Agentic CLEAR: Automating Multi-Level Evaluation of LLM Agents

Asaf Yehudai[I*], **Lilach Eden**[I*], **Michal Shmueli-Scheuer**[I]

[I]IBM Research
Asaf.Yehudai@ibm.com, {lilache, shmueli}@il.ibm.com

Abstract

Agentic systems are becoming more capable: agents define strategies, take actions, and interact with different environments. This autonomy poses serious challenges for overseeing and assessing agent behavior. Most current tools are limited, focusing on observability with basic evaluation or creating a static taxonomy of agent errors. To address this gap, we present Agentic CLEAR, an automatic, dynamic, and easy-to-use evaluation framework. It produces textual insights into the agent behavior on three levels of granularity: system, trace, and node. Agentic CLEAR operates above the observability layer, enabling seamless integration and featuring an intuitive UI that makes agent evaluation highly accessible. In our experiments on four benchmarks, seven agentic settings, and tens of thousands of LLM calls, we show that Agentic CLEAR produces high-quality, data-driven, insightful feedback. Our analysis shows strong alignment with human-annotated errors and the ability to predict task success rate.

Code: https://ibm.biz/ACLEAR-Code

1 Introduction

Agentic systems have become increasingly capable of defining strategies, executing actions, interacting with external environments, and solving complex, multi-step tasks (Schick et al., 2023; Wang et al., 2024). This success has driven widespread adoption across various domains, including software engineering (Anthropic, 2025), scientific discovery (Ghafarollahi and Buehler, 2025), and open-ended web browsing (OpenAI, 2025). Crucially, this paradigm shift is not limited to large-scale enterprise solutions. Individual developers are adopting agentic workflows to automate bespoke, day-to-day tasks. However, despite this democratization of agent building, agentic systems remain inherently brittle. They frequently exhibit subtle failure modes, repeated loops, misaligned sub-agent behavior, and error propagation across steps that are hard to detect from final outputs alone.

This pressing need for oversight has led to the proliferation of agent observability platforms (e.g., LangSmith, LangFuse). While invaluable for logging execution traces, their evaluation capabilities are largely limited to basic metric aggregation or coarse, single-prompt LLM-as-a-judge assessments applied to the full trace. Consequently, developers are still required to manually inspect large numbers of traces to identify systemic issues. In parallel, the research community has focused on constructing agent error taxonomies (Cemri et al., 2026; Zhu et al., 2026; Deshpande et al., 2025) and high-fidelity benchmarks (Jimenez et al., 2024; Yehudai et al., 2025a). Yet, these approaches yield static, rigid categories or require extensive, handcrafted engineering that cannot dynamically adapt to the bespoke tasks faced by everyday agent developers.

In this work, to bridge this gap, we present Agentic CLEAR, an automatic, dynamic, and easy-to-use evaluation method that produces rich, textual insights into agent behavior. Agentic CLEAR evaluates each trace, producing step-level and full-trace feedback, and then aggregates them across the full collection of execution traces to surface recurrent failures, quality degradation, and issues (See §2). Our approach produces structured, textual diagnostics across three levels of granularity, the system, node, and trace levels, enabling developers to quickly understand not only *what* failed, but *why*.

We provide Agentic CLEAR as a pip-installable Python package designed for easy integration into existing agent development workflows (See §3.1). It also provides an intuitive interactive UI for deep-dive trace analysis (See §3.2). Through experiments on diverse traces drawn from leading benchmarks and prominent agent architectures, we demonstrate that Agentic CLEAR delivers action-

[*]Equal contribution.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 750–763

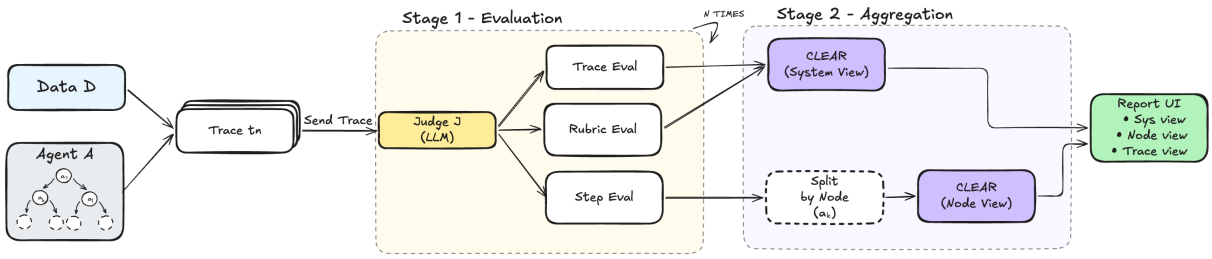

Figure 1: Agentic CLEAR Pipeline. We start by preparing the execution traces. Stage 1: Apply multi-level per-trace evaluation via an LLM Judge. Stage 2: Aggregate insights using CLEAR, split into System-wide patterns and Node-specific patterns, and prepare them for the UI.

able, high-level insights without requiring handcrafted evaluation rubrics or extensive human annotation (See §5). By lowering the barrier to meaningful agent evaluation and diagnostics, Agentic CLEAR supports faster iteration, improved reliability, and more systematic understanding of agent behavior across tasks and domains.

In summary, our contributions are:

1. **A Dynamic Evaluation Methodology:** We introduce a multi-level method that emphasizes automatic, dynamic, and granular evaluation insights.

2. **Open-Source Package:** We provide a Python package with easy integration and an interactive visual dashboard.

3. **Empirical Validation:** We demonstrate the efficacy of Agentic CLEAR across varied benchmarks, agents, and models, showing its ability to surface execution failures without human-engineered tests.

We hope that Agentic CLEAR will serve the broader NLP and software engineering communities, fostering faster iteration, improved agent reliability, and the development of next-generation evaluation tools.

2 Agentic CLEAR Method

Agentic CLEAR generates multi-level feedback by analyzing the agentic system behavior across an entire dataset. As described in Figure 1, the pipeline ingests execution traces and outputs insights at the system, node, and trace levels.

Formally, let $\mathcal{D} = \{x_n\}_{n=1}^{N}$ be a dataset of N tasks and A be a target agentic system (i.e., a multi-agent system) composed of distinct nodes (e.g., sub-agents or components, depending on the development framework). Invoking A on task x_n yields an execution trace $t_n = \{(i_k, o_k, a_k)\}_{k=1}^{K_n}$, consisting of a sequence of LLM calls, where each call is divided into an input and an output pair, $(\{(i_k, o_k, a_k)\})$, produced by a specific node a_k, as dictated by the agent structure and execution flow. Overall, by running the agent on $\mathcal{D}$, we get the resulting traces, denoted as $\mathcal{T} = \{t_n\}_{n=1}^{N}$.

Given this data, our evaluation proceeds in two stages: trace evaluation and system-level aggregation. As outlined in Algorithm 1, first, for every trace t_n, we employ an LLM judge J to perform three assessments: (1) **Step-wise Evaluation:** For each pair (i_k, o_k), J_s produces a quality score and a natural language critique (subscript notation indicates different evaluation modes of J). (2) **Trace-wise Evaluation:** Similarly, J_t evaluates the quality of the complete trace, taking into account step and full trace considerations. (3) **Rubric Evaluation:** We apply a two-step assessment. First, given x_n, the judge J_r generates a set of task-specific criteria/rubrics required to accomplish the task. Then, based on x_n and the generated rubrics r_n, the judge J_v assesses whether these criteria were met within the trace t_n.

In the second stage, to identify high-level insights, we leverage CLEAR (Yehudai et al., 2025b) to cluster and summarize the instance-level feedback into global insights. For each node (a_k), we group input-output pairs associated with it, and apply CLEAR to surface component-specific failures $(\mathcal{I}_{node})$. Similarly, we aggregate trace-level judgments to identify holistic system behaviors $(\mathcal{I}_{sys})$. Finally, we also link each insight to the specific execution step or trace that triggered it.

This hierarchical approach delivers clear, interpretable insights across multiple levels of granularity, giving the agent developer visibility into the system at different resolutions, from fine-grained nodes and traces to the full system view.

Algorithm 1: Agentic CLEAR Insight Generation Pipeline

Input: Dataset $\mathcal{D} = \{x_n\}_{n=1}^{N}$; Agent A; Judge J; Aggregator CLEAR
Output: System Insights $\mathcal{I}_{sys}$, Node Insights $\mathcal{I}_{node}$, Trace Evaluations $\mathcal{E}_{trace}$

$\Phi_{node} \leftarrow \emptyset$; $\Phi_{sys} \leftarrow \emptyset$ // Init feedback containers
/* Stage 1: Execution & Granular Evaluation */
for $n \leftarrow 1$ **to** N **do**
  $t_n \leftarrow \text{Execute}(A, x_n)$ // Trace $t_n = \{(i_k, o_k, a_k)\}$ with inputs, outputs, nodes
  // 1. Node-wise Evaluation
  foreach *step* k *in* t_n **do**
    $c_{n_k}^{node} \leftarrow J_s(i_k, o_k)$ // Critique individual step
    $\Phi_{node}[a_k] \leftarrow \Phi_{node}[a_k] \cup \{c_{n_k}^{node}\}$ // Group by node a_k
  // 2. Trace-wise Evaluation
  $c_n^{trace} \leftarrow J_t(t_n)$ // Holistic trace critique
  // 3. Rubric Evaluation
  $r_n \leftarrow J_r(x_n)$ // Generate task-specific criteria
  $c_n^{rubric} \leftarrow J_v(t_n, r_n)$ // Check compliance
  $\Phi_{sys} \leftarrow \Phi_{sys} \cup \{c_n^{trace}, c_n^{rubric}\}$ // Collect for system view
  $\mathcal{E}_{trace}[n] \leftarrow \{c_n^{trace}, c_n^{rubric}, c_{n_k}^{node}\}$
/* Stage 2: Insight Aggregation via CLEAR */
$\mathcal{I}_{node} \leftarrow \emptyset$
foreach *node* a *in* Φ_{node}.*keys* **do**
  $\mathcal{I}_{node}[a] \leftarrow \text{CLEAR}(\Phi_{node}[a])$ // Per-node recurring patterns
$\mathcal{I}_{sys} \leftarrow \text{CLEAR}(\Phi_{sys})$ // Global system patterns
return $\mathcal{I}_{sys}, \mathcal{I}_{node}, \mathcal{E}_{trace}$

3 Agentic CLEAR Framework

3.1 Pipeline

To allow easy integration and usability, we provide Agentic CLEAR as a Python package available on PyPI (Permissive Apache 2.0 License). The package supports the different end-to-end evaluation levels described in §2. Each evaluation level in the pipeline can be used on its own or combined with the others, allowing users to tailor the workflow to their specific evaluation needs and preferences.

For easy onboarding, we adopt an OpenTelemetry[1]-compatible format. Specifically, we utilize LangFuse-formatted[2] traces, which we convert to an intermediate representation that serves as input to the pipeline. For other trace formats, we require only minimal preprocessing to reach the same intermediate state that captures the LLM call's inputs and outputs in the trace, along with the necessary metadata. We focus our analysis on the LLM interactions, as they govern the system's decision-making and are its most stochastic element.

We design specific prompts for each judge evaluation mode. For J_s, the judge assesses step-level aspects such as correctness, completeness, and clarity. For J_t, we extend these criteria to trace-level dimensions, including execution quality and the final deliverable. In J_r, the judge needs to decide on the number of rubrics and generate them to suit the given task. Each prompt elicits a brief textual justification prior to the score, functioning as a chain-of-thought rationale. While our method primarily focuses on providing textual insights, we also surface these quantitative scores in the UI. When ground-truth evaluation scores are available for each trace, the system generates further insights into execution paths and predictive patterns of trace success, and additionally assesses the reliability of the judge. All the prompts are presented in App. A. To support customization, users can adjust the evaluation dimensions, override the prompts, or replace the judge with a custom Python implementation.

Code We provide Agentic CLEAR as a PyPI package. The analysis can be executed with a single CLI command, configured via a YAML file. Once processing completes, the interactive interface can be launched from the command line. The pipeline stores its results as a ZIP file in the designated output directory, which can then be loaded manually into the app.

[1] OpenTelemetry
[2] LangFuse

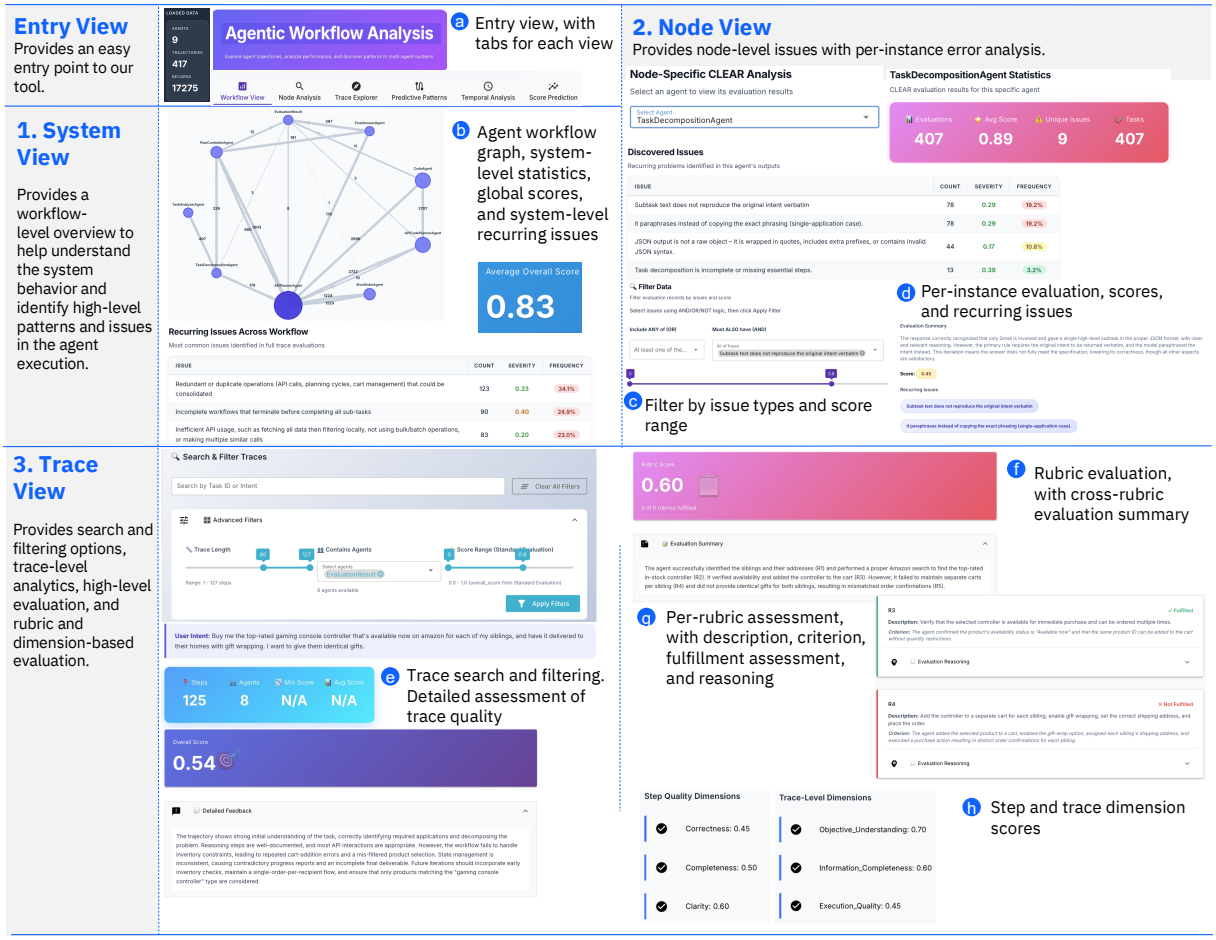

Figure 2: The interactive UI of Agentic CLEAR, enabling multi-granular evaluation and diagnosis of agentic workflows. (a) The entry module provides top-level navigation across different analysis tabs. (b) The **System View** offers a macro-level summary, visualizing agent topologies, global performance scores, and system-wide recurring issues. The **Node View** facilitates agent-specific error analysis via (c) issue- and score-based filtering to isolate relevant evaluations, alongside (d) per-instance scoring and error distributions. The **Trace View** enables fine-grained, instance-level inspection, featuring (e) trace search and filtering capabilities, (f) cross-rubric evaluation summaries, (g) detailed per-rubric assessments with fulfillment reasoning, and (h) granular step- and trace-level dimension scores.

```
$ pip install clear-eval
```

3.2 Agentic CLEAR UI

Agentic CLEAR dashboard (Figure 2) provides a hierarchical visual suite. We designed it to move beyond static telemetry, enabling agent developers and researchers to diagnose agent behaviors across levels. The interface is structured around three primary perspectives:

System Level This view dynamically reconstructs the multi-agent topology directly from execution traces. It presents high-level agent behavioral patterns, like node usage and flow dynamics. Finally, it aggregates global performance scores and surfaces systemic recurring issues.

Node View This view allows navigating between agent nodes. For each, it presents the dynamically generated issues the node exhibits. Users can filter steps by issue types and score ranges. This allows targeted inspection of per-instance error distributions, surfacing recurring patterns localized to individual prompts or behaviors.

Trace View Facilitating fine-grained analysis, the Trace View unpacks individual execution traces. It presents overall trace evaluation, alongside granular, step-level dimension scores, and rubric evaluation. Crucially, it exposes the LLM judge's natural language reasoning for each assessment, providing users with interpretable, context-aware justifications for every identified failure mode.

Benchmark	Agent	Model	# Traces	Source
AppWorld	CUGA	GPT-4o	417	Leaderboard
GAIA	Generalist Agent	Claude 4.5 Sonnet	165	HAL
	Generalist Agent	GPT-4.1	165	HAL
	HF DeepResearch	Claude 4.5 Sonnet	165	HAL
	HF DeepResearch	OpenAI o3	117	TRAIL
SWE-bench Ver. Mini	Generalist Agent	Claude 4.5 Sonnet	50	HAL
TAU-bench (Airline)	Generalist Agent	Claude 3.7 Sonnet	50	HAL

Table 1: Data statistics for the curated traces

4 Experimental Setup

To rigorously evaluate Agentic CLEAR across diverse settings, we curate execution traces generated by leading agent architectures and LLMs across prominent benchmarks. Specifically, we take traces from the following benchmarks: SWE-Bench Verified Mini (Jimenez et al., 2024), GAIA (Mialon et al., 2023), AppWorld (Trivedi et al., 2024), and τ^2-Bench (Barres et al., 2025). The agents are CUGA (Marreed et al., 2025), the SOTA agent on AppWorld, HAL generalist agent (Kapoor et al., 2026), and Hugging Face's Open Deep Research agent (Roucher et al., 2025), with top OpenAI and Anthropic models (See Table 1).

We collect traces from HAL (Kapoor et al., 2026), TRAIL (Deshpande et al., 2025), and the AppWorld leaderboard, and consolidate them into our unified intermediate representation schema. We select seven settings to support comparative analyses across models, agents, and benchmarks. We present detailed descriptions of the benchmarks, the evaluated agents, and the specific trace datasets in Appendix B.

As judges, we employ two leading models, OSS-120B (OpenAI et al., 2025) in high thinking mode as a representative of a leading open-source model, and GPT-5 (Singh et al., 2025) as a closed-source model.

We perform trace-wise evaluation across all seven trace datasets using two judge models. The resulting evaluations are then passed to the CLEAR aggregation stage for issue discovery.

5 Agentic CLEAR Issues Results

In the following, we report findings on the universal failure patterns, the effect of the agent architecture and the backbone model, benchmark-specific issues, and the impact of judge selection.

Universal Error Patterns Several recurring issue categories appeared among the 195 trace-level issues generated across all configurations, reflecting systemic weaknesses in current agent systems: (1) Redundant and Inefficient Tool Usage: unnecessary repeated calls, poorly designed queries, or wasted computation; (2) Insufficient Error Handling and Recovery: agents frequently failed to recover from tool errors or to shift to alternative strategies after failure and lacked effective fallback mechanisms; (3) Incomplete Workflows: agents failed to bring tasks to completion and fulfill all goals; (4) Output Formatting and Schema Compliance: agents failed to adhere to output formats.

Domain-Specific Issues (a) System-Level: Beyond these shared errors, each benchmark displayed its own domain-specific weaknesses. GAIA, a research-oriented benchmark, was dominated by **sourcing and verification failures** (e.g., *"Lack of cross-verification across independent sources"*); AppWorld, which tests multi-step API orchestration, exhibited unique failures such as **incomplete executions and domain-specific workflow breakdowns** (e.g., *"acting on contaminated shopping carts and dropping email attachments"*); Results on SWE-Bench Verified Mini highlight code-related issues, such as **monkey-patching and broken diff output**, while τ^2-Bench focused on **policy violations** (e.g., *"unauthorized payment selection, fabricated cost estimates"*). Notably, Agentic CLEAR discovered these domain-specific issues without any benchmark-specific prompting.
(b) Node-level: This differentiation extends further at the node level. Running our method on the CUGA agent reveals that while universal issues like **JSON malformation** appeared across nearly all nodes, different nodes surfaced distinct failure types matching their role: planning nodes were dominated by **task decomposition** and **API selection** issues (e.g., TaskDecompositionAgent: *"subtasks are ordered illogically or not in a natural execution sequence"*), while execution nodes surfaced functional bugs (e.g., APICodePlannerAgent: *"missing pagination handling for APIs that return multiple pages of results"*). Moreover, this evaluation mode allows pinpointing specific pitfalls behind each failure mode and addressing them directly. For example, hallucinations occur mainly during the planning stages (e.g., ShortlisterAgent: *"APIs not defined in the supplied API catalog are listed"*) but not during execution. Insights like these help agent developers fine-tune the relevant components more effectively. See Appendix C for concrete examples of both cross-benchmark and cross-level issue variations.

Backbone Model and Agent Differences Comparing GPT-4.1 and Claude 4.5 Sonnet as backbones for the HAL agent on GAIA (judged by GPT-5), the two models shared the majority of their system-level failure profile: both were flagged for **source verification gaps, tool misuse, and output formatting noncompliance**. For instance, both produced nearly identical issues around output compliance (GPT-4.1: *"noncompliance with required execution and output formats/protocols"*; Claude 4.5 Sonnet: *"failure to adhere to output formatting and deliverable specifications"*). However, each also exhibited unique tendencies: GPT-4.1 was flagged for *"prematurely giving up after errors instead of diagnosing, retrying, or pivoting to alternatives"*, while Claude 4.5 Sonnet was associated with *"contradictory or self-conflicting statements; does not commit to a consistent interpretation"*. Similarly, comparing the HF DeepResearch and HAL agents with Claude as the backbone over GAIA reveals a largely shared error profile, with some small distinctions, suggesting the dataset has a greater effect than the agent architecture on the error types.

Judge Selection Both judges were consistently able to uncover diverse and non-trivial recurring issues. However, they produced qualitatively different diagnoses, even of the same agent behavior. Their output differed not only in wording but also in depth, specificity, and the behavior they chose to emphasize. OSS-120B tended to generate shorter issues (67 vs. 130 characters on average) and to surface broader and more generic categories, more focused on operationally oriented failures (e.g., "Redundant searches and file inspections causing inefficiency" or "Misused tool arguments or invoked the wrong tool" on SWE-Bench Verified Mini). In contrast, GPT-5 produced longer, more nuanced, and domain-specific failure modes that more frequently targeted verification and validation failures, incorrect logic or reasoning, and methodological correctness (e.g., "breaks SQL query correctness due to missing alias remapping when combining SQL components"). These findings suggest that judge selection is consequential for determining the specificity and depth of the generated failures.

6 Analysis

We validate Agentic CLEAR through two complementary analyses. The first compares our issues against human-annotated errors. The second compares our score prediction methods with a few ground-truth benchmarks' labels.

Method	Micro F1	Macro Cat F1
Random (GT freq)	0.342	0.288
Always top-4	0.459	0.199
OSS-120B (full)	0.377	0.261
OSS-120B (full+partial)	0.427	0.374
GPT-5 (full)	0.467	0.368
GPT-5 (full+partial)	**0.497**	**0.459**

Table 2: Error category prediction performance against TRAIL (Planning and Reasoning categories).

Alignment with Human Error Taxonomies To validate that our automatically generated issues capture meaningful error patterns, we first perform a semantic mapping between our generated issues and TRAIL categories (Deshpande et al., 2025). TRAIL provides a hierarchical taxonomy of 20 error categories spanning reasoning, planning, and system execution failures. Here, we use the 12 non-execution categories as Agentic CLEAR focuses on LLM reasoning and planning. These categories account for 94% of the ground-truth labels.

Since our issues are taxonomy-free by design, we first apply a semantic alignment: we map each of our system-level issues into the TRAIL categories as either a full match (directly corresponding to a TRAIL category), or a partial match (overlaps conceptually but covers a broader or adjacent concern). The full mappings between the issues produced by both judges and the TRAIL taxonomy are presented in Appendix D.

The mapping was performed using Claude Opus 4.6 and verified by the authors. All 15 GPT-5 issues and all 12 OSS-120B issues map to at least one TRAIL category, collectively covering 12 and 10 of the 12 relevant categories, respectively.

To verify that the alignment holds at the instance level, i.e., traces flagged with issues by Agentic CLEAR exhibit the corresponding TRAIL errors, we propagate the mapping transitively to individual traces (117 in total) and measure agreement. We report macro-averaged F1 as the primary metric, as it equally weights all error categories and thus directly measures breadth of taxonomy coverage. To calibrate, we compare against two baselines: a random predictor weighted by the true category frequencies, and a majority baseline that always predicts the four most common categories.

Table 2 presents the results. The GPT-5 judge achieves the strongest agreement under the full+partial matching, with a macro-F1 of 0.459

			Step-Wise		Trace		Rubric	
Benchmark	**Agent**	**Model**	**OSS-120B**	**GPT-5**	**OSS-120B**	**GPT-5**	**OSS-120B**	**GPT-5**
AppWorld	CUGA	GPT-4o	0.758	0.823	0.837	0.890	0.778	0.828
GAIA	Generalist Agent	Claude 4.5 Sonnet	0.664	0.632	0.712	0.720	0.596	0.566
GAIA	Generalist Agent	GPT-4.1	0.707	0.742	0.839	0.848	0.560	0.597
GAIA	HF DeepResearch	Claude 4.5 Sonnet	0.541	0.546	0.609	0.716	0.537	0.571
GAIA	HF DeepResearch	OpenAI o3	0.783	0.706	0.774	0.819	0.729	0.736
SWE-bench	Generalist Agent	Claude 4.5 Sonnet	0.788	0.779	0.661	0.803	0.505	0.524
TAU-bench	Generalist Agent	Claude 3.7 Sonnet	0.409	0.529	0.618	0.554	0.539	0.597

Table 3: AUC for predicting trajectory success using Agentic CLEAR scores. We report trace-level, rubric-based, and step-wise (average) scores

and micro-F1 of 0.497. The frequency baseline is competitive on micro-F1 (0.459) due to the skewed category distribution, but its low macro-F1 (0.199) indicates that it fails to cover the tail of the error distribution. As expected, the GPT-5 judge outperforms the smaller OSS-120B judge.

Overall, Agentic CLEAR recovers the majority of reasoning and planning error categories without requiring predefined category definitions. The generated issues are often more fine-grained and actionable than the TRAIL categories they map to, capturing specific failure patterns where the taxonomy provides only broad groupings. This suggests that our method can preserve the diagnostic capabilities of expert taxonomies while surfacing more targeted and nuanced insights.

Score Prediction To evaluate our judge's ability to predict trace success, we compute the area under the ROC curve (AUC) between the ground-truth and the predicted scores. Agentic CLEAR provides three methods to predict trace success: (1) **Trace**: the overall score generated by the trace-wise evaluation; (2) **Rubric**: the proportion of task-level rubrics predicted as fulfilled; and (3) **Step-wise**: the average score across all steps within the trace.

Table 3 presents the full results. GPT-5 generally outperforms OSS-120B across the methods. Across configurations, the trace-level method is the strongest predictor, outperforming the step-wise and rubric methods. This likely reflects the fact that the underlying assumptions of each method do not hold uniformly across all settings. The rubric method assumes the task description contains all the requirements to determine success, which breaks for τ^2-Bench when implicit policy adherence is critical. The step-wise method assumes that trace evaluation can be decomposed into the quality of isolated steps, which is more suitable for some agents and benchmarks. For instance, modular agent architectures like CUGA are composed of distinct, self-contained components with clearly defined tasks, making it easier to assess each node's contribution. It can also benefit from a composable task structure, like in SWE-Bench Verified Mini, where the tasks naturally decompose into discrete phases, e.g., locating, understanding, and fixing a bug. These differences suggest that Agentic CLEAR evaluation methods can provide complementary signals depending on the target domain and agent.

Comparing results across benchmarks reveals large variations. AppWorld is the most predictable benchmark, with all results exceeding 0.75, and GPT-5 specifically achieving at least 0.82 AUC with all methods. τ^2-Bench results, on the other hand, do not exceed 0.62, and both GAIA and SWE-Bench Verified Mini exhibit variation depending on the method, agent, and model. These results call for further research to investigate the effectiveness of trace judges in different agentic configurations.

7 Conclusions

We presented Agentic CLEAR, an automatic evaluation framework that produces multi-level textual insights into agent behavior at scale, without requiring predefined error taxonomies or hand-crafted rubrics. Across four benchmarks and seven agentic configurations, we demonstrated alignment with human-annotated errors and meaningful predictive signal for task success. Key directions for future work include extending Agentic CLEAR to analyze system execution alongside reasoning and planning, improving judge capabilities and reliability across diverse agentic settings, and enabling systematic cross-configuration comparisons.

References

Anthropic. 2025. Claude-code.

Elron Bandel, Asaf Yehudai, Lilach Eden, Yehoshua Sagron, Yotam Perlitz, Elad Venezian, Natalia Razinkov, Natan Ergas, Shlomit Shachor Ifergan, Segev Shlomov, Michal Jacovi, Leshem Choshen, Liat Ein-Dor, Yoav Katz, and Michal Shmueli-Scheuer. 2026a. General agent evaluation. *Preprint*, arXiv:2602.22953.

Elron Bandel, Asaf Yehudai, Alexandre Lacoste, Avijit Ghosh, Graham Neubig, Margaret Mitchell, Michal Shmueli-Scheuer, and Leshem Choshen. 2026b. Agentic systems should be general. *SSRN Electronic Journal*.

Elron Bandel, Asaf Yehudai, and Michal Shmueli-Scheuer. 2026c. Ready for general agents? let's test it. In *ICLR Blogposts 2026*. Https://iclr-blogposts.github.io/2026/blog/2026/general-agent-evaluation/.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *Preprint*, arXiv:2506.07982.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2026. Why do multi-agent LLM systems fail? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. 2025. Trail: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*.

Ariel Gera, Odellia Boni, Yotam Perlitz, Roy Bar-Haim, Lilach Eden, and Asaf Yehudai. 2025. Justrank: Benchmarking llm judges for system ranking. *Preprint*, arXiv:2412.09569.

Alireza Ghafarollahi and Markus J Buehler. 2025. Sciagents: automating scientific discovery through bioinspired multi-agent intelligent graph reasoning. *Advanced Materials*, 37(22):2413523.

Harbor Framework Team. 2026. Harbor: A framework for evaluating and optimizing agents and models in container environments.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? *Preprint*, arXiv:2310.06770.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, and 12 others. 2026. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In *The Fourteenth International Conference on Learning Representations*.

Alexandre Lacoste, Nicolas Gontier, Oleh Shliazhko, Aman Jaiswal, Kusha Sareen, Shailesh Nanisetty, Joan Cabezas, Manuel Del Verme, Omar G. Younis, Simone Baratta, Matteo Avalle, Imene Kerboua, Xing Han Lù, Elron Bandel, Michal Shmueli-Scheuer, Asaf Yehudai, Leshem Choshen, Jonathan Lebensold, Sean Hughes, and 7 others. 2026. Cube: A standard for unifying agent benchmarks. *Preprint*, arXiv:2603.15798.

LangFuse. 2023. Langfuse: Observability for ai applications.

LangSmith. 2023. Langsmith: Evaluation framework for ai applications.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stanczak, Peter Shaw, Christopher Pal, and Siva Reddy. 2025. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. In *Second Conference on Language Modeling*.

Sami Marreed, Alon Oved, Avi Yaeli, Segev Shlomov, Ido Levy, Offer Akrabi, Aviad Sela, Asaf Adi, and Nir Mashkif. 2025. Towards enterprise-ready computer using generalist agent. *Preprint*, arXiv:2503.01861.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *Preprint*, arXiv:2311.12983.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, and more. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

OpenAI. 2025. Chatgpt agent: bridging research and action.

Aymeric Roucher, Albert Villanova del Moral, Merve Noyan, Thomas Wolf, and Clémentine Fourrier. 2025. Open-source DeepResearch – Freeing our search agents. Accessed: 2025-02-04.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, and more.

2025. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *Preprint*, arXiv:2407.18901.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6).

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025a. Survey on evaluation of llm-based agents. *Preprint*, arXiv:2503.16416.

Asaf Yehudai, Lilach Eden, Yotam Perlitz, Roy Bar-Haim, and Michal Shmueli-Scheuer. 2025b. Clear: Error analysis via llm-as-a-judge made easy. *Preprint*, arXiv:2507.18392.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Yusheng Su, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. 2026. Where LLM agents fail and how they can learn from failures.

A Prompts

The prompts are presented in our code repository.

B Data

B.1 Benchmarks

SWE-Bench Verified Mini A subset of 50 human-validated real-world software engineering tasks from popular Python repositories. Each provides a GitHub issue and repository snapshot; agents produce patches that are evaluated against hidden unit tests (Jimenez et al., 2024).

AppWorld A benchmark for evaluating user-assistance agents on realistic day-to-day digital tasks. The agent interacts with the environment by writing Python code that is executed in a dedicated interpreter with access to the AppWorld APIs (Trivedi et al., 2024).

τ^2-Bench evaluates customer-service agents across retail, airline, and telecom domains via LLM-simulated users, measuring both policy-compliant task completion and violation rejection (Barres et al., 2025).

GAIA comprises 466 human-designed, real-world questions for evaluating general AI assistants. Each task requires fundamental abilities such as web browsing, multi-modal, and multi-file handling (Mialon et al., 2023).

B.2 Traces Data

HAL A unified evaluation framework that standardizes agent benchmarking across diverse domains. It provides a large set of execution traces, enabling automated trace evaluation to uncover hidden failure modes, issues in agent behavior, and unsafe real-world actions (Kapoor et al., 2026).

TRAIL provides a set of execution traces with human-annotated agent errors, based on a predefined taxonomy, testing whether LLM judges can accurately pinpoint reasoning, planning, and system execution failures (Deshpande et al., 2025).

B.3 Agents

CUGA **C**onfig**U**rable **G**eneralist **A**gent (CUGA) is an open-source system specifically designed for enterprise automation. It handles complex tasks through multi-agent orchestration, dynamic reasoning, and API integrations while ensuring strict policy compliance.

GAIA	SWE-bench Verified Mini
Similar Issues	
Inefficient workflow that delays or neglects high-signal resources (e.g., provided local files/attachments), causing redundant searches and retries	Employs inefficient, noisy workflows with unnecessary detours and retries
Violates tool constraints or misuses provided tools (e.g., disallowed `open`/`import`/`apt-get`)	Misuses tools or output formatting (disallowed imports, use of `open`/`subprocess`, incorrect code fences)
Failure to adhere to output formatting and deliverable specifications (e.g., missing required terminators, wrong format)	Does not follow task instructions or required outputs (e.g., missing facts survey, plan, or final patch)
Inadequate edge-case handling in computations (e.g., zero derivative, convergence and rounding rules)	Delivers partial fixes that miss edge cases or cross-backend/platform differences
Benchmark-Specific Issues	
Failure to use and verify the mandated authoritative source and its exact version/timeframe; reliance on mirrors/snippets	Produces incorrect or non-applicable patch output (escaped content, partial diffs, missing unified diff)
Insufficient cross-validation and evidentiary support; claims presented without corroboration	Runs commands/tests without ensuring environment prerequisites and paths are correct
Unreliable data processing: fragile parsing and incorrect filtering logic	Introduces broad behavior changes without proper scoping or compatibility/regression analysis
Wrong methodological framework or inconsistent formalism for the task	Avoids established APIs/patterns and relies on fragile techniques like monkey-patching
Incomplete enumeration or coverage before counting (missing items/pages; partial lists)	Insufficient validation of changes (lacks repo tests/regression, skips context-specific checks)
Poor disambiguation of task terms or scope, leading to misinterpretation of requirements	Provides no comments or documentation explaining rationale and potential impacts

Table 4: System-level issues generated by Agentic CLEAR for two benchmarks using the same agent (HAL Generalist), model (Claude 4.5 Sonnet), and judge (GPT-5). Top: shared issues surfaced for both benchmarks. Bottom: benchmark-specific issues.

HAL Generalist Agent An agent developed by the HAL team, designed to work across their unified evaluation framework.

HF Open Deep Research Agent An open-source agentic search framework developed by Hugging Face. It is engineered to autonomously navigate the web, synthesize information across long trajectories, and generate comprehensive, citation-backed answers for complex research queries.

C Issue Examples

We present two examples illustrating the issues generated by Agentic CLEAR across different configurations and analysis levels.

Table 4 presents the top 10 system-level issues discovered for both GAIA and SWE-Bench Verified Mini under the same configuration (same agent, model, and judge). Four of the issues are shared across the benchmarks, capturing universal error patterns such as inefficient workflows or tool misuse. The remaining issues are domain-specific: GAIA surfaces issues like inadequate source verification or unreliable data processing, while SWE-Bench Verified Mini reveals engineering-oriented failures such as broken patch output or missing regression tests. This differentiation occurred without any benchmark-specific prompting, demonstrating Agentic CLEAR's ability to adapt issue discovery to the relevant data.

Table 5 presents the top 10 issues discovered at the system level and at the node level for the TaskDecompositionAgent, both generated from the same CUGA traces on AppWorld. The system-level issues capture system-wide failure modes such as incomplete task execution or entity resolution errors. The node-level issues pinpoint planning-stage errors, including wrong app assignments or unsupported capability assumptions. Several themes appear at both levels but with different granularity. For example, the system level flags incomplete execution, while the node level traces it to the TaskDecompositionAgent omitting the finalization step. Together, the two views offer complementary diagnostics: the system level surfaces

System-Level Issues	TaskDecompositionAgent (Node-Level)
Execution flow management and processing strategy flaws: inefficient execution and incomplete coverage, loss of critical variables across steps, and continuing after success	Assumes unsupported app capabilities; violates strict app constraints
Validation and preconditions gaps: skips critical pre/post-action checks, neglects idempotency, fails to pre-check required permissions/tokens, and claims success without evidence	Workflow coherence errors: reasoning, tasks, and ordering don't align; steps lack clear dependencies or references to prior outputs
Blockage handling failures: does not prompt for missing info, avoids practical fallbacks or alternative paths, and declares failure prematurely	Fails to return the user's intent verbatim for single-app tasks; paraphrases or alters details
Incomplete execution: stops short of finishing the core task (checkout, send/forward, save/receipt)	App boundary and handoff mistakes: wrong app assignment, unclear division of responsibilities, or cross-app access without explicit handoff
Entity resolution weaknesses: brittle matching/normalization and poor disambiguation among multiple candidates	Insufficient disambiguation criteria for selecting among multiple emails/items or handling time zones
API/framework contract noncompliance: schema/role/output violations, misdeclared capabilities, misuse of fields/params, or using interactive prompts in non-interactive environments	Poor handling of absent data and capability limits/out-of-scope cases: lacks fallbacks, conditional logic, or alternative suggestions
Intent and channel selection errors: misinterprets the request, uses the wrong app/medium, or ignores medium-specific limits	Missing required parameters or constraints (time, recipients, recurrence, labels, totals, etc.)
Shopping cart and selection integrity issues: proceeds with contaminated carts, mishandles cart restoration or item-to-product mapping, and fails to validate product variants	Insufficient handling of edge cases and input/format variations (e.g., boundary dates, file formats, catalog matching)
Edge-case robustness deficiencies: mishandles time zones, date boundaries/rollovers, unit normalization, and variant-level inventory	Missing finalization: fails to perform the final action (e.g., send/forward, purchase) or to return/deliver the final answer to the user
Data integrity in inputs: uses hard-coded or unverified identifiers/values instead of extracting them from retrieved data	Adds unsupported assumptions or extra details not provided by the user

Table 5: Top 10 system-level and node-level issues generated by Agentic CLEAR for the CUGA agent on AppWorld (GPT-4o backbone, GPT-5 judge), sorted by frequency. System-level analysis captures system-wide failure modes, while node-level analysis pinpoints component-specific root causes within the TaskDecompositionAgent.

broad failure patterns, while the node level localizes problems to specific components and uncovers nuanced failures not visible at the system level.

D Agentic CLEAR Issues to TRAIL Mapping

Tables 6 and 7 present the full mapping between the issues generated by Agentic CLEAR at the system-level using GPT-5 and OSS-120B, respectively, and the TRAIL taxonomy.

E Rubric Analysis

To better understand what our generated rubrics capture, we compared our generated rubrics against benchmark-native evaluation criteria on sampled tasks. Notably, our rubrics are generated from the task description alone, without access to benchmark-internal metadata. This gap affects benchmarks differently. In AppWorld, our rubrics correctly capture the expected agent behavior, but describe the process qualitatively, while gold assertions are programmatic state checks against precomputed outcomes. For example, on a shopping task, our rubrics capture the workflow (retrieve list, parse items, add to cart, checkout), while gold constraints focus on assertions that validate the gold state is reached, like "exactly one new order created" and "no address records modified. In τ^2-Bench, the difference is much starker. Many tasks are adversarial, meaning the correct behavior is to refuse the user's request. Because our generator sees only the surface-level request, it inadvertently produces rubrics that reward task completion instead. These findings suggest that rubric generation from task descriptions alone can be enhanced by task metadata and should be examined based on the target agentic setting.

F Related Work

General Agent Evaluation Our work takes a first step towards automatic environment-agnostic agent evaluation. Recent work has begun to understand the importance of standardizing agentic evaluation (Bandel et al., 2026b,c) and has made first steps towards achieving it (Kapoor et al., 2026). These efforts focus on the runtime and execution layers across environment type (Bandel et al., 2026a; Harbor Framework Team, 2026), standardizing agent evaluation protocols (Lacoste et al., 2026), and building frameworks that enable easy and scalable agent and benchmark integration. While these efforts focus on standardizing the benchmarking infrastructure, Agentic CLEAR operates above the execution layer and addresses how to interpret traces, providing out-of-the-box multi-level agent evaluation.

Agent Meta-Evaluation Several recent works focus on creating benchmarks that assess judges' ability to detect agent erroneous steps and classify them into the right pre-defined category (Cemri et al., 2026; Zhu et al., 2026; Deshpande et al., 2025; Lù et al., 2025). These works extend a large body of works on meta-evaluation of LLMs (Zheng et al., 2023; Gera et al., 2025). Unlike these approaches, which assume a fixed error taxonomy and evaluate judges' recovery of it, Agentic CLEAR operates without predefined categories, dynamically surfacing failure patterns that adapt to the target system and domain.

#	Issue	Full Match	Partial Match
1	Did not leverage available tools and site-specific features; relied on memory or generic methods instead of invoking search/visit/inspect	*Tool Selection Errors*	*Hallucinations – Lang.* *Poor Info. Retrieval*
2	Repeated tool misuse without correction (e.g., wrong arguments to functions, failure to use helper functions or archives)	*Tool Selection Errors*	*Hallucinations – Tool* *Context Handling Failures* *Resource Abuse*
3	Made claims without case-specific substantiation: guessed/fabricated data, provided generic explanations, or asserted verification without reproducible evidence (quotes, permalinks, screenshots, tool outputs, URLs/-parameters)	*Hallucinations – Lang.*	*Hallucinations – Tool* *Poor Info. Retrieval*
4	Failed to follow the outlined plan; skipped core retrieval and verification steps	*Task Orchestration* *Goal Deviation*	*Instruction Non-compl.*
5	Incomplete execution and coverage of the task: did not enumerate all candidates, apply filters, check all occurrences, or perform required computations/counts/-filtering	*Task Orchestration* *Goal Deviation*	*Instruction Non-compl.*
6	Poor error recovery; repeated failing steps instead of pivoting to alternative strategies	*Task Orchestration*	*Resource Abuse* *Context Handling Failures*
7	Inefficient and redundant actions; excessive planning without tangible progress	*Resource Abuse*	*Task Orchestration*
8	Faulty or superficial parsing/extraction of source content (PDF/HTML or narratives), leading to incorrect values or miscounts	*Tool Output Misinterp.*	*Poor Info. Retrieval* *Formatting Errors*
9	Internal inconsistencies or logic errors within the analysis (e.g., off-by-one indices, contradictory counts, changing the required metric mid-analysis)	NA	*Goal Deviation* *Incorrect Problem Id.* *Context Handling Failures*
10	Did not ensure requirements and scope clarity: failed to state/apply definitions, timeframe/version constraints, or to ask for clarification/pause when inputs were unavailable	*Incorrect Problem Id.*	*Instruction Non-compl.*
11	Output formatting and numerical precision requirements not followed (units, rounding rules, extra words, template violations)	*Formatting Errors*	*Instruction Non-compl.*
12	Minor factual inaccuracies and inconsistencies (e.g., misspelled names, wrong coordinates, misread code/-grammar)	*Hallucinations – Lang.*	*Tool Output Misinterp.*
13	Ignored explicit source requirements; did not consult the specified source	*Instruction Non-compl.*	*Poor Info. Retrieval*
14	Failed to verify data adjustment/measurement methodology (e.g., adjusted vs. unadjusted, intraday vs. close)	*Tool Output Misinterp.*	*Incorrect Problem Id.* *Poor Info. Retrieval*
15	Lack of cross-verification across independent sources	*Poor Info. Retrieval*	*Hallucinations – Lang.*

Table 6: Mapping of GPT-5 system-level issues (GAIA) to TRAIL error categories. *Lang.* = Language-only; *Tool* = Tool-related; *Misinterp.* = Misinterpretation; *Id.* = Identification; *Non-compl.* = Non-compliance; *Info.* = Information. "—" indicates no full match.

#	Issue	Full Match	Partial Match
1	Answers generated without retrieving or verifying data using tools, leading to unverified or fabricated information	*Hallucinations – Lang.*	*Poor Info. Retrieval* *Tool Selection Errors*
2	Repeated identical or unnecessary tool calls	*Resource Abuse*	*Task Orchestration*
3	Poor error handling and lack of adaptation to failures	*Task Orchestration*	*Context Handling Failures* *Resource Abuse*
4	No source citations or evidence provided for answers	*Instruction Non-compl.*	*Hallucinations – Lang.*
5	Misused tool arguments or invoked the wrong tool	*Tool Selection Errors*	*Hallucinations – Tool*
6	Excessive planning steps that cause inefficiency	*Resource Abuse*	*Task Orchestration*
7	Missing required formatting tags or improper final answer handling	*Formatting Errors*	*Instruction Non-compl.*
8	Factual inaccuracies caused by unsupported assumptions	*Hallucinations – Lang.*	*Poor Info. Retrieval*
9	Failed to handle missing or unsupported files gracefully	*Task Orchestration*	*Tool Output Misinterp.* *Context Handling Failures*
10	Did not use the most appropriate specialized API (e.g., Wikipedia API) for precise data extraction	*Tool Selection Errors*	*Poor Info. Retrieval*
11	Inefficient or incorrect handling of pagination and multi-page navigation	*Poor Info. Retrieval*	*Context Handling Failures* *Tool Output Misinterp.* *Resource Abuse*
12	Inappropriate tool selection for the task (e.g., omitting a required web_search)	*Tool Selection Errors*	*Poor Info. Retrieval*

Table 7: Mapping of OSS-120B system-level issues (GAIA) to TRAIL error categories. Abbreviations as in Table 6.