C} = \{\tau_1, \ldots, \tau_n\}$, each interleaving inputs, reasoning steps, tool calls, environment observations, and outputs, optionally with metadata such as task-specific model inputs, outcome labels, configuration variants, or timestamps. The output is a set of corpus-level insights describing salient behavioral patterns across subsets of $\mathcal{C}$, each supported by trace-level evidence. Unlike skill distillation (Ni et al., 2026) or harness optimization (Lee et al., 2026), the output is itself the object of evaluation: each insight states a prevalence over a named cohort and cites trace IDs, so both can be checked deterministically against the corpus. With labels, insights can characterize systematic differences between cohorts (e.g., success versus failure); without them, the task reduces to unsupervised pattern discovery.

Agent roles. The **Orchestrator** is the sole user-facing agent. It decomposes the objective, dispatches subagents in parallel, judges whether findings suffice across rounds, and synthesizes the final report, inspecting traces minimally to keep its context free for coordination. A **Scout** explores a representative trace sample with LLM summarization and extraction tools and proposes diverse testable hypotheses; its objective is breadth, not validation. An **Investigator** receives one hypothesis, validates it at corpus scale with LLM extraction and programmatic analysis (distributional statistics, cohort comparisons), and returns a finding labeled confirmed, refuted, or inconclusive with quantitative evidence. Prompts are in Appendix A.2; models and runtime settings are in Appendix A.3.

Data layer. Agents operate over a processed trace store rather than raw text, so they can apply arbitrary data-science transformations in Python. Traces and extracted features share a short string identifier rather than a UUID, which reduces token usage; consistent with prior observations that LLMs mis-copy long random identifiers and that copying fidelity degrades with identifier length (Hsieh et al., 2024; Dai et al., 2024), we also observe fewer hallucinated references with short identifiers. A schema cache $\mathcal{S}$ of per-column statistics, null rates, value distributions, and notable correlations is computed automatically and injected into every subagent's context at dispatch. A hybrid semantic-keyword vector store over the traces serves content retrieval to all agents as a tool. Section 4.5 reports how removing components of this layer affects downstream scaffold improvement.

Programmatic tool use. Each subagent works in a stateful Python environment with IG's analysis primitives pre-injected as functions; to the agent, the interface is a single code-execution tool. The primitives cover *trace inspection* (loading traces, retrieving raw content, querying the schema cache, hybrid search), *LLM-driven analysis* (structured feature extraction, per-trace and group-level summarization, A/B cohort comparison), and *data management* (persisting computed columns and finding-supporting cohorts, refreshing the cached DataFrame, consolidating extraction parquets). The Orchestrator keeps a coordination subset, the Scout adds extraction to that subset, and the Investigator further adds cohort comparison and finding persistence. Agents chain operations in one code block, so intermediate trace content never enters model context. Appendix A.4 lists every primitive and its per-role availability.

Iterative analysis loop. At each turn the Orchestrator inspects the candidate hypotheses $\mathcal{H}^*$ and confirmed findings $\mathcal{F}$ and takes one of three actions: dispatch a parallel batch of Scouts to surface new hypotheses, dispatch a parallel batch of Investigators to validate prioritized hypotheses at corpus scale, or submit the final report. Scout and Investigator rounds can interleave in any order and number, depending on coverage. When the Orchestrator judges that no more information is needed,

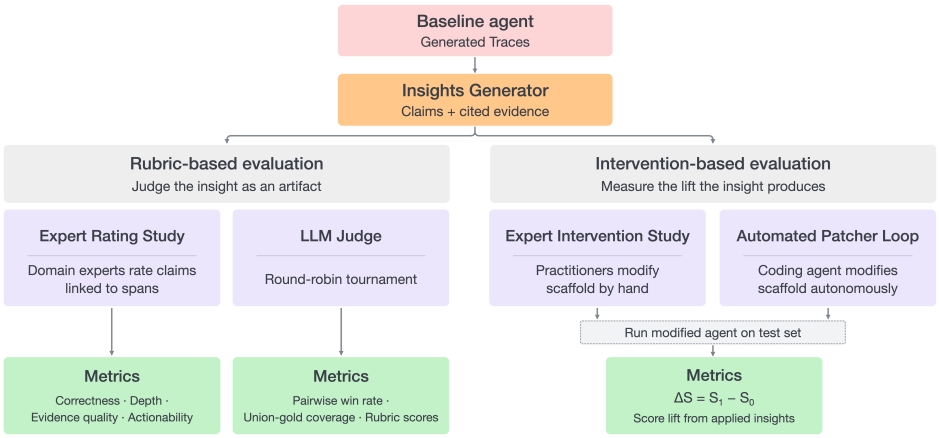

Figure 2: The four evaluation settings. Rubric-based: the Expert Rating Study and the LLM judge score the report as an artifact. Intervention-based: the Expert Intervention Study and the automated patcher loop measure the lift a report produces when practitioners or a coding agent modify the scaffold.

it synthesizes the confirmed findings from all rounds into the report. Algorithm 1 (Appendix A.1) formalizes the loop.

4 Evaluation

We evaluate IG along two axes, *who evaluates* (an LLM judge or human experts) and *what is measured* (report quality or downstream scaffold impact), which gives four settings (Figure 2): the LLM judge (Section 4.1); the **Expert Intervention Study**, in which experts modify a scaffold given a report (Section 4.2); the **Expert Rating Study**, in which experts score individual insights (Section 4.3); and the automated patcher loop (Section 4.4). Because IG is meant to support agent developers rather than replace them, the Expert Intervention Study is our primary measure of downstream utility; the patcher loop is a complementary test of end-to-end insight transfer. Section 4.5 reports component ablations and cost scaling.

We use three benchmarks (Appendix A.5): **SSB** (Ma et al., 2024; Shortcut Research Team, 2025), spreadsheet-manipulation tasks with a deterministic cell-match grader and the primary benchmark for the human studies; **HLE** (Phan et al., 2026), expert-level academic questions with an LLM grader; and **APEX-Agents** (Vidgen et al., 2026), long-horizon, document-grounded professional tasks with roughly 13× more tool calls per trace than SSB, which serves as the harder second domain for the judge, intervention, and patcher settings.

4.1 Report Quality under an LLM Judge

We compare IG with four systems given the same corpora and the same canonical diagnostic question: **Single-Agent Coding** (Claude Code, Anthropic, 2025, with its native tool surface); **CC Subagents**, which adds subagent dispatch and IG's scout-investigator split without IG's data layer; a **Trace2Skill**-style system (Ni et al., 2026) with flat parallel analysts and hierarchical consolidation; and **RLM** (Zhang et al., 2025a), a vanilla Recursive Language Model at maximum depth 2. Together these vary tool surface, multi-agent topology, and role decomposition. A shared module normalizes all reports to one JSON schema, and every SSB and HLE metric is a mean over three report replicates per system, with one report per system on APEX-Agents (Appendix A.6).

Protocol. The judge pools all systems' findings and clusters them into a *union-gold* set of canonical observations. It scores each report on **coverage**, the fraction of clusters for which it cites at least one member trace, and on a 0–3 rubric per dimension against the clusters (Appendix A.7); Table 1

Table 1: LLM-judge results (Claude Opus 4.6 judge) on SSB, HLE, and APEX-Agents. **Pairwise WR**: win rate (%) in a position-swapped round-robin between reports; Avg: mean over the three benchmarks. Win rates are computed within a 12-configuration tournament pool on SSB and HLE and an 8-configuration pool on APEX-Agents, where one report per system was judged (Appendix A.8); n.j.: not judged on APEX-Agents (Trace2Skill and Single-Agent Coding; Section 5). **Quality**: 0–3 rubric scores against union-gold clusters for mechanism (Mech.), specificity (Spec.), actionability (Act.), and their mean (Comp.), averaged over SSB and HLE; APEX: composite on APEX-Agents. Bold: best per column.

	Pairwise WR (%)				Quality (0–3)				
System	SSB	HLE	APEX	Avg	Mech.	Spec.	Act.	Comp.	APEX
IG	**72.4**	**83.3**	**85.7**	**80.5**	**1.34**	**1.34**	**1.27**	**1.32**	**1.50**
CC Subagents	63.3	61.4	71.4	65.4	1.13	1.11	1.07	1.10	1.47
Trace2Skill	56.2	49.0	n.j.	n.j.	1.18	1.14	1.23	1.18	n.j.
RLM	13.8	9.0	21.4	14.7	0.96	0.95	0.84	0.92	1.13
Single-Agent Coding	8.6	12.4	n.j.	n.j.	0.78	0.77	0.73	0.76	n.j.
IG architecture ablations									
IG, Generic Subagents	59.0	76.0	78.6	71.2	1.14	1.13	1.03	1.10	1.47
IG, Orchestrator-only	35.0	29.0	28.6	30.9	0.87	0.87	0.82	0.85	0.92

reports mechanism, specificity, and actionability. Because the union-gold reference is built from the evaluated systems' own pooled findings, coverage is not an independent measure, so we omit it from Table 1 and report it in Appendix A.8. Separately, a **pairwise** round-robin tournament asks the judge to prefer one of two reports or declare a tie, and computes win rate as $(\text{wins} + 0.5 \times \text{ties})/n_{\text{rounds}}$ with no gold reference. Because the judge shares its model (Claude Opus 4.6) with every analysis system, we also re-ran the full pipeline with GPT-5.4 as the judge on the same reports and rubric, as a control for judge-family bias; the two judges' rankings agree at Spearman $\rho = 0.98$ (SSB), 0.87 (HLE), and 0.93 (APEX-Agents) (Table 10, Appendix A.8).

Results. IG ranks first on all three benchmarks, with an average pairwise win rate of 80.5% against 65.4% for the next-best system (CC Subagents), and leads on every rubric dimension (composite 1.32 vs. 1.18 for Trace2Skill; Table 1). The ablations locate the gain: disabling subagent dispatch (**Orchestrator-only**) lowers the average win rate to 30.9%, while replacing the typed Scout and Investigator with an untyped subagent (**Generic Subagents**) recovers most of it (71.2%), so parallel dispatch carries most of the gain and typed roles add a smaller increment; the ordering holds under GPT-5.4 (Table 10), per-benchmark breakdowns are in Appendix A.8, and downstream component ablations in Section 4.5.

4.2 Expert Intervention Study

The Expert Intervention Study, our primary downstream evaluation, tests whether a report helps practitioners make better scaffold modifications, measured on held-out tasks. We compare IG with CC Subagents, the alternative with the highest pairwise win rate in Section 4.1 and a lightweight architecture practitioners can adopt directly. Participants, all professional software engineers who develop LLM agents, were randomly assigned to a report condition, blinded to its source, and asked to improve the scaffold with standard coding tools within a fixed time budget (1.5 hours on SSB, 2 hours on APEX-Agents); each SSB participant's score is the mean of three evaluation runs over 100 held-out tasks, and each APEX-Agents participant's score, like the unmodified baseline in Table 2, is a single evaluation run over the 144 held-out tasks. Tooling, benchmark access, and time budget are identical across arms, so the report is the only element varied by design. All participants in an arm received the same report. We recruited practitioners rather than crowd workers, prioritizing real-world relevance over sample size (protocol in Appendix A.9; statistical analysis in Appendix A.10).

Results. On SSB, practitioners given IG reports improved the scaffold by 30.4 pp over the unmodified baseline, against 16.2 pp for those given CC Subagents reports (57.4% vs. 43.2%, Table 2; two-sided Welch's t-test $p = 0.016$, Cohen's $d = 1.89$). On APEX-Agents, IG recipients again

Table 2: Expert Intervention Study. Held-out score of the scaffold after n practitioners per arm modified it given one report; the unmodified scaffold is the baseline. Parentheses: standard error of the mean across participants, in percentage points (pp). Bold: best per benchmark.

Benchmark	Condition	n	Score	Δ vs. baseline
SSB (pass rate)	Unmodified scaffold	–	27.0%	–
	CC Subagents report	6	43.2% ($\pm$4.1)	+16.2 pp
	IG report	6	**57.4%** ($\pm$1.5)	+30.4 pp
APEX-Agents (Pass@1)	Unmodified scaffold	–	19.44%	–
	CC Subagents report	8	18.40% ($\pm$0.75)	−1.04 pp
	IG report	8	**20.49%** ($\pm$0.63)	+1.05 pp

scored higher, 20.49% vs. 18.40% Pass@1 (+2.09 pp, Cohen's $d = 1.06$), with the two-sided Welch's t-test at $p = 0.053$ for $n = 8$ per arm.

4.3 Expert Rating Study

Twelve experts scored 23 insights, every finding of one IG report (11) and one CC Subagents report (12) generated on the SSB training corpus, on a four-dimension 1–5 rubric (correctness, depth, evidence quality, actionability), one insight at a time in a fixed format and blinded to source (Table 12, Appendix A.11). Aggregate ratings are a near-tie (4.25 for IG vs. 4.22 for CC Subagents): IG is ahead on depth (+0.17) and evidentiary support (+0.06), CC Subagents on correctness (+0.14), and actionability is nearly identical, with none of these differences distinguishable from zero given the per-insight standard errors in Table 12. Inter-rater reliability is low (exact agreement 39%; quadratic-weighted $\kappa = 0.115$ between the two most active raters), so the ratings cannot support a sharp claim in either direction. The closest LLM-judge analogue to these ratings is its rubric, and on SSB the rubric also separates the two systems only slightly (composite 1.20 vs. 1.12 on the 0–3 scale, Appendix A.8); the judge's large margin is in the pairwise tournament, which compares whole reports rather than individual insights. We therefore treat downstream outcomes, not insight ratings, as the arbiter of report quality: practitioners blinded to report source improved the scaffold by 30.4 pp with IG reports against 16.2 pp with CC Subagents reports (Section 4.2), the same ordering the LLM judge gives.

4.4 Automated Patcher Loop

We next test whether reports support *iterative* scaffold improvement with no human in the loop. In each round a coding agent (Claude Code with Claude Opus 4.6) modifies the scaffold given the current report and the scaffold source; the patched scaffold collects fresh traces on the training split, the analysis system regenerates its report, and the loop continues until a validation-set patience criterion is met (Appendix A.12; patcher prompt in Appendix A.13). We compare IG, RLM, and CC Subagents reports with a pure patcher that receives neither traces nor a report. Scores are the best held-out test result across rounds (pass rate on SSB; tasks solved of 144 on APEX-Agents), with the validation split used only to decide when a loop stops.

Results. On SSB, all three report-equipped systems raise the scaffold from 0.41–0.44 unmodified to held-out pass rates of 0.81–0.84 within five rounds, whereas the pure patcher reaches 0.80 after one round and then falls to 0.58 when a prompt edit contradicts the grader's exact-match contract (Appendix A.12). On APEX-Agents the arms spread apart (Tables 13 and 16): IG reports raise the scaffold from 30.8/144 to 42/144 (0.292), a +7.8 pp gain, the largest of any arm and significant against the baseline trials at the unadjusted level ($p = 0.017$, two-sided prediction-interval test); the pure patcher reaches 35/144 and the CC Subagents and RLM reports 32/144 each. Statistical caveats for these single-trial scores are in Appendix A.12.

4.5 Ablations and Cost Scaling

Component ablations. Holding the corpus, backbone, patcher, splits, and a $50 per-round analysis budget fixed, we removed one IG component at a time: **Orchestrator-only** disables subagent dispatch,

Generic Subagents keeps parallel dispatch but replaces the typed Scout and Investigator with one untyped subagent, **no Python REPL** removes the agent-writable Python environment and code chaining, so agents reach the data layer only through discrete LLM-mediated tools, and **no LLM extraction** removes the per-trace extraction tool while the trace table and column persistence remain. Every removal lowers the held-out patcher score below full IG at the same cap (34/144): 32, 31, 30, and 19 tasks respectively. Only removing LLM extraction moves the score outside single-trial baseline noise (more than four baseline standard deviations below the unmodified scaffold); the other removals are within it. Table 17 pairs these downstream scores with the judged quality of uncapped reports from the same ablation configurations under both judge families.

Cost scaling. On the SSB and HLE judge corpora, an IG analysis run averages about $44 in LLM cost and 48 minutes of wall-clock, against $8 for RLM, $26 for Trace2Skill, and $38 for CC Subagents (Appendix A.14). To test whether IG's advantage is bought with budget, we ran the APEX-Agents loop with each analysis system's per-round spend capped at $10, $25, and $50, and uncapped, where IG's billed spend on the longer APEX-Agents traces is about $174 per round. Budget buys breadth and then depth: Scouts per round grow from 1 at $10 to about 8 from $50 on, and Investigators first appear at $50 and reach 12 per round uncapped. Report quality rises with budget under both judge families across the capped tiers (Opus 4.6 composite 1.24, 1.36, and 1.61 at $10, $25, and $50; 1.87 uncapped), while the downstream score stays at 33–34/144 across the capped tiers and reaches 42/144 uncapped (Table 18). At matched caps of $10 to $50, neither IG nor CC Subagents nor RLM separates from the unmodified scaffold (all cells within 27–35/144; Table 16); IG's downstream gain appears when the cap is removed.

5 Discussion

Implications for agent development. The results carry a concrete implication for practitioners developing LLM agents: IG-style corpus-level diagnostic reports support more effective scaffold improvements than competing analysis approaches when consumed by human experts. The 14.2 pp gap between IG and CC Subagents in the SSB intervention study (Section 4.2) suggests that report depth and evidentiary grounding matter for practitioner effectiveness, not only topical coverage, a pattern consistent with IG's lead on mechanism and specificity under the LLM judge and with the Investigator role's explicit corpus-scale validation step. The automated patcher loop (Section 4.4) shows that regenerated corpus-level analysis sustains improvement: all three report-equipped patchers converge to comparable pass rates on SSB while the no-report control regresses, and on the harder APEX-Agents benchmark IG reports produce the largest held-out gain. Report quality also rises with analysis budget (Section 4.5), so practitioners can trade cost for diagnostic depth.

Limitations and extensions. Both human studies are small because participants must be experienced agent developers: $n = 6$ and $n = 8$ per arm in the intervention studies and 23 rated insights in the rating study. The SSB effect survives leave-one-out and permutation checks and the APEX-Agents effect is large ($d = 1.06$) but does not reach significance ($p = 0.053$); a fully powered replication across more benchmarks is the natural next step. Each APEX-Agents patcher cell is a single trial of a long coding-agent rollout, so differences of a few tasks lie within run-to-run noise, and repeated trials per arm would sharpen the downstream comparisons, including the budget-matched ones. All analysis systems and the primary judge run on Claude Opus 4.6 (Claude Haiku 4.5 for extraction and summarization); the GPT-5.4 judge shows that the ranking does not depend on the judge family, but we did not vary the generation backbone, so backbone generality is untested. Two comparison systems, Trace2Skill and Single-Agent Coding, were not judged on APEX-Agents; their APEX-Agents and cross-benchmark average cells in Tables 1 and 10 are marked n.j., and extending the tournament to them requires only the existing protocol. On scale, no single trace in our corpora exceeds the agent context window; content-aware chunking with per-chunk retrieval decouples per-trace length from context, but preserving cross-chunk dependencies in million-token trajectories remains open. The SSB and HLE corpora hold one trace per task, so per-prompt grouping, which isolates nondeterministic failures across repeated runs of one task, is not exercised here although IG's cohort mechanism supports it. Finally, we evaluate one downstream use of insights, scaffold improvement by practitioners and by an automated patcher. Confirmed failure cohorts and quantified failure patterns are also natural inputs to SFT data filtering, RL reward design, and evaluation-set construction, which we leave to future work.

6 CONCLUSION

We present Insights Generator, a multi-agent system for corpus-level trace diagnostics. We formalize population-level pattern discovery in agent traces as a problem distinct from single-trajectory debugging and taxonomy classification, introduce a scout-investigator architecture with a persisted, schema-cached data layer, and propose a four-setting evaluation framework varying who evaluates and what is measured, applied on three benchmarks under two judge families. IG ranks first on pairwise report quality under both judges and, under the primary judge, leads every rubric dimension. Practitioners given IG reports improve an SSB scaffold by 30.4 pp, nearly double the gain from the strongest alternative, and an automated patcher given IG reports achieves the largest held-out gain on APEX-Agents, with report quality rising as analysis budget grows. As agent complexity grows, IG provides a systematic framework for analyzing traces, diagnosing failure modes, and generating impactful improvements.

AI USE STATEMENT

In this work, we used generative AI tools for implementation (LLM-assisted coding of experiment harnesses and of the judge re-run pipeline), qualitative analysis, and result interpretation during the evaluation. We have not used generative AI tools to generate synthetic data sets, to develop the conceptual framework, to formulate hypotheses, to design the methodology, or to clean datasets, and mathematical claims and proofs and translation are not applicable to this work. Additionally, we used generative AI tools to edit the paper text for readability. Separately from these uses, LLMs are the object of study: IG, the comparison systems, the patcher, and the LLM judges are LLM-based systems described in Sections 3 and 4, and their outputs are the results we report. We have reviewed all AI-assisted work: AI-assisted code was reviewed and tested by the authors, and AI-assisted analyses and interpretations were checked by the authors against the underlying data. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The Expert Intervention Study (SSB and APEX-Agents) and the Expert Rating Study involved professional software engineers who participated voluntarily and were compensated for their time (Appendix A.9). Participants were randomly assigned, blinded to report source, and produced only scaffold code and rubric scores. We collected no personal data beyond participation records and report no participant-identifying information. The trace corpora were generated by LLM agents on public benchmark tasks and contain no user data. Benchmarks are used under their published licenses.

REPRODUCIBILITY STATEMENT

All experiments use fixed trace corpora whose generation settings are listed in Appendix A.5, and every SSB and HLE judge metric is a mean over three report replicates. System prompts for every IG role (Appendix A.2), the production configuration (Appendix A.3), the analysis primitives (Appendix A.4), benchmark and corpus statistics (Appendix A.5), comparison-system implementations (Appendix A.6), judge prompts and rubric anchors (Appendix A.7), the statistical analyses (Appendix A.10), cost and runtime (Appendix A.14), the expert rating rubric (Appendix A.9.1), the patcher protocol and prompt (Appendices A.12 and A.13), and the participant protocol (Appendix A.9) are included. We do not release code with this submission; the appendices document each component needed to reimplement the system and rerun the evaluation.

REFERENCES

Anthropic. Claude Code. https://code.claude.com/docs/en/, 2025.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. AgentRx: Diagnosing AI agent failures from execution trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026. URL https://arxiv.org/abs/2602.02475. To appear.

Amanda Bertsch, Adithya Pratapa, Teruko Mitamura, Graham Neubig, and Matthew R. Gormley. Oolong: Evaluating long context reasoning and aggregation capabilities, 2025. URL https://arxiv.org/abs/2511.02817.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, et al. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2503.13657.

Context Labs. HALO: Hierarchical agent loop optimizer, 2026. URL https://github.com/context-labs/halo. Software repository.

Hui Dai, Dan Pechi, Xinyi Yang, Garvit Banga, and Raghav Mantri. DENIAHL: In-context features influence LLM needle-in-a-haystack abilities, 2024. URL https://arxiv.org/abs/2411.19360.

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.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2404.06654.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.08435.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses, 2026. URL https://arxiv.org/abs/2603.28052.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, et al. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2512.06749.

Xuyan Ma, Xiaofei Xie, Yawen Wang, Junjie Wang, Boyu Wu, Mingyang Li, and Qing Wang. Demystifying the lifecycle of failures in platform-orchestrated agentic workflows, 2025. URL https://arxiv.org/abs/2509.23735.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. SpreadsheetBench: Towards challenging real world spreadsheet manipulation. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.14991.

Seiji Maekawa, Hayate Iso, and Nikita Bhutani. Holistic reasoning with long-context LMs: A benchmark for database operations on massive textual data. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.11996.

Rahul Nanda, Chandra Maddila, Smriti Jha, Euna Mehnaz Khan, Matteo Paltenghi, and Satish Chandra. Wink: Recovering from misbehaviors in coding agents, 2026. URL https://arxiv.org/abs/2602.17037.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills, 2026. URL https://arxiv.org/abs/2603.25158.

Long Phan, Alice Gatti, Nathaniel Li, Adam Khoja, Ryan Kim, Richard Ren, et al. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649(8099):1139–1146, 2026. doi: 10.1038/s41586-025-09962-4. URL https://www.nature.com/articles/s41586-025-09962-4.

Shortcut Research Team. SpreadsheetBench Verified: A curated evaluation set. https://shortcut.ai/blog/posts/spreadsheetbench-verified, December 2025.

Varun Ursekar, Apaar Shanker, Veronica Chatrath, Yuan Xue, and Samuel Marc Denton. VeRO: A harness for agents to optimize agents. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.22480.

Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, Neel Venugopal, Alannah Hsia, Isaac Robinson, Calix Huang, Olivia Varones, Daniyal Khan, Michael Haines, Austin Bridges, Jesse Boyle, Koby Twist, Zach Richards, Chirag Mahapatra, Brendan Foody, and Osvald Nitski. APEX-Agents. *arXiv preprint arXiv:2601.14242*, 2026. URL https://arxiv.org/abs/2601.14242.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models, 2025a. URL https://arxiv.org/abs/2512.24601.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.03312.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, et al. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025b. URL https://arxiv.org/abs/2410.10762.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, et al. Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In *International Conference on Machine Learning*, 2025c. URL https://arxiv.org/abs/2505.00212.

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 Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2306.05685.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, et al. Where LLM agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025. URL https://arxiv.org/abs/2509.25370.

A Appendix

A.1 Analysis Loop Pseudocode

Algorithm 1 Insights Generator analysis loop. Scout and Investigator batches run in parallel; only confirmed findings enter the report.

Require: Trace corpus $\mathcal{C} = \{\tau_1, \ldots, \tau_n\}$, diagnostic question Q

$\mathcal{S} \leftarrow$ BuildSchemaCache($\mathcal{C}$)
$\mathcal{H}^* \leftarrow \emptyset$ {accumulated candidate hypotheses}
$\mathcal{F} \leftarrow \emptyset$ {confirmed findings}
while Orchestrator has not issued SubmitReport **do**
  $a \leftarrow$ OrchestratorDecide($\mathcal{S}$, $\mathcal{H}^*$, $\mathcal{F}$, Q)
  if $a =$ DispatchScouts **then**
    $\mathcal{H} \leftarrow$ DispatchScoutAgents($\mathcal{C}_{\text{sample}}$, $\mathcal{S}$, Q) {parallel; propose}
    $\mathcal{H}^* \leftarrow \mathcal{H}^* \cup \mathcal{H}$
  else if $a =$ DispatchInvestigators **then**
    $\mathcal{H}_{\text{sel}} \leftarrow$ Prioritize($\mathcal{H}^*$)
    $\mathcal{F}_{\text{new}} \leftarrow$ DispatchInvestigatorAgents($\mathcal{C}$, $\mathcal{S}$, $\mathcal{H}_{\text{sel}}$) {parallel; validate}
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{f \in \mathcal{F}_{\text{new}} : f.\text{status} = \texttt{confirmed}\}$
    $\mathcal{S} \leftarrow$ Consolidate($\mathcal{S}$) {merge new extractions; refresh schema}
  end if
end while
$\mathcal{F} \leftarrow$ RemapTraceIDs($\mathcal{F}$, $\mathcal{C}$)
return SubmitReport($\mathcal{F}$, Q)

A.2 Agent System Prompts

The system prompts below correspond to the production configuration in which both Scout and Investigator subagents are dispatched and the full IG analysis toolkit is enabled. Ablation variants are rendered programmatically from the same Jinja templates with conditional blocks gated on which subagent roles are present.

Orchestrator.

```
You are the Insights Generator Orchestrator -- an analysis
coordinator that discovers patterns in AI agent execution
traces.

Your Role.  You find failures (explicit actions, silent
errors, and omissions -- things the agent should have done
but didn't), successes, behavioral differences, efficiency
issues, and any noteworthy patterns across a corpus of agent
traces.  Your primary job is dispatching specialized
subagents and synthesizing their findings -- you do NOT do
the heavy statistical analysis yourself.  You DO have light
read-only tools so you can deep-dive on a search hit when
that informs your next dispatch decision; reserve heavy
extraction, cohort comparison, and cohort-saving for the
scouts.

Workflow.  (1) Review the schema.  (2) Dispatch scout agents
with specific exploration directives.  (3) Review hypotheses
and prioritize the most promising ones.  (4) Dispatch
investigator agents with hypotheses to validate.  (5)
Evaluate coverage.  (6) Write the final report.

Tools.  A Python REPL for orientation (schema, trace counts,
search, summarization, extraction-column plumbing); plus
dispatch tools for parallel subagent dispatch.  Heavy LLM
and statistical work is intentionally not in the
orchestrator's toolkit.
```

Dispatching subagents. Directives focus on specific areas ("verbose database queries correlate with null results", not "look at tool calls"). Mix directive shapes across each batch: some target active errors (wrong action, bad output, failure text), others target expected actions or checks the agent may have OMITTED. Dispatches return compact summaries with full per-hypothesis evidence written to a sandbox sidecar that the orchestrator can read via the REPL.

Anti-overfitting. Do not generalize from a single scout agent's sample. Every finding must be validated by an investigator agent with statistical evidence. Small-cohort findings are often high-value; do not drop a hypothesis because its cohort is small.

Iteration. The pipeline is flexible: each dispatch tool may be called multiple times across the run, in any order. Default toward iterating; submit only when no credible lead remains undispatched.

Scout (Hypothesis Agent).

You are a Hypothesis Agent (Scout) -- you explore traces of agent execution to discover patterns and generate hypotheses.

Your role. Look at trace data to find interesting patterns the orchestrator should investigate further. Emphasize breadth over depth -- scan many traces to identify potential issues, not deeply validate any single one.

What to look for. Explicit failures, silent failures (agent completes but does a poor job), efficiency issues, success patterns, behavioral differences, and OMISSIONS (steps the agent should have taken but didn't). Do not limit yourself to a failure taxonomy -- discovery is the goal.

Sampling guidance. Use representative samples -- typically 50-200 traces -- to form hypotheses.

Workflow. (1) Orient via schema and a small Polars pass to understand data shape. (2) Discover via search, issuing per-concept queries (failures, successes, loops, escalations, possible omissions). (3) Read samples on interesting hits. (4) Quantify by extracting on small focused samples (10-30 traces) to test patterns. Before extracting, check what's already been extracted by other agents. (5) Submit hypotheses.

Searching for omissions. Search cannot match absence directly. Imagine the text an agent that HAD performed the expected action would produce, search for THAT, then contrast cohorts in Polars or in a follow-up search.

Output. A JSON list of hypotheses, each with name, description, evidence (trace id, verbatim quote, and explanation), and estimated prevalence. Optionally include 1-3 well-motivated suggestions for further investigation that this Scout did not have time or scope to pursue.

Investigator (Explore Agent).

You are an Explore Agent (Investigator) -- you validate hypotheses about agent execution traces with statistical evidence.

Your role. Take a specific hypothesis and validate it with rigorous evidence from the full trace corpus (or a large representative subset). Emphasize depth over breadth -- quantify patterns, run comparisons, and provide statistical backing.

Validation approach (hierarchical). Level A -- broad stats: compute statistics across all traces using Polars group-bys. Level B -- focused comparison: extract relevant features on 50-100 traces; check existing extractions first to avoid duplicating work. Level C -- deep comparison: run cohort comparisons 2-3 times with different samples for robust findings.

Output. A single submission with status (confirmed, refuted, or inconclusive), 1-2 sentence summary, quantified prevalence, and 8-10+ evidence items. Each evidence item contains a trace id, a verbatim quote from the trace, and a 2-3 sentence explanation that names the specific mechanism visible in THIS trace and connects it to the broader pattern. Hypothesis-test outputs (p-values, Cramér's V, odds ratios) go in additional observations, not the main summary. Provide a single concrete suggested action or null.

Anti-overfitting. Do not generalize from fewer than 20 traces in a comparison. Quantify everything. Flag confidence level: high (more than 100 traces), medium (20-100), low (fewer than 20).

Saving the affected cohort. Record the defensible cohort -- the subset you are confident exhibits the pattern, NOT the broader pool you analyzed. Submission is rejected if this has not been called.

A.3 Implementation Details

Table 3 lists the production IG configuration used in all reported experiments. All fields correspond directly to attributes of the IG configuration dataclass; ablation experiments override the relevant fields and leave the rest at their defaults.

Table 3: IG production configuration. Concurrency, timeout, and sandbox parameters are those used for the runs reported here and may be lowered for cost-constrained deployments.

Group	Field	Value
Models	Orchestrator	Claude Opus 4.6
	Hypothesis (Scout)	Claude Opus 4.6
	Explore (Investigator)	Claude Opus 4.6
	Cohort comparison	Claude Opus 4.6
	Per-trace extraction	Claude Haiku 4.5
	Trace summarization	Claude Haiku 4.5
Extraction	Concurrency	50 parallel calls
	Checkpoint interval	5,000 traces
	Auto-save threshold	250 traces
	Per-call timeout	300 s
Orchestration	Max orchestrator turns	500 (self-terminates via report)
	Max subagent turns	500 (self-terminate via submit)
Traces	Long-trace chunk threshold	50,000 tokens
	Retrieval top-k	20 (search default)
Ablation	Mode	full
	Tool backend	REPL
	REPL state	stateful
Sandbox	CPU	16 cores
	Memory	65,536 MB (64 GB)
	Timeout	14,400 s (4 h)

All LLM calls route through a centralized LLM proxy. Subagents run in a shared sandboxed container with the analysis primitives pre-injected into a stateful Python REPL session; the orchestrator and each subagent get their own session.

A.4 INSIGHTS GENERATOR TOOLS

Table 4 lists the 14 IG analysis primitives organized by category, with per-role availability across the Orchestrator, Scout (Hypothesis Agent), and Investigator (Explore Agent).

Table 4: IG analysis primitives, pre-injected as Python functions in every agent's REPL session and called from agent-written code. ✓/×: available or not to the Orchestrator (Orch.), Scout, and Investigator (Inv.).

Function	Description	Orch.	Scout	Inv.
Trace inspection				
`load_traces`	Load full corpus + extractions as a Polars DataFrame	✓	✓	✓
`get_trace`	Return full content of one trace; chunk-list outline if oversized	✓	✓	✓
`get_trace_chunk`	Retrieve a specific chunk of a long trace	✓	✓	✓
`get_schema`	Per-column metadata: types, null rates, distributions, correlations	✓	✓	✓
`search_traces`	Hybrid semantic-keyword retrieval over trace content	✓	✓	✓
`get_extractions`	List previously persisted extraction columns and their field definitions	✓	✓	✓
LLM-driven analysis				
`extract`	Structured feature extraction across traces with chunking, batching, and checkpointing	×	✓	✓
`summarize_trace`	Per-trace LLM summary with caching and progressive summarization for long traces	✓	✓	✓
`summarize_group`	Single LLM summary characterizing a group of traces	✓	✓	✓
`compare_segments`	Deep A/B comparison between two cohorts with token budgeting	×	×	✓
Data management				
`save_column`	Persist a computed Polars column as an extraction parquet	✓	✓	✓
`save_affected_traces`	Persist a trace-id list as the cohort supporting a finding	×	×	✓
`reload_data`	Refresh cached DataFrame after new extractions land	✓	✓	✓
`consolidate`	Merge extraction parquets into the base store and refresh the schema cache	✓	✓	✓

A.5 BENCHMARK AND CORPUS STATISTICS

The experiments in this paper use three benchmarks: **SpreadsheetBench (SSB)** (Ma et al., 2024), **HLE** (Phan et al., 2026), and **APEX-Agents** (Vidgen et al., 2026). For each benchmark, traces were collected by executing a fixed ReAct-style scaffold over the benchmark's task set; all downstream analysis, ablation, and intervention experiments reuse those fixed corpora rather than re-collecting traces per system. The SSB human evaluation experiments (Sections 4.2 and 4.3) use traces generated by `openai/gpt-5.2-2025-12-11`, while the LLM judge (Section 4.1) and patcher-loop (Section 4.4) experiments, and all APEX-Agents experiments, use traces generated by `openai/gpt-5.4-2026-03-05`. The sole exception is the automated patcher loop, which, by construction, regenerates a fresh trace corpus each round from the patched scaffold $\mathcal{A}_r$. Corpus-level statistics for all three corpora are summarized in Table 5.

SpreadsheetBench (SSB). SpreadsheetBench tasks require an agent to read an Excel workbook, perform a manipulation specified in natural language (e.g. filter, summarize, transform, write a formula), and persist the result back to the workbook. Outputs are graded by exact cell-value match against ground truth, which makes the benchmark deterministically scorable. SSB is the primary benchmark for the human studies because its tasks are familiar to technology professionals, run

quickly, and admit prompt, tool, and control-flow fixes. The scaffold is a ReAct agent with a 5-turn budget that exposes `openpyxl` and `pandas` via a Python code-execution tool; the agent has no web access. The LLM judge corpus (`ssb_gpt-5.4-med_20260430`, used in Section 4.1) contains 296 traces with a baseline pass rate of 42.2% (125 correct, 171 incorrect). Tasks are categorized by the benchmark into *Cell-Level* and *Sheet-Level* manipulation, with markedly different difficulty profiles (approximately 27% vs. 76% pass rate in our corpus). The human-expert experiments (Sections 4.3 and 4.2) use the SSB training-split corpus of 250 traces (110 correct, 140 incorrect; baseline pass rate 44%); the training-split bundle is described in detail in Appendix A.9.1. The automated patcher loop (Section 4.4) collects fresh training-split traces on every round from the patched scaffold; the round-0 baseline pass rate is 0.41 on the held-out test split.

HLE. HLE (Humanity's Last Exam) is a closed-book expert-level question-answering benchmark spanning physics, mathematics, biology, chemistry, and humanities, where each task requires the agent to return a single short answer along with a self-reported confidence in $[0, 100]$. The scaffold is a ReAct agent that exposes web search, web-page fetch, and Python code execution; turn counts in our corpus range from 0 (instant-answer / hang) to 18 with a long-tail distribution. The LLM judge corpus (`hle_gpt-5.4_20260430`) contains 250 traces with a baseline pass rate of 23.6% (59 correct, 191 incorrect); 207 of 250 traces use at least one tool. The 43 zero-tool traces achieve only 9.3% accuracy versus 26.6% for tool-using traces, a sub-population gap that surfaces consistently across systems' findings. Correctness for HLE is graded by an LLM judge applied to the benchmark's reference answers (rather than by deterministic match), so HLE serves as a complementary stressor to SSB's deterministic grader.

APEX-Agents. APEX-Agents contains long-horizon, document-grounded professional tasks in law, investment banking, and management consulting. Relative to SSB its traces contain roughly 13× more tool calls, its tasks expose over 100 more files and over 10 more tools each, and each trial produces about 4× more output tokens. Failures we observed include long-horizon planning errors, tool-output context exhaustion, and silent data-processing errors, alongside isolated API misuse. The Expert Intervention Study used a 236-trace corpus, a 20-task development split for participant iteration, and a 144-task held-out test split; the automated patcher loop and budget grid report strict Pass@1 on the same 144 held-out tasks against an unmodified-scaffold baseline pooled over five trials (154/720, 0.214). The scaffold is a ReAct-style agent (GPT-5.4, medium reasoning effort) with filesystem search and read, spreadsheet, document, slide, and PDF reader tools plus code execution and no web access, a 100-turn budget, and a one-hour wall-clock limit; 215 of the 236 traces call at least one tool, and 20 of the 21 zero-tool traces are law tasks.

Table 5: Trace corpora for SSB, HLE, and APEX-Agents. Eval-corpus traces come from a generic ReAct scaffold on `openai/gpt-5.4-2026-03-05`; the SSB human-study split uses GPT-5.2 traces (Appendix A.9). Eval corpus: the corpus released to every analysis system for the LLM-judge evaluation (Section 4.1); human-study and patcher splits are listed separately. Parenthesized trace counts are correct / incorrect.

	SSB	**HLE**	**APEX-Agents**
Citation	Ma et al. (2024)	Phan et al. (2026)	Vidgen et al. (2026)
Domain	Spreadsheet manipulation	Closed-book expert QA	Long-horizon professional tasks
Eval metric	Exact cell-value match	LLM-judged correctness	Pass@1 (rubric grader)
Eval-corpus traces	296 (125 correct / 171 incorrect)	250 (59 correct / 191 incorrect)	236 (47 correct / 189 incorrect)
Eval-corpus pass rate	42.2%	23.6%	19.9%
Tool-using traces	all (code-exec mandatory)	207 / 250 (82.8%)	215 / 236 (91.1%)
Tool surface	`openpyxl`, `pandas`	web search, web fetch, code-exec	filesystem, spreadsheet, docs, slides, PDF, code-exec
Turn budget	5	20 (observed max 18)	100 (1 h wall-clock)
Human-study split	250 train (110 / 140), 44% pass	—	236 train, 20 dev, 144 test
Replicates per system (Section 4.1)	3 independent re-runs	3 independent re-runs	1
Union-gold clusters	50	50	37 (Opus 4.6), 34 (GPT-5.4)

Replicates, gold clusters, and corpus size. For the LLM judge evaluation each system is run 3 times on SSB and HLE (once on APEX-Agents) on the same fixed corpus to capture run-to-run variance, and the corresponding union-gold reference set per benchmark is built by pooling

findings across all systems and replicates and clustering them with the meta-evaluator prompt in Appendix A.7; clustering produced 50 canonical observations on SSB and HLE and 37 on APEX-Agents (the clustering imposes no cluster cap), which is the cluster count against which coverage and per-dimension quality are scored in Tables 6 and 1. No single trace exceeds the agent context window, but the full 296-trace SSB corpus and 250-trace HLE corpus each exceed any single LLM context, motivating the corpus-level analysis primitives described in Section 3.

A.6 Comparison System Implementation Details

All five systems answer the same canonical diagnostic question (Appendix A.7) over the same trace corpus, and all emit a session artifact normalized to the same JSON schema by a shared report-utils module. The systems differ along three axes: *tool surface* (raw filesystem and coding tools vs. IG's structured analysis primitives), *topology* (single agent vs. multi-agent dispatch), and *data layer* (raw text vs. structured Polars store with extraction columns). All systems use Claude Opus 4.6 as their backbone agent model.

Single-Agent Coding. A single Claude Code CLI session, invoked headless, with a filesystem-based knowledge layout (one file per trace plus an index) and Claude Code's native tool surface (read, write, edit, glob, grep, bash). The agent discovers patterns through filesystem retrieval, ad-hoc scripting, and manual aggregation. Establishes the floor for what a strong coding agent can achieve without a dedicated trace-analysis scaffold.

CC Subagents. Claude Code with subagent dispatch via the `Task` tool, implementing the same scout-investigator role decomposition as IG. Each subagent receives a focused directive from the orchestrator and returns a compact summary; subagents do not share a structured data layer and each runs in an isolated context, so all cross-agent integration happens via the orchestrator's reasoning.

Trace2Skill-style. A pipeline modeled after Trace2Skill (Ni et al., 2026): a flat parallel fleet of analyst subagents over individual trajectories, followed by a hierarchical consolidation pass that merges per-batch findings into a conflict-free output.

RLM. The vanilla Recursive Language Model implementation from Zhang et al. (2025a), wrapped to load a parquet of agent traces and produce a structured-report JSON. The root LM dispatches recursive sub-LM calls at maximum depth 2; sub-LMs at maximum depth fall back to plain LM calls. Distinct from HALO (Context Labs, 2026), which extends an RLM with a closed-loop coding-agent stage (Section 2).

IG (this work). The full production IG configuration described in Section 3 and Appendix A.3, with the orchestrator, scout, and investigator agents operating over a shared structured trace store with schema-aware extraction primitives (Algorithm 1).

A.7 LLM Judge: Evaluation Prompts and Rubrics

The LLM judge evaluation in Section 4.1 is layered into three stages, each using a separate judge prompt. All three stages use the same judge model (Claude Opus 4.6). The shared canonical diagnostic question (the input given to every system) is also reproduced below.

Canonical diagnostic question. Every comparison system received the following question over the same trace corpus:

> *This corpus contains execution traces from an AI agent attempting the {benchmark name} benchmark. Surface the notable behavioural patterns in the corpus: recurring errors, silent failures, systematic efficiency issues, unexpected strengths, and how sub-populations of traces differ (e.g. correct vs. incorrect, high- vs. low-confidence, tool-using vs. not). For every finding: quantify prevalence over the total number of traces in the corpus; cite at least 3–5 specific trace IDs and direct quotes as evidence; compare relevant sub-populations; and distinguish confirmed / refuted / inconclusive. Do not focus on headline benchmark metrics (accuracy, pass-rate); focus on agent behaviours. Return the structured JSON report as specified.*

Stage 1: Union-gold clustering. A single judge call ingests every finding from every system and clusters them by underlying observation. The output is a canonical reference set against which each system is later scored. Prompt:

```
You are a meta-evaluator clustering findings from multiple
automated trace-analysis systems that all analyzed the SAME
corpus of AI-agent execution traces.  Each input finding is
a claim about the corpus.  Your job is to produce a 'union
gold' set:  the canonical list of distinct observations
about the corpus that appear across ANY of the systems.
Rules:  (1) Two findings belong in the same cluster iff they
describe the same underlying phenomenon in the corpus (same
behavior, failure mode, or pattern), even if their framing,
prevalence numbers, or prescriptions differ.  Use the
affected trace-id lists as strong evidence for sameness.
(2) A cluster may contain a single finding (unique
observations are valuable).  (3) Give each cluster a crisp
canonical name (at most 80 characters) and a 1-3 sentence
description.  (4) Include up to 5 representative trace ids
that most strongly evidence the cluster (prefer ids cited by
multiple systems).  (5) Output VALID JSON. No prose before
or after the JSON block.
```

Stage 2: Per-cluster rubric scoring. For each system, a separate judge call scores the system's report against each union-gold cluster on five dimensions. Each dimension is scored on a 0–3 scale with explicit anchor descriptions:

detection	0 absent; 1 passing mention; 2 explicit finding; 3 explicit finding given prominence
mechanism	0 none; 1 vague; 2 plausible; 3 sharp and testable
evidence	0 none; 1 single; 2 multiple; 3 quantified across corpus
specificity	0 generic; 1 mildly specific; 2 specific; 3 precise with quantified scope
actionability	0 none; 1 vague; 2 concrete; 3 concrete and testable

The judging-discipline instructions explicitly direct the judge to: score 0 when a dimension is not evidenced in the report (do not give credit for what a well-intentioned reader might infer); cap detection at 1 if the report makes a factually wrong claim about the observation; and use the report's own findings verbatim rather than rewarding paraphrase. The main-body quality table (Table 1) reports mean scores on the three most discriminative dimensions (*mechanism*, *specificity*, *actionability*) plus a composite; *detection* is consumed by the coverage metric (Appendix A.8); *evidence* is omitted from the tables because it is highly correlated with mechanism in our runs.

Stage 3: Pairwise ranking. For each unordered pair of systems, the judge is presented with both reports twice in opposite positions to control for position bias (Zheng et al., 2023). Each round is scored independently. The pairwise judge prompt:

```
You are an expert ML research engineer comparing two
automated trace-analysis reports that were produced by
different systems analyzing the SAME corpus of AI-agent
execution traces and answering the SAME canonical question.
Your job:  pick the report a scaffold-improvement engineer
would find more useful for actually fixing the agent.  Weigh
specificity, evidentiary grounding (cited trace ids and
quantitative prevalence), novelty (surfacing non-obvious
patterns), and actionability (concrete scaffold changes).
Penalize generic platitudes, unsupported claims, and reports
that confuse breadth for depth.
Answer in strict JSON, with fields:  winner (A, B, or tie);
confidence (low, med, or high); rationale (1-3 sentences).
```

The win rate reported in Table 1 is computed per system as $(\text{wins} + 0.5 \cdot \text{ties})/n_{\text{rounds}}$ across all rounds in which the system participated.

A.8 LLM JUDGE: PER-BENCHMARK BREAKDOWNS, ABLATIONS, AND CROSS-FAMILY JUDGE

This appendix provides per-benchmark breakdowns of the LLM judge results summarized in Table 1. All SSB and HLE metrics are means across three independent report replicates per system; APEX-Agents rows use one report per system. The four comparison systems are described in Section 4.1, along with the IG architecture ablations.

Coverage. Table 6 reports the coverage metric omitted from Table 1: the percentage of union-gold clusters (50 on SSB and HLE, 37 on APEX-Agents) for which a report cited at least one member trace.

Table 6: Coverage (%): share of union-gold clusters for which a report cites at least one member trace (Claude Opus 4.6 judge). SSB and HLE: 50 clusters, mean of three replicates; APEX-Agents: 37 clusters, one report per system. Bold: best per column; n.j.: not judged on APEX-Agents.

System	SSB	HLE	Avg (SSB, HLE)	APEX
IG	87	95	91	50.8
CC Subagents	93	97	95	50.6
Trace2Skill	**96**	**98**	**97**	n.j.
RLM	89	96	92.5	41.1
Single-Agent Coding	65	95	80	n.j.
IG, Generic Subagents	87	93	90	**52.1**
IG, Orchestrator-only	78	86	82	32.4

Pairwise win-rate rankings. Table 7 reports per-benchmark pairwise win rates sorted by rank. IG leads on all three benchmarks; the gap is larger on HLE (83.3%) than SSB (72.4%). Each pair is judged in two position-swapped rounds and each round is tallied independently: a pair whose two verdicts disagree contributes one win and one loss to each system, and a verdict the parser cannot read is recorded as a tie.

Table 7: Pairwise win rate (WR, %) per benchmark with systems sorted by rank (Claude Opus 4.6 judge); WR $= (\text{wins} + 0.5\times\text{ties}) \,/\, n_{\text{rounds}}$ over position-swapped round-robin games. SSB and HLE: three replicates per system; APEX-Agents: one report per system, with Trace2Skill and Single-Agent Coding not judged.

SSB			HLE			APEX-Agents		
Rank	System	WR%	Rank	System	WR%	Rank	System	WR%
1	IG	72.4	1	IG	83.3	1	IG	85.7
2	CC Subagents	63.3	2	CC Subagents	61.4	2	CC Subagents	71.4
3	Trace2Skill	56.2	3	Trace2Skill	49.0	3	RLM	21.4
4	RLM	13.8	4	Single-Agent Coding	12.4			
5	Single-Agent Coding	8.6	5	RLM	9.0			

Per-benchmark rubric scores. Table 8 breaks out the per-dimension quality scores by benchmark. The cross-benchmark averages reported in Table 1 are the means of these rows.

Table 8: Per-dimension rubric scores (0–3) against union-gold clusters, by benchmark (Claude Opus 4.6 judge): mechanism (Mech.), specificity (Spec.), actionability (Act.), and their mean (Comp.). SSB and HLE: mean of three replicates; APEX-Agents: one report per system; n.j.: not judged on APEX-Agents.

System	Bench.	Mech.	Spec.	Act.	Comp.
IG	SSB	1.23	1.23	1.13	1.20
	HLE	1.45	1.44	1.41	1.43
	APEX	1.59	1.51	1.41	1.50
CC Subagents	SSB	1.15	1.15	1.05	1.12
	HLE	1.10	1.07	1.09	1.09
	APEX	1.49	1.51	1.41	1.47
Trace2Skill	SSB	1.06	1.03	1.08	1.06
	HLE	1.30	1.25	1.38	1.31
	APEX	n.j.			
RLM	SSB	0.95	0.95	0.81	0.90
	HLE	0.96	0.94	0.87	0.92
	APEX	1.22	1.16	1.00	1.13
Single-Agent Coding	SSB	0.78	0.75	0.65	0.73
	HLE	0.78	0.78	0.80	0.79
	APEX	n.j.			

IG's quality advantage is larger on HLE (composite 1.43 vs. next-best Trace2Skill at 1.31) than on SSB (1.20 vs. CC Subagents at 1.12), consistent with the wider pairwise gap on HLE. Trace2Skill's actionability score is notably strong on HLE (1.38), approaching IG (1.41); on SSB and HLE this is the only dimension where another system comes within 0.03 of IG.

IG architecture ablations: per-benchmark detail. Table 9 reports the per-benchmark breakdown of the two IG architecture ablations summarized in Table 1. Both ablations preserve IG's full analysis toolkit (Appendix A.4) and stateful Python data layer; they differ only in subagent topology. The **Orchestrator-only** variant disables subagent dispatch entirely, forcing the orchestrator to run all extraction, summarization, and cohort comparison itself within its own context window. The **Generic Subagents** variant retains parallel subagent dispatch but replaces the typed Scout (breadth-first hypothesis proposal) and Investigator (depth-first corpus-scale validation) roles with a single generic analyst-subagent role that receives an unstructured analysis directive.

Table 9: IG architecture ablations by benchmark (Claude Opus 4.6 judge). IG (main) is the full system of Table 1. Columns: coverage (%), pairwise win rate (WR, %), and 0–3 rubric scores for mechanism (Mech.), specificity (Spec.), actionability (Act.), and their mean (Comp.). APEX-Agents: one report per system, coverage over 37 clusters.

System	Bench.	Coverage (%)	Pairwise WR (%)	Mech.	Spec.	Act.	Comp.
IG (main)	SSB	87	72.4	1.23	1.23	1.13	1.20
	HLE	95	83.3	1.45	1.44	1.41	1.43
	APEX	50.8	85.7	1.59	1.51	1.41	1.50
IG, Generic Subagents	SSB	87	59.0	1.02	1.02	0.89	0.98
	HLE	93	76.0	1.26	1.23	1.17	1.22
	APEX	52.1	78.6	1.57	1.54	1.30	1.47
IG, Orchestrator-only	SSB	78	35.0	0.92	0.90	0.79	0.87
	HLE	86	29.0	0.81	0.83	0.85	0.83
	APEX	32.4	28.6	0.92	0.92	0.92	0.92

The Orchestrator-only variant degrades sharply on every benchmark (composite $1.32 \rightarrow 0.85$ on SSB and HLE, pairwise WR $80.5\% \rightarrow 30.9\%$), and its coverage also drops below the multi-agent floor reached by every other multi-agent system in Table 6 (82% vs. $\geq 90\%$). Without parallel subagents,

the orchestrator cannot afford to perform repeated corpus-scale extractions inside its own context, so findings consolidate around a smaller subset of the union-gold clusters.

The Generic Subagents variant is intermediate: coverage and HLE pairwise WR remain high (90% Avg coverage, 76% HLE WR), but per-dimension quality drops by roughly 0.20 in composite. The pattern is consistent across mechanism, specificity, and actionability rather than localized to any one dimension, suggesting that the typed Scout/Investigator decomposition affects *how grounded* a finding is (depth of mechanism, prevalence quantification, citation specificity) rather than *whether* the finding is surfaced at all. Together, the two ablations order the configurations as Orchestrator-only $<$ Generic Subagents $<$ full IG, with pairwise WR differences of approximately 40.3 pp (Orchestrator-only $\rightarrow$ Generic Subagents) and 9.3 pp (Generic Subagents $\rightarrow$ full IG); because win rates depend on the tournament pool, these differences are not an additive decomposition of IG's 80.5%.

Cross-family judge. To test whether the ranking depends on the judge sharing a model family with the systems under evaluation, we re-ran all three judge stages (union-gold construction, per-cluster rubric scoring, and the pairwise tournament) with GPT-5.4 as the judge on the same reports and the same rubric and prompts. The 12 judged configurations are the seven in Table 1 plus three IG tool ablations (no LLM extraction, no Python REPL, no vector search) and two further Claude Code variants (CC Subagents with an enhanced prompt, and Claude Code with skills); these five additional configurations enter only the rank correlation and are not tabulated. Table 10 reports per-system win rates under the GPT-5.4 judge; on APEX-Agents, eight configurations were judged with one report each.

Table 10: Cross-family judge: GPT-5.4 re-judges the reports of Table 1 with the same rubric and prompts. **WR**: pairwise win rate (%); Avg: mean over the three benchmarks. **Comp.**: rubric composite (0–3) averaged over SSB and HLE, or on APEX-Agents alone. APEX-Agents: one report per system; n.j.: not judged on APEX-Agents. Bold: best per column. Last row: Spearman rank correlation between the GPT-5.4 and Opus 4.6 rankings of all judged configurations (12 on SSB and HLE, 8 on APEX-Agents).

	Pairwise WR (%)				Comp. (0–3)	
System	SSB	HLE	APEX	Avg	SSB+HLE	APEX
IG	**77.6**	**81.4**	**92.9**	**84.0**	1.17	**1.49**
CC Subagents	61.0	69.0	78.6	69.5	1.07	1.28
Trace2Skill	47.1	49.0	n.j.	n.j.	**1.25**	n.j.
RLM	19.0	10.5	28.6	19.4	0.98	1.15
Single-Agent Coding	19.0	13.3	n.j.	n.j.	0.87	n.j.
IG architecture ablations						
IG, Generic Subagents	55.2	78.1	78.6	70.6	1.16	1.38
IG, Orchestrator-only	38.1	29.5	42.9	36.8	1.00	0.95
Rank agreement with Opus 4.6 (ρ)	0.98	0.87	0.93	–		

Faithfulness audits. Citation validity is checked by resolving every cited trace ID against the corpus and every quoted snippet against the cited trace: 951/951 cited traces resolve on SSB and HLE, while on APEX-Agents citation validity falls to 81.6% at the lowest analysis-budget cap and reaches 99.8% uncapped (Appendix A.12); verbatim quote match is 62.0% on SSB and 41.2% on HLE, and the remaining quotes are not verbatim matches, which we treat as paraphrases of the cited trace but cannot confirm by this check. Prevalence claims are recomputed from the corpus's recorded execution labels (pass/fail, task type, tool-call and runtime counts) for every checkable figure in each report; every one matched exactly or, for continuous rates, to three decimal places, and for the 84% finding in Section 1 the denominator (incorrect Python-using traces) reproduces exactly from the labels, while the numerator, traces exhibiting a silent computation failure, is a semantic judgment this check cannot verify. Whether a cited trace exhibits the described mechanism is not checked deterministically by this audit; an LLM entailment pass over cited traces is run but not treated as ground truth. On reference independence, the pairwise tournament uses no gold, and IG contributed findings to only 12 of the 50 SSB union-gold clusters, so 76% of the reference was derived from other systems' findings; this bounds IG-specific circularity but not circularity in general, since the

reference is built from the evaluated systems' outputs, and coverage credits any citation of a cluster's member traces.

A.9 Human Participant Study Protocol

We conducted three human-expert studies. Experiment 1 is the Expert Rating Study; Experiments 2 and 3 form the Expert Intervention Study on SSB and APEX-Agents. Experiment 1 (Section 4.3) elicits expert ratings of individual insights drawn from competing reports against a structured rubric, isolating report quality from downstream implementation. Experiment 2 (Section 4.2) measures whether IG reports help practitioners make better scaffold modifications under a held-out benchmark protocol. Experiments 1 and 2 use SSB (Ma et al., 2024) as the underlying corpus and CC Subagents as the comparison report source, and both blind participants to the system that produced each report they evaluated.

Recruitment and eligibility. Participants were professional software engineers with experience in LLM agent development, recruited through professional networks. Each participant had to be capable of reading agent traces, diagnosing failure modes, and reasoning about scaffold code in a real development environment.

A.9.1 Experiment 1: Expert Rating Study

Materials. Each participant received (i) the SSB training-split trace corpus (250 agent traces: 110 correct, 140 incorrect; baseline pass rate 44%); (ii) a benchmark description covering the agent's code-execution loop, the available tooling (`openpyxl` and `pandas`), the 5-turn budget, and the cell-by-cell evaluation procedure; and (iii) a sequence of insights drawn from the two reports under comparison. Insights were presented one at a time and the source system was not disclosed. The trace corpus was provided as a single bundled file ($\sim$6,000+ lines); participants were instructed to use a coding assistant (Claude Code or Cursor) to navigate and spot-check claims rather than read traces sequentially.

Insight presentation. Each insight followed a fixed structured format identical across both report systems: a status label (`confirmed`, `refuted`, or `inconclusive`), a corpus-level prevalence estimate, a narrative description of the pattern, an evidence block with cited trace IDs and verbatim quotes each accompanied by a per-trace explanation, the full list of affected trace IDs, a suggested scaffold action, and any additional observations. The shared format prevents presentation-style confounds between systems.

Rubric. Each insight was scored on a 1–5 scale across four dimensions: *correctness* (whether the claim is actually true against the traces), *depth* (whether the insight explains the underlying mechanism rather than describing the symptom), *evidence quality* (whether the claim is backed by specific, verifiable trace citations), and *actionability* (whether the suggested improvement is concrete, proportionate to the evidence, and free of overfitting risk). Anchor descriptions were provided for each level on each dimension; per-dimension score breakdowns are reported in Appendix A.11. Rubric instructions explicitly flagged common confounds: *correctness* and *evidence quality* are independent (a finding may cite many traces but draw the wrong conclusion, or be true and verifiable without citing specifics); a finding that claims population-level prevalence while citing only 1–2 traces was capped at evidence-quality ≤ 3; and a finding that recommended a sweeping change based on a narrow pattern received a lower actionability score regardless of its correctness on the cited traces.

Time budget and grading discipline. Approximately 120 minutes total per participant, with a 5–10 minute target per insight. Participants recorded a start time so that per-insight grading time could be measured post-hoc. Participants were instructed to score each insight on its own merits, not to adjust scores based on how other insights had scored, and not to deflate scores when two insights described overlapping phenomena.

Cohort and sample. Twelve human-expert evaluators scored all 23 insights (11 IG, 12 CC Subagents) of one IG report and one CC Subagents report generated on the SSB training split. Each

(insight, evaluator) pair produced one set of four dimension scores; aggregate results in Section 4.3 are means over evaluators within each report condition.

A.9.2 Experiment 2: Expert Intervention Study on SSB

Random assignment and blinding. Eligible participants were randomly assigned to one of the two report conditions (IG or CC Subagents). Assignment labels were not disclosed: participants were told that they would be evaluating one of several candidate analysis-report formats but were not informed which system had generated the report they received, and the report itself contained no system-identifying metadata. Graders of the resulting scaffold modifications had access only to the held-out benchmark task set and the participant's modified scaffold; they were not told which report condition the participant had been in.

Baseline-competency screen. Prior to the main intervention task, each participant completed a short qualification task designed to confirm baseline competence with the trace data and tooling: given a sample of execution traces, the participant was asked to use a coding assistant (Claude Code or Cursor) to compute the distribution of tool calls per trace. Participants who completed the screen task were enrolled in the main study; the screen was scored pass/fail on whether the reported counts matched ground truth on a small held-out sample. The screen was identical across conditions and was completed before condition assignment was revealed, so it cannot bias the between-arm contrast.

Conditions. Two conditions: participants received either an IG report or a CC Subagents report (six participants per arm). Condition assignment was blinded so participants did not know which report-generation system produced the analysis they received. Headline results in Section 4.2 compare both arms against the unmodified baseline scaffold.

Task briefing. Each participant received a self-contained package ($\sim$400 MB) containing: (i) the SSB scaffold source; (ii) 250 training traces and 50 baseline-eval traces; (iii) the assigned condition's insight report; (iv) the reference baseline pass rate (44% on 22/50 dev tasks); (v) a Docker runtime image with Python, LibreOffice, the benchmark eval data, and the harness; and (vi) a Claude Code warm-up task to verify environment setup. Participants were instructed to use Claude Code or Cursor as their coding assistant and were provided a $50 API budget for agent calls (Claude Opus 4.6).

Train/eval split. The 250 training traces were generated with GPT-5.2 and serve as the failure-mode corpus that each insight report describes; the 50-task evaluation set is run with Claude Opus 4.6, which is also the model the participant's modified scaffold is scored against. Train and eval task sets are disjoint, and because training traces come from GPT-5.2 while evaluation runs Claude Opus 4.6, baseline pass rates are not comparable between the two. Participants were instructed to treat training traces as illustrative of scaffold failure modes rather than as a performance ceiling.

Time and iteration. Maximum 1.5 hours per participant. Iterative scaffold edits via the participant's IDE were picked up automatically by the running container, allowing rapid edit/eval cycles.

Locked configuration. Four scaffold parameters were locked and the eval runner enforced the lock: the model (Claude Opus 4.6), the maximum number of agent turns (5), the number of preview rows shown to the agent (5), and the per-cell execution timeout (60 s). Participants modified the prompts, the agent loop, and non-locked configuration fields only. Modifying the eval runner, the task-id list, or the data directory invalidated a submission.

Evaluation. Each participant's modified scaffold was scored on the 100-task held-out test split (disjoint from the 250 training tasks and the 50-task dev set used during iteration). Scoring was performed by re-running the eval harness three times per submission and averaging; reported scores in Section 4.2 are these per-participant means. Evaluation is deterministic per run (cell-by-cell spreadsheet comparison), so run-to-run variance reflects only the agent's non-determinism at Claude Opus 4.6.

Submission and audit trail. Participants exported their coding-assistant chat transcripts as part of the submission, ordered chronologically. The transcript audit trail was used post-hoc to verify that

participants did not bypass the locked configuration or access eval-side artifacts. No participant in either report condition was excluded for protocol violations.

Compensation and labor. To maintain study quality, we provided participants in Experiments 1 and 2 with a comprehensive briefing package governing study procedures, evaluation rubrics, and (for Experiment 2) the scaffold-improvement task and locked-configuration constraints. Due to the inclusion of personally identifying recruitment correspondence and proprietary internal protocols, we summarize the core requirements above and do not release the unredacted briefing materials. On average, each Experiment 1 grading session required approximately 2 hours of expert labor to review traces, score insights, and submit responses; each Experiment 2 intervention session required up to 1.5 hours of expert labor to analyze the assigned report, modify the scaffold, and submit results. Participants in Experiments 1, 2, and 3 were compensated under contractual terms that met or exceeded applicable local wage requirements. No unpaid volunteer or crowd labor was used.

A.9.3 Experiment 3: Expert Intervention Study on APEX-Agents

Experiment 3 repeats the Experiment 2 design on APEX-Agents: random assignment to the IG or CC Subagents report condition, blinding to report source, identical tooling and benchmark access, and a fixed two-hour budget. Participants iterated against a 20-task development split whose evaluation takes up to ten minutes per run, and were scored on the 144-task held-out split.

A.10 Statistical Analysis of the Expert Intervention Study

This appendix provides robustness analyses supporting the Welch two-sample t-test reported in Section 4.2. The SSB analyses below operate on the per-participant data ($n = 6$ per arm; each participant's score is the mean of three independent evaluation runs).

Welch two-sample t-test (primary). The CC Subagents arm has a per-participant mean of 43.2% (SD 9.95, SEM 4.06); the IG arm has 57.4% (SD 3.69, SEM 1.51). The between-arm difference is 14.17 pp with standard error 4.33, yielding Welch's $t = 3.27$, Welch–Satterthwaite $\mathrm{df} = 6.35$, and two-sided $p = 0.016$. The 95% confidence interval on the between-arm difference is $[3.56, 24.78]$ pp; Cohen's $d = 1.89$.

Exact permutation test. As a non-parametric robustness check that does not assume normality of the per-participant means, we enumerate all $\binom{12}{6} = 924$ ways of relabeling the 12 participants between arms. The observed difference (14.17 pp) is matched or exceeded in 12 of 924 relabelings, giving an exact two-sided $p = 0.013$, in agreement with the parametric Welch result.

Leave-one-participant-out sensitivity. We refit the Welch test 12 times, each time omitting one of the 12 participants. The between-arm difference ranges from 12.4 to 17.9 pp; the test statistic ranges $t \in [2.60, 6.96]$ and the two-sided p-value ranges $p \in [0.0001, 0.048]$, with $p < 0.05$ in every refit. The worst-case refit leaves $p \approx 0.048$; the best-case refit leaves $p < 0.001$. No single participant is responsible for the headline result.

Mixed-effects model. Because each participant contributes three evaluation runs, we additionally fit a random-intercepts linear mixed model to the $2 \times 6 \times 3 = 36$ run-level scores using REML, with arm as a fixed effect and participant as a random intercept:

$$y_{ijk} = \beta_0 + \beta_1 \cdot \mathbb{1}[\mathrm{arm}_{ij} = \mathrm{IG}] + u_j + \epsilon_{ijk}, \quad u_j \sim \mathcal{N}(0, \sigma_u^2),\ \epsilon_{ijk} \sim \mathcal{N}(0, \sigma_\epsilon^2).$$

The fitted arm contrast is $\hat{\beta}_1 = 14.17$ pp ($\mathrm{SE} = 4.33$, $z = 3.27$, $p = 0.001$; with 12 participants a normal reference is anti-conservative, so we rely on the Welch result). The estimated participant random-effect variance is $\hat{\sigma}_u^2 = 53.3$ and the residual variance is $\hat{\sigma}_\epsilon^2 = 8.8$, giving an intraclass correlation of $\mathrm{ICC} = 0.86$. The high ICC indicates that nearly all of the score variance is between-participant rather than within-participant, which justifies aggregating to per-participant means in the primary analysis.

Sensitivity (minimum detectable effect). A two-sample t-test with $n_1 = n_2 = 6$ at $\alpha = 0.05$ (two-sided) achieves power ≥ 0.80 for standardized effect sizes $d \gtrsim 1.8$. We do not claim power to detect smaller effects; a larger replication remains an important direction for future work.

APEX-Agents intervention study. Sixteen participants (8 per arm) completed the SSB design on APEX-Agents (236-trace corpus, 20 development tasks, 144 held-out test tasks, two-hour budget). Per-arm means are 20.49% (SE 0.63 pp) for IG and 18.40% (SE 0.75 pp) for CC Subagents, against an unmodified-scaffold baseline of 19.44%. Table 11 reports the tests on the 16 per-participant means. At $n = 8$ per arm a two-sample t-test at $\alpha = 0.05$ has power ≥ 0.80 only for standardized effects $d \gtrsim 1.5$, so the study was underpowered for the observed $d = 1.06$. The percentile bootstrap interval excludes zero, but at $n = 8$ per arm it is less conservative than the three tests, none of which reaches $p < 0.05$.

Table 11: Significance tests for the APEX-Agents Expert Intervention Study, IG vs. CC Subagents on the 16 per-participant means ($n = 8$ per arm). Bootstrap interval in percentage points (pp).

Test	Statistic	p
Two-sided Welch t	$t = 2.121$	0.053
Mann–Whitney U (exact)	$U = 14$	0.065
Exact permutation (all 12,870 label assignments)	–	0.066
Bootstrap 95% CI of the difference	$[+0.26, +3.82]$ pp	–
Cohen's d	1.06	–

A.11 Expert Rating Study: Per-Dimension Scores

Table 12: Expert Rating Study on SSB: mean 1–5 ratings by 12 evaluators over every insight of one IG report ($N = 11$) and one CC Subagents report ($N = 12$), per dimension and averaged (Avg). Parentheses: SEM across insights, after averaging each insight over the 12 evaluators.

Report	N	Correctness	Depth	Evidence	Actionability	Avg
CC Subagents	12	4.278 ($\pm$0.08)	4.139 ($\pm$0.09)	4.285 ($\pm$0.08)	4.174 ($\pm$0.12)	4.219 ($\pm$0.07)
IG	11	4.136 ($\pm$0.12)	4.311 ($\pm$0.10)	4.341 ($\pm$0.12)	4.220 ($\pm$0.11)	4.252 ($\pm$0.10)

A.12 Automated Patcher Loop: Protocol and Per-Round Detail

A.12.1 Protocol

With $\mathcal{A}_0$ as the base scaffold, at round r, traces $\mathcal{T}_r$ are collected by executing scaffold $\mathcal{A}_r$ against the benchmark training split. The analysis system under test produces a report R_r from $\mathcal{T}_r$, which the patcher (Claude Code with Claude Opus 4.6 as the underlying model) consumes alongside the scaffold source code to produce $\mathcal{A}_{r+1}$. Each round's modified scaffold is evaluated on a validation split; the loop terminates when validation pass rate fails to improve by $\varepsilon = 0.01$ for two consecutive rounds, or after five rounds on SSB (APEX-Agents settings below), whichever comes first. We report held-out test pass rate at each round.

A.12.2 Conditions

On SSB we compare four conditions (APEX-Agents adds a traces-only condition, Appendix A.12.7): (i) the IG report; (ii) RLM (Zhang et al., 2025a); (iii) CC Subagents (the same analysis system used in Sections 4.2 and 4.3); and (iv) a pure patcher, in which the patcher receives only the scaffold source code, isolating what the patcher achieves without any analysis input.

A.12.3 Best held-out scores on SSB and APEX-Agents

Table 13 summarizes the best held-out score each arm reached on the two benchmarks; the SSB per-round test and validation trajectories follow (Tables 14 and 15), and the APEX-Agents budget grid, component ablations, and budget scaling are in the APEX-Agents subsection below.

Table 13: Automated patcher loop: best held-out score per arm. SSB: held-out pass rate at the best round (round in parentheses; five-round cap). APEX-Agents: uncapped analysis budget, tasks solved of 144 with Pass@1 in parentheses. Unmodified SSB: round-0 pass rate of the IG arm (0.41–0.44 across arms, Table 14); unmodified APEX-Agents scaffold is pooled over five trials. Bold: best per column.

Arm	SSB pass rate (round)	APEX-Agents (of 144)
Unmodified scaffold	0.41	30.8 (0.214)
IG report	**0.84** (4)	**42 (0.292)**
CC Subagents report	0.81 (3)	32 (0.222)
RLM report	**0.84** (3)	32 (0.222)
Pure patcher (no traces, no report)	0.80 (1)	35 (0.243)

A.12.4 PER-ROUND TRAJECTORIES

Table 14: Patcher-loop trajectory on SSB: held-out test pass rate of scaffold $\mathcal{A}_r$ after round r. "–": loop stopped earlier on the validation criterion. Bold: row-wise peak.

Method	$\mathcal{A}_0$	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$
IG	0.41	0.77	0.81	0.80	**0.84**	0.80
RLM	0.42	0.79	0.81	**0.84**	–	–
CC Subagents	0.44	0.76	0.71	**0.81**	–	–
Pure patcher	0.41	**0.80**	0.58	0.58	–	–

Table 15: Patcher-loop validation pass rate on SSB for scaffold $\mathcal{A}_r$ after round r; this series drives the patience stopping rule (test pass rates in Table 14). Bold: row-wise peak.

Method	$\mathcal{A}_0$	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$
IG	0.46	0.78	0.84	**0.86**	**0.86**	0.84
RLM	0.22	**0.82**	0.80	**0.82**	–	–
CC Subagents	0.24	**0.86**	0.68	0.78	–	–
Pure patcher	0.42	**0.72**	0.56	0.48	–	–

A.12.5 IG REPORT CONTENT EVOLVES ACROSS ROUNDS

The findings IG introduces at successive rounds describe failure modes at progressively finer levels of granularity. At round 1, the dominant finding identifies the agent writing Excel formula strings into cells using `openpyxl`, which stores them as literal text rather than evaluating them, a pattern accounting for 84% of round-1 failures. At round 3, the report identifies a scaffold control-flow bug in which a fallback path executes the model's natural-language response as Python when no fenced `python` block is present. At round 5, the report identifies an `openpyxl` API version-skew (the agent uses `wb.defined_names.definedName`, removed in the installed version), and a save-and-reload sequence that shifts row indices and breaks formula references in cells written earlier in the same task. Across rounds, IG's report length grows from 4,074 to 4,816 words and the number of distinct findings grows from 7 to 9.

A.12.6 PURE PATCHER ROUND-2 REVERSAL

The pure-patcher condition reaches the highest round-1 test score in the cohort (0.80) and then regresses to 0.58 at round 2 and again at round 3. We trace the round-2 regression to a single round-2 prompt edit: the round-2 patcher, lacking a regenerated analysis input, added the instruction "If you are unsure about a value, write your best estimate rather than leaving it as None — an approximate value is better than a missing one." SSB grades cells by exact-value match, including expected `None` values for legitimately blank answer cells; the round-1 scaffold passed those tasks by leaving the cells empty. The round-2 instruction directs the agent to fill them with heuristic guesses, and

12 of the 13 round-2 regressions exhibit the eval detail `expected None, got <sentinel>` (where the sentinel is a whitespace character, the string `"N/A"`, or a hallucinated value). Without an analysis input grounded in the round-1 traces, the patcher's round-2 priorities are derived from the scaffold source and its own priors, with no benchmark-grounded check; the resulting edit is confidently structural but contradicts the eval contract.

A.12.7 APEX-AGENTS LOOP, BUDGET GRID, AND COMPONENT ABLATIONS

The APEX-Agents runs use the loop above with the analysis system's per-round LLM spend capped at \$10, \$25, or \$50, or left uncapped; realized spend is approximate and occasionally deviates from the nominal cap. The loop stops after two rounds without a validation improvement of at least 0.01, up to a maximum of eight rounds (the SSB loop was capped at five). In addition to the four SSB conditions, APEX-Agents includes a traces-only condition in which the patcher receives the raw training traces but no report; it is reported here as a control rather than as a comparison system. Table 16 reports the best held-out test score per arm, Table 17 the component ablations at the \$50 cap with their judged report quality, and Table 18 the within-IG budget scaling. In the ablation-pool tournament each arm plays eight games and no ties were recorded. The IG loop also recovered from a round-1 regression in which the patcher added a planning step that made the agent emit plans instead of answers: the next IG report identified the failure and the patcher reverted it, whereas the pure patcher did not recover from its breaking change.

Statistical caveats. The unmodified-scaffold baseline is pooled over five independent trials (154/720 tasks, 0.214, SD 2.59 tasks), and each arm's significance against it is a two-sided prediction-interval t-test that treats the arm's single trial as one new draw from the baseline trial distribution (standard error $s\sqrt{1+1/5}$, four degrees of freedom). The reported score is the best held-out result across the loop's rounds and the baseline variance does not include patcher stochasticity, so these p-values are optimistic. IG ($p = 0.017$) and traces-only ($p = 0.045$) are significant at the unadjusted level; after a Holm correction across the five uncapped arms neither remains so. IG's three-task margin over traces-only is within the baseline's single-draw noise (SD 2.59 tasks), and its ten-task margin over the CC Subagents and RLM reports is a difference between two single trials (standard error $2.59\sqrt{2} \approx 3.7$ tasks, $t \approx 2.7$ on four degrees of freedom, two-sided $p \approx 0.05$). The IG loop completed seven rounds before the patience rule stopped it, against four for CC Subagents and RLM, which gives it more rounds from which the best score is taken. Across the capped grid every cell lies within 27–35/144, the best capped cell (RLM at \$25, 35/144) is 1.6 baseline standard deviations above the pooled baseline, and neither CC Subagents nor RLM improves monotonically as the cap rises; realized spend was not matched, since RLM stayed near \$12 per round at every cap whereas CC Subagents spent up to its cap. Among the \$50-cap ablations, only the no-LLM-extraction arm (19/144) lies outside single-trial baseline noise; the other removals cost two to four tasks, within it.

Table 16: APEX-Agents automated patcher loop: best held-out test score (tasks solved of 144; Pass@1 in parentheses) per analysis system and per-round analysis cap. Unmodified scaffold: 30.8/144 (0.214), pooled over five trials, SD 2.59 tasks. *Above the baseline prediction interval at $p < 0.05$ (two-sided; unadjusted). Bold: best cell.

Analysis system	\$10 cap	\$25 cap	\$50 cap	Uncapped
IG	33 (0.229)	34 (0.236)	34 (0.236)	**42 (0.292)***
CC Subagents	27 (0.188)	33 (0.229)	27 (0.188)	32 (0.222)
RLM	34 (0.236)	35 (0.243)	32 (0.222)	32 (0.222)
Traces only (no report)	–	–	–	39 (0.271)*
Pure patcher (no traces, no report)	–	–	–	35 (0.243)

Table 17: IG component ablations on APEX-Agents. Patcher: best held-out test score (of 144) at a $50 per-round analysis cap. WR: pairwise win rate (%) within the five-report ablation pool only (full IG and the four ablations), so not comparable to Table 1; Comp.: rubric composite (0–3) against the union-gold set of the eight-system APEX judge run (Table 10). Both are shown under the Opus 4.6 and GPT-5.4 judges and score each arm's uncapped report rather than its $50-cap run. Comp. does not penalize citations of trace IDs absent from the corpus (18.2% of the no-LLM-extraction report's findings, 0% for full IG), whereas WR and Patcher do.

		WR (%), ablation pool		Comp. (0–3)	
Arm	Patcher	Opus 4.6	GPT-5.4	Opus 4.6	GPT-5.4
Full IG	34	87.5	100.0	1.50	1.49
Orchestrator-only (no dispatch)	32	25.0	37.5	0.92	0.95
Generic Subagents (no typing)	31	75.0	75.0	1.47	1.38
No Python REPL	30	0.0	12.5	1.03	1.23
No LLM extraction	19	62.5	25.0	1.68	1.76
Unmodified scaffold	30.8				

Table 18: IG budget scaling on the APEX-Agents patcher loop. Best test: best held-out score (of 144). WR: pairwise win rate (%) within a five-report pool of IG reports (the four tiers shown plus a $100-cap report, not shown), so not comparable to Table 1; Comp.: rubric composite (0–3) from the same judge run over the tier reports, which builds its own union-gold set and is therefore not comparable to Table 10. WR and Comp. are shown under the Opus 4.6 / GPT-5.4 judges. Scouts, Investigators (Invest.), and Orchestrator (Orch.) turns are per-round means; Cite valid.: fraction of cited trace IDs that resolve; count: citations per report.

			WR (%)	Comp.				
IG cap	Rounds	Best test	Opus / GPT	Opus / GPT	Scouts	Invest.	Cite valid., count	Orch. turns
$10	5	33	12.5 / 0.0	1.24 / 1.24	1	0	81.6%, 111.8	16.5
$25	4	34	25.0 / 25.0	1.36 / 1.44	5	0	84.3%, 129	16.8
$50	4	34	50.0 / 62.5	1.61 / 1.86	8.67	1.67	95.7%, 256.7	19.3
Uncapped ($174/round)	7	42	87.5 / 75.0	1.87 / 1.83	8.33	12	99.8%, 633.3	35.2

A.13 PATCHER PROMPT

The Patcher prompt template is rendered at runtime with conditional sections for the four input conditions (codebase only; codebase with traces; codebase with report; codebase with report and traces); the template below shows the form with both an IG report and the trace bundle present. For the codebase-only condition the IG-report and traces sections are dropped. The benchmark-specific specification is appended verbatim after these instructions.

```
You are a coding agent tasked with improving an agent
scaffold to maximize its benchmark performance.
Your workspace.  The source directory contains the agent
scaffold source code -- this is what you modify.  An IG
report file contains an analysis of the agent's execution
traces with identified failure modes and recommended fixes;
start by reading it.  A traces directory contains the
agent's execution traces; use them to understand where the
agent fails and why.
Workflow.  Approach this rigorously -- plan first, then
implement.  Premature edits produce shallow patches that
don't generalize.  Most of your effort should be in steps
1-2.
(1) Explore.  Get a thorough understanding of the scaffold
code -- its entry points, the tool surface, and where the
agent's behavior is configured vs hard-coded.  Read the IG
report carefully for the failure-mode analysis and
recommended interventions.  Examine traces to identify
```

```
concrete failure patterns from real runs.  You may dispatch
focused-exploration subagents and use a TodoWrite tool to
track your plan as you progress.
(2) Diagnose and plan.  For each failure mode, name (a) the
specific scaffold layer that's wrong (system prompt, tool
definition, control flow, retry policy, response synthesis);
(b) the mechanism -- why the current code produces that
failure (not just the surface symptom); and (c) the smallest
concrete change that fixes it.
(3) Implement.  Apply the changes from your plan.  Prefer
small, targeted edits over rewrites.
(4) Evaluate.  Call the evaluate-and-submit tool to test
your changes.  With one attempt available, make your best
changes then submit.  With multiple attempts, you receive
evaluation results after each call and can iterate.
Constraints.  Only modify files in the source directory.  Do
not access anything outside it.  Do not change the model or
LLM configuration.  Do not use web search or web fetch.  Do
not access, read, or inspect any evaluation code, test data,
ground truth, or benchmark task definitions.  Do not attempt
to read task metadata, answer keys, or expected outputs from
the environment object.  Focus on scaffold changes:  system
prompts, tool implementations, error handling, control flow,
response synthesis.
```

A.14 Cost and Runtime Detail

Table 19 reports per-system LLM cost and wall-clock for a single analysis run, averaged over n=6 runs (3 replicates $\times$ 2 benchmarks: SSB and HLE). Costs are imputed from token counts at Anthropic's published list price (`claude-opus-4-6`: \$5/M input, \$25/M output; `claude-haiku-4-5`: \$1/M input, \$5/M output) where the source schema does not persist a dollar field; otherwise costs are taken verbatim from the harness's billing telemetry. Wall-clock is the elapsed time of the analysis run itself, from the same source records.

Table 19: LLM cost and wall-clock per analysis run, mean over n=6 runs (three replicates each on SSB and HLE). *: cost imputed from recorded token counts at Anthropic list prices (Claude Opus 4.6: \$5/M input, \$25/M output; Claude Haiku 4.5: \$1/M input, \$5/M output); other costs are recorded by the harness. IG's wide range spans SSB ($\sim$\$17 mean) and the longer-trace HLE ($\sim$\$71 mean).

	LLM cost (\$)			**Wall-clock (min)**		
System	**Mean**	**Median**	**Range**	**Mean**	**Median**	**Range**
RLM*	7.72	7.36	5.09 – 9.96	19.5	17.5	10.6 – 29.3
Trace2Skill	26.43	25.63	22.22 – 32.49	37.3	35.8	28.6 – 53.4
CC Subagents	38.09	38.05	33.00 – 42.95	34.4	34.3	27.2 – 42.1
IG*	44.21	42.19	12.17 – 81.55	48.4	48.6	38.1 – 56.4