

A Diagnostic Failure Taxonomy and Trajectory Critic for Data Agent Improvement

Suchen Liu¹, Yuanfeng Song^{2*}, Jun Gao^{1*}, Xing Chen²

¹Peking University, Beijing, China

²ByteDance, China

{2301111946, gaojun}@pku.edu.cn {songyuanfeng, chenxing.xc}@bytedance.com

Abstract

Large Language Models (LLMs) have shown strong potential as autonomous agents for complex data analysis. However, extending their capabilities to open-ended insight discovery remains challenging, as failures inevitably arise during multi-step reasoning and often cascade into severe downstream failures. Existing generic self-correction methods struggle to address these issues due to a lack of structural understanding of analytical failures. To tackle this, we propose the **Taxonomy-Guided Trajectory Critic (TGT-Critic)**, a novel runtime correction framework that detects and mitigates failures within the agent’s execution trajectory. To ground our critic with domain-specific diagnostic capabilities, we first construct a fine-grained Agent Failure Taxonomy and an annotated dataset (Insight-Bench) to systematically map the root causes of analytical failures across six dimensions. By formalizing failure diagnosis as a taxonomy-grounded inference problem, TGT-Critic dynamically provides targeted revisions for both high-level planning and low-level action execution. Experimental results demonstrate that our framework effectively arrests cascading failures, significantly outperforming generic baselines and yielding a 10.52% relative gain in the quality of generated insights.

1 Introduction

Large language model (LLM) agents are a powerful paradigm for automating complex workflows. In data analysis and data science, these agents combine natural language reasoning with tool execution to manipulate datasets, generate code, and synthesize analytical insights. Systems like ReAct (Yao et al., 2022) and Plan-and-Solve (Wang et al., 2023a) Prompting demonstrate early forms of reasoning, acting, and structured planning. Building on these ideas, recent agent systems adopt hierarchical Planner-Actor architectures, where a planner decomposes tasks and an executor interacts with tools over long horizons (Shi et al., 2025).

Despite advances, autonomous data analysis remains highly failure-prone. Open-ended insight discovery demands logical consistency in goal planning, data grounding, method selection, and result interpretation. A failure at any early stage often cascades through the agent’s trajectory, leading to syntactically correct but analytically flawed conclusions. While recent efforts have explored self-reflection and critic-based mechanisms to improve LLM reliability, such as Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023), most existing critics rely on generic prompts (e.g., “check for failures”). They lack a structural understanding of *why* and *how* data analysis agents fail, making them ineffective at pinpointing the root causes of complex, multi-step analytical breakdowns.

To address this critical gap, we propose the **Taxonomy-Guided Trajectory Critic (TGT-Critic)**, a runtime detection and correction framework specifically designed for data analysis agents. Unlike generic self-correction modules, TGT-Critic formalizes failure diagnosis as a taxonomy-grounded inference problem. It evaluates both the agent’s high-level plans and

*Corresponding authors.

low-level actions at runtime, mapping observed anomalies to specific predefined failure modes, and subsequently injecting targeted corrective suggestions back into the agent’s reasoning loop for autonomous correction.

Crucially, the effectiveness of TGT-Critic depends on a rigorous diagnostic foundation. To enable this, we conduct a systematic empirical study and establish a **comprehensive Data Analysis Agent Failure Taxonomy** for data analysis agents. We construct a failure-annotated corpus of agent trajectories based on InsightBench, and identify failure patterns across six operational dimensions: *Cognitive, Data, Method, Execution, Insight, and Communication*. Our empirical analysis shows that failures in final insight generation are rarely isolated, but instead emerge as downstream consequences of earlier cognitive planning failures and data schema misunderstandings. This finding highlights that failures in data analysis agents are inherently propagative and structured, which directly motivates the design of a taxonomy-guided critic that performs hierarchical diagnosis and early-stage intervention. We provide the complete taxonomy definitions and correction strategies in Appendix A.

In summary, our contributions are summarized as follows:

- We present the first systematic **Agent Failure Taxonomy** for data analysis, grounded in large-scale empirical analysis of agent trajectories, which reveals the structured and propagative nature of failures across multiple reasoning stages.
- We propose **TGT-Critic**, a taxonomy-guided runtime critic that performs structured failure diagnosis and targeted correction by mapping trajectory-level anomalies to predefined failure modes, enabling early intervention before failures propagate to downstream stages.
- Through extensive experiments on complex data analysis tasks, we demonstrate that taxonomy-guided correction significantly improves robustness in complex analytical workflows. By integrating TGT-Critic, we achieve a 10.52% relative improvement in task success over standard baseline agents.

2 Related Work

2.1 LLM Agents for Data Analysis

Recent advances in LLM-based agents have enabled systems to perform complex multi-step reasoning with tool use and environment interaction. Foundational paradigms such as ReAct (Yao et al., 2022) and Toolformer (Schick et al., 2023) establish the integration of reasoning and acting. Building on these ideas, recent works explore increasingly capable agent architectures with planning, memory, self-reflection, collaborative reasoning, and long-horizon memory mechanisms (Yehudai et al., 2025; Mohammadi et al., 2025; Liu et al., 2026b;a; Liang et al., 2026).

In the context of data analysis, LLM agents are evolving toward end-to-end analytical pipelines that combine code generation, data manipulation, and insight synthesis. Recent benchmarks and systems such as DABstep (Egg et al., 2025) and InsightBench (Sahu et al., 2025) explicitly evaluate multi-step data analysis workflows, highlighting the challenges of iterative reasoning, tool usage, and cross-step dependency handling. These works demonstrate that even state-of-the-art agents struggle with complex analytical tasks, achieving low success rates in realistic multi-step scenarios.

Despite these advances, existing work primarily focuses on improving agent capability and task success rates, while treating failures as opaque outcomes. There is limited understanding of *how* and *where* failures arise within data analysis workflows, particularly in insight generation stages.

2.2 Benchmarks for Data Analysis Agents

A growing body of work focuses on benchmarking LLM agents in increasingly realistic environments. Early benchmarks such as Spider (Yu et al., 2018) and TabFact (Chen et al.,

2019) evaluate structured reasoning over tabular data, but are limited to single-step or well-defined tasks. More recent agent benchmarks, including AgentBench (Liu et al., 2024) and AgentBoard (Ma et al., 2024), evaluate agents across interactive environments and multi-step decision-making settings.

Recent work further pushes toward realistic and dynamic evaluation settings. For example, GAIA (Mialon et al., 2023) and its recent extension Gaia2 (Froger et al., 2026) introduce dynamic, asynchronous environments where agents must operate under uncertainty and evolving contexts. Similarly, NatureGAIA (Zheng et al., 2025) propose trajectory-centric benchmarks that emphasize long-horizon reasoning and reproducibility. These works reflect a broader shift from static benchmarks to more realistic, process-oriented evaluation settings (Mohammadi et al., 2025).

In the specific domain of data analysis, IDA-Bench (Li et al., 2025) and DABstep (Egg et al., 2025) represent early efforts to evaluate iterative analytical workflows. More recently, benchmark efforts have expanded toward broader data settings, including structured and unstructured analytics (Weng et al., 2026). InsightBench (Sahu et al., 2025) further elevated the task complexity: unlike previous benchmarks that focus on answering single, well-defined queries, InsightBench evaluates agents on their ability to perform end-to-end data analytics.

2.3 Failure Analysis and Debugging of LLM Agents

Understanding and mitigating failures in LLMs has attracted increasing attention. Prior work has identified common failure modes such as hallucination (Ji et al., 2023), and tool misuse in agent settings. Techniques such as chain-of-thought prompting (Wei et al., 2022), self-consistency (Wang et al., 2023b), and reflection-based methods like Reflexion (Shinn et al., 2023) aim to improve reasoning reliability through iterative refinement.

More recent works explore self-debugging and verification mechanisms in agent systems. For instance, self-debugging approaches allow models to iteratively detect and correct failures in generated code or reasoning steps (Chen et al., 2023). Verification frameworks and critic models have also been proposed to assess reasoning correctness or enforce consistency during generation. Additionally, recent works highlight the need for systematic evaluation frameworks that capture agent reliability, robustness, and safety (Yehudai et al., 2025; Mohammadi et al., 2025).

3 Empirical Foundation: The Failure Taxonomy

To build a reliable critic capable of diagnosing and mitigating multi-step analytical failures, we must first systematically understand the underlying *generative mechanisms* of these failures. Existing generic self-correction methods fail precisely because they lack this structural knowledge. In this section, we establish the empirical foundation for our proposed TGT-Critic by constructing a comprehensive Agent Failure Taxonomy, annotating a corpus of execution trajectories, and analyzing how failures propagate.

3.1 Categorization Framework as a Knowledge Base

A central design principle of our taxonomy is to decompose agent trajectories into a sequence of functional decision stages. Rather than treating failures as surface-level artifacts, this decomposition attributes failures to specific decision points, providing a structured knowledge base $\mathcal{K}$ for our critic module. We classify trajectory failures into six primary categories, as mentioned in Fig. 1:

- **Cognitive Failures:** Arise from failures in goal understanding, planning, and problem formulation at the earliest stage. These include misinterpretation of user intent and flawed task decomposition.



Figure 1: The Multi-Dimensional Data Analysis Agent Failure Taxonomy: A systematic categorization framework illustrating the generative mechanisms of analytical breakdowns across Cognitive, Data, Method, Execution, Insight, and Communication dimensions.

- **Data Failures:** Encompass issues during data acquisition, cleaning, and transformation. Misinterpretation of data schemas (e.g., misunderstanding column semantics) is a critical failure mode in this category.
- **Method Failures:** Occur when inappropriate analytical models or statistical techniques are selected, such as violations of statistical assumptions or flawed comparative analysis setups.
- **Execution Failures:** Involve breakdowns during code execution, tool invocation, or rendering outputs (e.g., runtime exceptions or missing dependencies).
- **Insight Failures:** Arise when the agent produces conclusions that are superficial, unsupported by the evidence, or hallucinatory.
- **Communication Failures:** Refer to deficiencies in how analytical results are structured and conveyed, such as biased narratives or unclear visualizations.

3.2 Corpus Construction and Quality Control

To operationalize this taxonomy, we construct a failure-annotated corpus of 100 agent trajectories collected from complex, multi-step data analysis tasks on InsightBench (Sahu et al., 2025). Each trajectory contains the full sequence of plans, tool calls, intermediate results, and final reports.

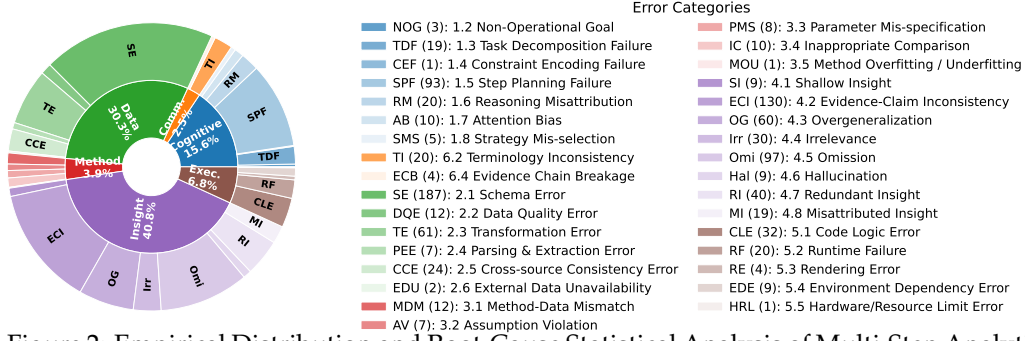


Figure 2: Empirical Distribution and Root-Cause Statistical Analysis of Multi-Step Analytical Failures Extracted from Agent Execution Trajectories.

Table 1: Top 10 Most Frequent Fine-grained (L3) Failure Types

L3 Category	2.1.1	5.5.1	5.2.1	5.2.3	1.5.3	5.7.1	5.3.1	4.1.1	2.3.3	2.3.2
Frequency	179	91	67	51	51	40	35	30	28	27

We map observed failures to our taxonomy using a robust dual-model independent annotation strategy. Specifically, we use two internal agents to independently annotate each trajectory. Their outputs are then cross-validated against each other, and finally are validated by expert human reviewers to filter out hallucinations and ensure consistency. Besides, to prevent the over-representation of symptomatic failures, our guidelines instruct annotators to identify the root failure node and prune subsequent cascading failures along that specific flawed reasoning branch.

3.3 Empirical Analysis of Failure Patterns

With the annotated corpus, we analyze failure distributions and failure propagation to explicitly motivate the design of our TGT-Critic.

3.3.1 Overall Distribution and Single-Failure Impact

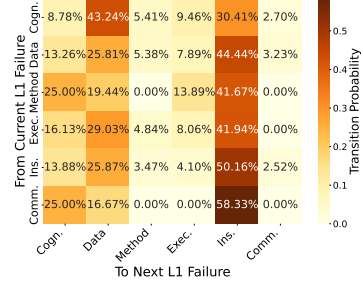
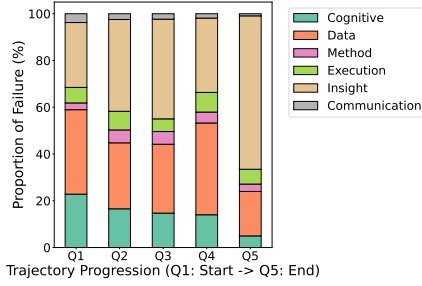
Our high-level (L1) analysis reveals that failures are predominantly concentrated in the **Category 5 Insight** phase (40.8%, $N = 394$) and the **Category 2 Data** phase (30.3%, $N = 293$). At a finer granularity, the single most common L3 failure is **2.1.1 Field misinterpretation** ($N = 179$), highlighting agents’ pervasive difficulty in understanding complex tabular schemas prior to generating insights.

To quantify the marginal impact of each failure category on the final task score, we compute Cliff’s Delta (δ_c). Our results show that **5.1 Shallow Insight** and **5.4 Irrelevance** exhibit the largest negative effect sizes ($p < 0.05$). Specifically, L3 failures such as **5.4.1 Deviates from main goal** and **5.1.1 Only describes patterns, lacks reasoning/prediction** severely degrade model performance. This confirms that while agents may successfully execute code, their inability to generate deep, relevant insights remains the primary bottleneck.

3.3.2 Failure Co-occurrence and Compounding Effects

In real-world agent trajectories, failures rarely occur in a vacuum. To understand these dependencies, we calculate the Jaccard Similarity Index for co-occurring failures.

At the L2 level, we observe an exceptionally high Jaccard index (0.75) between **2.1 Schema Failure** and **5.2 Evidence-Claim Inconsistency**. This demonstrates a direct pipeline effect: misinterpreting the underlying data schema almost inevitably forces the agent to generate insights that contradict the actual evidence. Furthermore, synergy analysis reveals that the combination of early-stage failures (e.g., **1.3 Task Decomposition Failure** combined with **2.1 Schema Failure**) exponentially amplifies the negative impact on the final outcome. This tight coupling suggests that fixing isolated reasoning steps is insufficient; intervention must target the root causes (Data/Planning) early in the trajectory.



(a) Temporal Dynamics and Transition Matrix of High-Level Failure Categories

(b) Modeling Failure Dependencies: Temporal Evolution and Transition State Matrix

Figure 3: Temporal evolution and cross-category transition dynamics of agent failures.

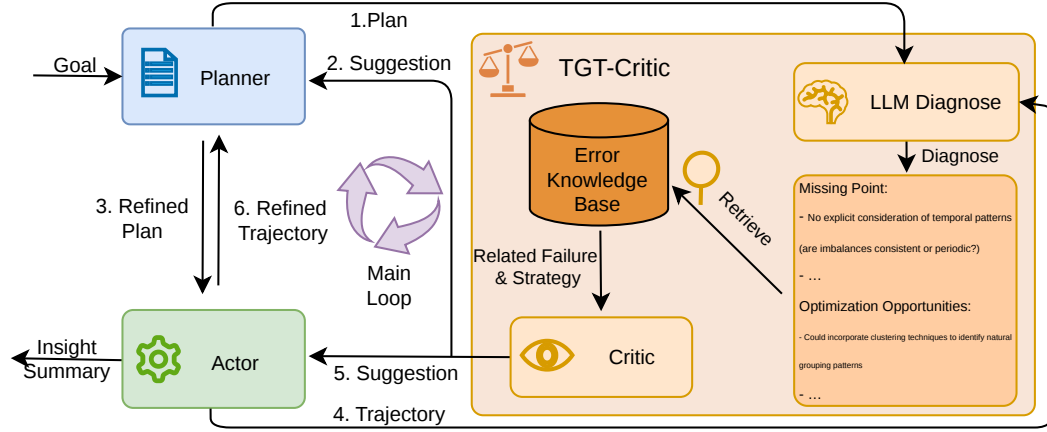


Figure 4: Architectural Overview of the Taxonomy-Guided Trajectory Critic (TGT-Critic): A runtime detection and mitigation framework that integrates taxonomy-grounded failure diagnosis into hierarchical Planner-Actor reasoning loops for early-stage intervention.

3.3.3 Cross-Category Cascades: Motivating Runtime Intervention

To capture the temporal dynamics of these compounding failures, we construct a transition matrix $M_{ij} = P(C_j | C_i)$ between high-level failure categories.

As shown in Figure 3b, failures originating from **Category 2 Data**, **Category 3 Method**, and **Category 4 Execution** overwhelmingly transition into **Category 5 Insight** failures (accounting for 44.44%, 41.67%, and 41.94% of their respective transitions). Additionally, **Category 1 Cognitive** failures frequently transition backward into **Category 2 Data** failures (43.24%), indicating that flawed planning often leads the agent to re-query or misinterpret data structures.

Takeaway for Method Design: These cascading transition patterns empirically demonstrate that Insight failures predominantly arise as downstream consequences of upstream breakdowns. Moreover, since the vast majority of failures ultimately propagate into Insight-level failures, mitigating them solely through post-hoc evaluation or end-of-task reflection is inherently insufficient. A more effective solution demands a *runtime critic* that can intercept and correct such cascades as they unfold. Critically, different failure categories also exhibit distinct characteristics, such as varying temporal dynamics, severity levels, and manifestation patterns, which further motivates the need for a systematic, stage-wise mechanism that handles variant failure types in different way along the trajectory.

4 Taxonomy-Guided Trajectory Critic (TGT-Critic)

Motivated by the cascading nature of analytical failures observed in Section 3, we propose the **Taxonomy-Guided Trajectory Critic (TGT-Critic)**. This framework transforms post-

hoc failure analysis into a proactive, runtime intervention mechanism. The key idea is to formalize failure diagnosis as a **taxonomy-grounded inference problem** and integrate it into both planning and action execution loops.

4.1 Formalizing the Hierarchical Execution Trajectory

We formalize the autonomous data analysis process as a hierarchical interactive system. The execution unfolds over a sequence of **macro-steps** k , each encompassing multiple **micro-steps** t . At each macro-step k , the **Planner** observes the global state S_k (e.g., current dataset schema, memory context) and the global historical trajectory $H_{<k}$. It then formulates a high-level sub-goal or methodological plan:

$$\rho_k \sim \pi_{\text{plan}}(\cdot \mid S_k, H_{<k}). \quad (1)$$

Conditioned on this overarching plan ρ_k , the **Actor** takes over to execute a multi-step loop. At each micro-step $t \in \{1, \dots, T_k\}$ within macro-step k , the Actor generates an executable action (e.g., generating and executing Python code) based on the current local state $s_{k,t}$ and the local execution history $h_{k,<t}$:

$$\alpha_{k,t} \sim \pi_{\text{act}}(\cdot \mid \rho_k, s_{k,t}, h_{k,<t}). \quad (2)$$

Upon executing $\alpha_{k,t}$, the environment yields an observation $o_{k,t}$ (e.g., stdout, statistical plots, or failure tracebacks). The Actor repeats this micro-step loop until ρ_k is fulfilled or terminated, after which control returns to the Planner for macro-step $k + 1$.

In traditional agent setups, the agent blindly accepts $o_{k,t}$ and proceeds, allowing latent failures embedded in either the plan ρ_k or the action $\alpha_{k,t}$ to cascade into subsequent steps.

4.2 Taxonomy-Grounded Diagnosis via Thought-Augmented Retrieval

To accurately map execution anomalies to our theoretical taxonomy without overwhelming the LLM context window, we design the Critic module $\mathcal{C}$ as a two-stage, thought-augmented retrieval pipeline. We formalize the taxonomy knowledge base as a searchable vector store $\mathcal{K} = \{(e_i, S_i, C_i)\}_{i=1}^N$, where each entry contains the operational failure definition e_i , trigger signals S_i , and specific correction strategies C_i . Given the latest execution context $T_{k,t} = (\text{Goal}, \text{Trajectory}, \text{Latest Output})$, the diagnostic process proceeds as follows:

Stage 1: Diagnostic Thought Generation. First, instead of querying the knowledge base directly with raw code or execution logs, we prompt an expert *Diagnostician LLM* to analyze $T_{k,t}$ and generate a dense diagnostic reasoning text $z_{k,t}$:

$$z_{k,t} = \text{LLM}_{\text{diag}}(\text{Analyze potential logical mistakes in } T_{k,t}). \quad (3)$$

Stage 2: Diagnosis-Guided Knowledge Retrieval. We then use the generated thought $z_{k,t}$ as the semantic query to retrieve the top- K most relevant failure templates from the knowledge base, filtered by the active functional module (e.g., Analytical Reasoning, Execution):

$$\mathcal{K}_{k,t}^* = \text{TopK}(\text{Sim}(z_{k,t}, \mathcal{K})). \quad (4)$$

Stage 3: Structured Critique Output. Finally, an *Failure Critic LLM* synthesizes the execution context $T_{k,t}$ and the retrieved taxonomy guidelines $\mathcal{K}_{k,t}^*$ to output a strictly formatted JSON decision object:

$$c_{k,t} = \text{LLM}_{\text{critic}}(T_{k,t}, \mathcal{K}_{k,t}^*). \quad (5)$$

The output $c_{k,t}$ is parsed into four components: $(y_{\text{err}}, y_{\text{replan}}, \text{expl}, \text{sugg})$, where $y_{\text{err}} \in \{0, 1\}$ indicates if a failure occurred, $y_{\text{replan}} \in \{0, 1\}$ dictates if a high-level plan revision is required, and the latter two provide human-readable explanations and actionable suggestions.

4.3 Two-Phase Intervention Mechanism

Unlike conventional reflective agents that evaluate the trajectory only after a full execution cycle, TGT-Critic operates as a **dual-phase intervention mechanism**. To precisely intercept failures before they cascade, the Critic module is invoked sequentially at two distinct temporal boundaries within a single iteration: **post-planning** (Plan evaluation) and **post-execution** (Action evaluation). This two-phase intervention is formalized as follows:

Phase 1: Post-Planning Plan Refinement. At iteration k , the Planner first generates an initial step or sub-goal $\rho_k^{(0)} = \pi_{\text{plan}}(S_k, H_{<k})$. Immediately after this planning phase, but before the Actor is permitted to act on it, the *Planner Critic* synchronously evaluates the logical soundness of the proposal:

$$c_k^{\text{plan}} = \mathcal{C}(\text{Goal}, H_{<k}, \rho_k^{(0)}). \quad (6)$$

If the critic detects a latent Category 1 (Cognitive) or Category 3 (Method) failure, it outputs a rejection signal ($y_{\text{replan}} = 1$) along with targeted suggestions. The framework suspends the transition, feeds the critique back to the Planner, and forces a regeneration of the step:

$$\rho_k^* = \pi_{\text{plan}}(\rho_k^{(0)}, \text{sugg}^{\text{plan}}). \quad (7)$$

This post-planning intervention ensures that the Actor only operates on methodologically valid constraints, preventing structurally flawed plans from wasting downstream computational resources.

Phase 2: Post-Execution Action Revision. Once a valid plan ρ_k^* is secured, the Actor executes the necessary tools or generated code, yielding an initial observation $o_k^{(0)} = \mathcal{E}(\pi_{\text{act}}(\rho_k^*, H_{<k}))$. Subsequently, the *Actor Critic* assesses the execution result:

$$c_k^{\text{act}} = \mathcal{C}(\text{Goal}, H_{<k}, o_k^{(0)}). \quad (8)$$

On execution failure, the critic rolls back locally. Crucially, instead of discarding the overarching plan ρ_k^* , the framework temporarily injects the critic’s suggestion (sugg^{act}) into the immediate execution history. The Actor is then re-invoked to rectify its output:

$$o_k^* = \mathcal{E}(\pi_{\text{act}}(\rho_k^*, H_{<k} \cup \{o_k^{(0)}, \text{sugg}^{\text{act}}\})). \quad (9)$$

After the Actor successfully recovers and generates o_k^* , the temporary critique context is popped from the global history to prevent prompt pollution. Finally, the extracted insights are appended to the global state.

By structurally decoupling post-planning evaluation from post-execution evaluation within the same iteration loop, TGT-Critic can intercept both high-level hallucinations and low-level execution bugs exactly at the moment they materialize.

5 Experimental Evaluations

Experimental Setup. We evaluate TGT-Critic on two open-ended data analysis benchmarks, **InsightBench** (Sahu et al., 2025) and **InsightEval** (Zhu et al., 2025). Both benchmarks require agents to inspect tabular data, perform multi-step analytical reasoning, execute data transformations, and synthesize a final written report rather than returning a single short answer. We preserve the original Easy, Medium, and Hard task partitions in each benchmark, which allows us to examine whether runtime failure diagnosis remains effective as the required trajectory becomes longer and more methodologically demanding. For each dataset, we report split-level scores across all evaluated tasks.

Compared Methods. Our experiments include two types of comparisons. First, we compare against representative data-analysis and tool-using agent baselines: **ReAct** (Yao et al., 2022), **Pandas Agent** (LangChain, 2024), **AgentPoirot** (Sahu et al., 2025), and **Aime** (Shi et al., 2025). These baselines cover general reasoning-action interleaving, tabular-data manipulation, insight-oriented analytical agents, and hierarchical Planner–Actor designs. Second, to

Table 2: Performance comparison on InsightBench and InsightEval.

Model	InsightBench				InsightEval			
	Easy	Medium	Hard	Avg	Easy	Medium	Hard	Avg
ReAct (Yao et al., 2022)	0.5644	0.5057	0.4936	0.5194	0.5742	0.5574	0.5501	0.5600
Pandas Agent (LangChain, 2024)	0.5534	0.5343	0.4278	0.5059	0.5324	0.5340	0.5420	0.5361
AgentPoirot (Sahu et al., 2025)	0.4740	0.4476	0.3584	0.4270	0.5762	0.5560	0.5234	0.5517
Aime (Shi et al., 2025)	0.5942	0.5269	0.5100	0.5417	0.6527	0.6142	0.6140	0.6257
Baseline	0.5727	0.5864	0.5219	0.5616	0.5382	0.5822	0.5362	0.5543
Baseline (+ Generic Critic)	0.6213	0.6046	0.5553	0.5938	0.6341	0.6152	0.6089	0.6188
Baseline (+ TGT-Critic (Actor))	0.6335	0.5978	0.5643	0.5970	0.6292	0.6245	0.6188	0.6240
Baseline (+ TGT-Critic (Planner))	0.6432	0.6121	0.5460	0.6002	0.6379	0.6324	0.6423	0.6372
Baseline (TGT-Critic)	0.6549	0.6293	0.5785	0.6207	0.6503	0.6682	0.6438	0.6550

isolate the effect of our taxonomy-guided critic from the effect of the underlying scaffold, we conduct a controlled comparison on the same **Planner-Actor** agent. **Baseline** denotes the plain Planner-Actor agent without runtime criticism. **Baseline (+ Generic Critic)** augments the same scaffold with a free-form critic prompt that checks for potential errors but does not use taxonomy-grounded retrieval. **Baseline (+ TGT-Critic (Actor))** applies taxonomy-guided criticism only after tool execution. **Baseline (+ TGT-Critic (Planner))** applies taxonomy-guided criticism only after planning. **Baseline (TGT-Critic)** is the full two-phase version that checks both planning and execution.

Model and Evaluation Configuration. Unless otherwise specified, all agent-side modules in the controlled variants use **DeepSeek-V3-250324**, including the planner, actor, diagnostician, and critic. We use **GPT-4o-2024-11-20** only as the LLM judge in the evaluation pipeline, not as a task-solving component. The sampling temperature is fixed at 0.2 for agent generation. For TGT-Critic, the number of retrieved taxonomy examples is set to $k = 3$ by default, based on the ablation in Table 3. The final score of method m on a split D_s is computed as the mean judge score over all tasks in that split:

$$\text{Score}(m, D_s) = \frac{1}{|D_s|} \sum_{i=1}^{|D_s|} J(r_i^m, x_i), \quad (10)$$

where x_i is the task instance, r_i^m is the final analytical report produced by method m , and $J(\cdot)$ is the Summary-level G-Eval judge following prior work (Sahu et al., 2025). Dataset-level averages are task-weighted averages across the corresponding difficulty splits.

Overall Performance. Table 2 shows that TGT-Critic consistently improves the underlying Planner-Actor agent and achieves the best average score on both datasets. On InsightBench, the full model improves the baseline average from 0.5616 to 0.6207, corresponding to an absolute gain of 0.0591 and a relative gain of 10.52%. The improvement is not concentrated in a single difficulty group: TGT-Critic raises Easy, Medium, and Hard scores by 0.0822, 0.0429, and 0.0566, respectively. On InsightEval, the improvement is even larger, increasing the average score from 0.5543 to 0.6550, with an absolute gain of 0.1007 and a relative gain of 18.17%. These results indicate that taxonomy-guided intervention is not merely fitting the distribution of one benchmark, but transfers to a second benchmark with different tasks and data distributions.

Effect of Taxonomy Grounding. A central question is whether the observed gain comes from adding any runtime critic or specifically from grounding the critic in the failure taxonomy. The controlled comparison against **Baseline (+ Generic Critic)** answers this question. On InsightBench, the generic critic improves the baseline from 0.5616 to 0.5938, but the full TGT-Critic further improves the score to 0.6207. On InsightEval, the generic critic reaches 0.6188, while TGT-Critic reaches 0.6550. Thus, generic self-correction is useful, but structured failure typing and retrieved correction cases provide additional gains of 0.0269 on InsightBench and 0.0362 on InsightEval. This supports our central hypothesis that open-ended data analysis agents benefit from critic feedback that is not only evaluative but also diagnostically grounded.

Planner-Side versus Actor-Side Intervention. The actor-only and planner-only variants further clarify where failures arise in long-horizon data analysis. Actor-side intervention

Table 3: Impact of the number of retrieved examples (k) on InsightBench.

Split	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 6$	$k = 10$
Easy	0.6213	0.6411	0.6429	0.6549	0.6509	0.6471
Medium	0.6046	0.6211	0.6384	0.6293	0.6214	0.6139
Hard	0.5553	0.5518	0.5604	0.5785	0.5777	0.5697
Avg	0.5938	0.6049	0.6148	0.6207	0.6163	0.6097

improves the baseline average to 0.5970 on InsightBench and 0.6240 on InsightEval, showing that many failures emerge during code execution, data transformation, and final insight synthesis. Planner-side intervention is slightly stronger overall, reaching 0.6002 on InsightBench and 0.6372 on InsightEval, especially on the Hard split of InsightEval where it obtains 0.6423. This pattern is consistent with our empirical failure analysis: early planning errors often propagate into downstream analytical mistakes. The full two-phase TGT-Critic achieves the best average on both datasets because it combines these complementary effects. Planner criticism prevents structurally invalid steps from being executed, while actor criticism repairs local execution and reporting errors that remain after planning.

Difficulty-Level Analysis. The gains are particularly informative across difficulty levels. On Easy tasks, TGT-Critic mainly corrects common schema, formatting, and shallow-insight failures, producing the largest relative gain on InsightBench. On Medium tasks, the critic improves multi-step decomposition and helps the agent maintain consistency between intermediate observations and final claims. On Hard tasks, the absolute gains remain substantial even though the trajectories are longer and the error surface is more complex. The Hard-split improvements suggest that TGT-Critic does not simply patch superficial mistakes; it also mitigates cascading failures where an early invalid assumption, omitted preprocessing step, or weak comparison can compromise the final report.

Impact of Top- k Retrieval. Table 3 analyzes the retrieval depth used by TGT-Critic. When $k = 0$, the model reduces to a non-retrieval critic and obtains an average score of 0.5938, matching the generic-critic setting. Adding one or two retrieved cases improves the average to 0.6049 and 0.6148, demonstrating that concrete failure-and-correction examples provide useful diagnostic context. The best overall performance is achieved at $k = 3$, with an average score of 0.6207. Increasing k to 6 or 10 slightly reduces performance, which suggests that excessive retrieved examples introduce marginally relevant or redundant context. In open-ended analysis, this additional context can dilute the critic’s focus and make the correction less targeted. Therefore, our default configuration uses $k = 3$ as a balance between diagnostic coverage and prompt economy.

Summary of Findings. The experiments lead to three main findings. First, TGT-Critic consistently improves a strong Planner–Actor baseline across two data-analysis benchmarks. Second, taxonomy grounding yields additional gains beyond a generic runtime critic, confirming the value of structured failure diagnosis. Third, both planning-stage and execution-stage interventions are necessary: the former prevents invalid analytical trajectories from forming, while the latter repairs local code, data, and insight-generation failures before they appear in the final report.

6 Conclusion

In this paper, we address the challenge of cascading failure in autonomous data analysis agents. Through a large-scale empirical analysis of execution trajectories, we construct a comprehensive Agent Failure Taxonomy, demonstrating that final analytical failures predominantly stem from early-stage failures. To tackle this, we introduced the Taxonomy-Guided Trajectory Critic (TGT-Critic), a runtime framework that formalizes failure diagnosis as a taxonomy-grounded inference problem. By employing a two-phase intervention mechanism, TGT-Critic systematically intercepts failures before they propagate. Extensive experiments validate that our approach yields a 10.52% relative improvement in task success across varying difficulty levels.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62272008.

Ethics Statement

We acknowledge the use of artificial intelligence tools in the preparation of this manuscript. Specifically, we utilize AI language models to assist with language refinement of the text. Furthermore, AI coding assistants are employed to assist in writing and debugging the codebase for our experiments.

We emphasize that all text and code are thoroughly reviewed, fact-checked, and edited by the human authors. The authors take full and sole responsibility for the final content, originality, and scientific integrity of this paper.

References

- Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyu Zhou, and William Yang Wang. Tabfact: A large-scale dataset for table-based fact verification. *arXiv preprint arXiv:1909.02164*, 2019.
- Xinyun Chen et al. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.
- Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, 114(3):494, 1993.
- Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. Dabstep: Data agent benchmark for multi-step reasoning. *arXiv preprint arXiv:2506.23719*, 2025.
- Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, et al. Gaia2: Benchmarking llm agents on dynamic and asynchronous environments. *arXiv preprint arXiv:2602.11964*, 2026.
- Paul Jaccard. Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. *Bull Soc Vaudoise Sci Nat*, 37:241–272, 1901.
- Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38, 2023.
- LangChain. Pandas dataframe. <https://python.langchain.com/v0.2/docs/integrations/toolkits/pandas/>, 2024. Accessed: 2025-08-04.
- Hanyu Li, Haoyu Liu, Tingyu Zhu, Tianyu Guo, Zeyu Zheng, Xiaotie Deng, and Michael I Jordan. Ida-bench: Evaluating llms on interactive guided data analysis. *arXiv preprint arXiv:2505.18223*, 2025.
- Weijie Liang, Yuanfeng Song, Xing Chen, Caleb Chen Cao, Sirui Han, and Yike Guo. Vizomem: A visual-textual memory framework for efficient long-horizon reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.
- Suchen Liu, Yuanfeng Song, Jun Gao, and Xing Chen. Dataseer: A manager-centric collaborative multi-agent framework with multi-branch reasoning for automated insight discovery. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026a.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

- Xiaochuan Liu, Yuanfeng Song, Xiaoming Yin, and Xing Chen. Datasage: Multi-agent collaboration for insight discovery with external knowledge retrieval, multi-role debating, and multi-path reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026b.
- Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.
- Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.
- Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. Evaluation and benchmarking of llm agents: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 6129–6139, 2025.
- Karl Pearson. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.
- Gaurav Sahu, Abhay Puri, Juan A. Rodriguez, Amirhossein Abaskohi, Mohammad Chegini, Alexandre Drouin, Perouz Taslakian, Valentina Zantedeschi, Alexandre Lacoste, David Vazquez, Nicolas Chapados, Christopher Pal, Sai Rajeswar, and Issam H. Laradji. Insightbench: Evaluating business analytics agents through multi-step insight generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=ZGqd0cbBvm>.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.
- Yexuan Shi, Mingyu Wang, Yunxiang Cao, Hongjie Lai, Junjian Lan, Xin Han, Yu Wang, Jie Geng, Zhenan Li, Zihao Xia, et al. Aime: Towards fully-autonomous multi-agent framework. *arXiv preprint arXiv:2507.11988*, 2025.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.
- Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 2609–2634, 2023a.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, et al. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023b.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

- Han Weng, Zhou Liu, Yuanfeng Song, Xiaoming Yin, Xing Chen, and Wentao Zhang. Unidatabench: Evaluating data analytics agents across structured and unstructured data. In *Proceedings of the 64rd Annual Meeting of the Association for Computational Linguistics*, 2026.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.
- Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*, 2025.
- Tao Yu, Rui Zhang, Kai Yang, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing. In *EMNLP*, 2018.
- Zihan Zheng, Tianle Cui, Chuwen Xie, Jiahui Zhang, Jiahui Pan, Lewei He, and Qianglong Chen. Naturegaia: Pushing the frontiers of gui agents with a challenging benchmark and high-quality trajectory dataset. *arXiv e-prints*, pp. arXiv-2508, 2025.
- Zhenghao Zhu, Yuanfeng Song, Xin Chen, Chengzhong Liu, Yakun Cui, Caleb Chen Cao, Sirui Han, and Yike Guo. Insighteval: An expert-curated benchmark for assessing insight discovery in llm-driven data agents. *arXiv preprint arXiv:2511.22884*, 2025.

A Failure Taxonomy and Correction Strategies

Based on the rationale in Section 3 above, our taxonomy classifies agent trajectory failures into the following six primary categories. Each category is further broken down into secondary and tertiary subcategories, accompanied by specific correction strategies to mitigate these failures.

A.1 Cognitive Failures

Cognitive failures refer to deviations that occur during the agent’s mental processes, including understanding, planning, and reasoning. These failures often lead to fundamental task failures from the very beginning.

A.1.1 Ambiguous Goal Definition

The agent fails to clearly understand the high-level goal provided by the user due to ambiguity or conflicts, preventing accurate task initiation.

- **Goal description too vague:** The user’s provided goal lacks sufficient information, preventing the agent from forming a clear, executable intent (e.g., asking to “analyze sales data” without specifying dimensions, timeframes, or objectives).
- **Conflicting objectives:** The user proposes multiple contradictory goals within a single request that cannot be simultaneously satisfied without further constraints (e.g., demanding both “maximum profit” and “lowest price”).

Correction Strategy: Clarify goal statements, define measurable objectives, consult task specifications.

A.1.2 Non-Operational Goal

The understood goal lacks measurable or executable attributes, making it impossible to effectively decompose and execute.

- **Goal cannot be measured or executed:** The objective is too abstract or subjective, lacking clear success criteria or execution paths (e.g., “improve user happiness” without quantifiable metrics like NPS or retention rates).

Correction Strategy: Break down high-level goals into specific, operationalizable tasks with success criteria.

A.1.3 Task Decomposition Failure

Failures occur when the agent breaks down a high-level goal into specific sub-tasks or steps.

- **Missing critical sub-tasks:** The decomposition process omits crucial steps necessary to achieve the final goal (e.g., failing to check server logs when analyzing a website traffic drop).
- **Priority inversion:** Incorrect judgment regarding the importance and urgency of sub-tasks, placing secondary tasks before core tasks, leading to resource misallocation or logical confusion.
- **Redundant tasks:** Generating unnecessary or duplicate sub-tasks that increase execution costs without contributing to the final objective.

Correction Strategy: Use stepwise planning checklists; verify all critical sub-tasks are included and correctly ordered.

A.1.4 Constraint Encoding Failure

The agent fails to correctly understand, encode, or apply explicit or implicit constraints provided by the user.

- **Output format mis-specified:** Failure to follow the user’s requirements for the output format (e.g., generating plain text when a table is requested).
- **Role mis-assignment:** Failure to adopt the specified persona or role (e.g., responding like a beginner when asked to act as a “senior data analyst”).
- **Explicit constraints missing:** Ignoring explicit limitations set by the user during execution (e.g., including post-2022 data when explicitly told not to).

Correction Strategy: Explicitly encode all constraints and verify against task requirements.

A.1.5 Step Planning Failure

Logical flaws in the sequence of execution steps formulated by the agent to complete the task.

- **Step order incorrect:** The planned execution steps are logically disordered, violating task dependencies (e.g., starting model training before data cleaning).
- **Steps omitted:** Necessary steps identified during the decomposition phase are left out of the final execution plan.
- **Redundant steps:** The execution plan includes repetitive or invalid steps, resulting in low execution efficiency.

Correction Strategy: Verify step sequence and completeness using predefined planning heuristics.

A.1.6 Reasoning Misattribution

The agent establishes incorrect associations during causal inference or logical reasoning.

- **Incorrect causal link:** Misjudging correlation as causation, or drawing flawed causal conclusions without sufficient evidence.
- **Wrong assumptions in reasoning:** The reasoning process is built on a series of invalid or unverified assumptions, causing the entire logical chain to collapse.

Correction Strategy: Cross-check causal reasoning and assumptions against task data and prior knowledge.

A.1.7 Attention Bias

The agent fails to rationally allocate attention to all relevant information during processing, leading to cognitive bias.

- **Focus on irrelevant info:** The agent overly focuses on details or noise unrelated to the core objective, ignoring the main issues.
- **Missing key observation:** Failure to notice critical information or data patterns essential for solving the problem, missing key clues.

Correction Strategy: Use attention checklists or importance weighting to prioritize relevant data.

A.1.8 Strategy Mis-selection

The agent selects inappropriate macro-strategies or methodologies for problem-solving or reasoning.

- **Wrong inference strategy:** Selecting an unsuitable reasoning framework based on the problem’s nature (e.g., using simple ReAct instead of Tree-of-Thoughts for tasks requiring deep exploration).
- **Improper tool/method chosen:** Selecting tools during the planning phase that are incapable of solving the problem or are highly inefficient.

Correction Strategy: Compare alternative strategies and select the one aligning with task objectives.

A.2 Data Failures

Data failures encompass issues arising during data acquisition, parsing, cleaning, transformation, and utilization, directly leading to unreliable analytical outcomes.

A.2.1 Schema Failure

Deviations in the agent’s understanding of data structures, field definitions, or table relationships.

- **Field misinterpretation:** Misunderstanding the business meaning of a field in a data table (e.g., interpreting `is_vip` as “important customer” instead of “paid member”).
- **Type mismatch:** Failing to correctly identify and process data types.
- **Table relationship misunderstanding:** Misunderstanding the relationship or join keys between tables during JOIN operations.

Correction Strategy: Verify data schema, field types, and relationships before processing.

A.2.2 Data Quality Failure

Inherent defects in the raw data that the agent fails to effectively handle.

- **Missing values:** Records containing blank or null values that affect statistical calculations.
- **Outliers:** Data containing extreme values that significantly deviate from other observations.
- **Duplicates:** Datasets containing identical records or duplicates on key fields.
- **Internal inconsistency:** Logical contradictions within a single record or data source.

Correction Strategy: Run data validation checks, impute missing values, and remove duplicates.

A.2.3 Transformation Failure

Computational or logical failures occurring during data processing, aggregation, or the derivation of new variables.

- **Aggregation failure:** Selecting incorrect aggregation functions or dimensions during GROUP BY operations.
- **Feature engineering failure:** Flawed calculation logic when constructing new features.
- **Time processing failure:** Errors when handling time-series data, such as improper timezone conversions.

Correction Strategy: Verify transformation logic and test on small sample datasets.

A.2.4 Parsing & Extraction Failure

Failures or failures when extracting information from unstructured or semi-structured data.

- **Extraction failure:** Failing to successfully extract required information entities from the target text.
- **Format failure:** Parsing failures on structured data (e.g., JSON, XML) due to corrupted source formatting.
- **Code output parsing failure:** The agent executes code but fails to correctly interpret its printed or returned results.

Correction Strategy: Check parsing scripts and validate extracted values.

A.2.5 Cross-source Consistency Failure

Contradictory information regarding the same entity across different data tables or sources.

- **Conflicting data across tables/sources:** Inconsistencies found when joining multiple data sources (e.g., status mismatches across user and order tables).

Correction Strategy: Compare overlapping data from multiple sources and reconcile inconsistencies.

A.2.6 External Data Unavailability

External data sources relied upon by the agent are inaccessible or out of service.

- **API or external dataset inaccessible:** Failure to retrieve data due to network issues, permission changes, or service deprecation.

Correction Strategy: Verify availability of all external data sources prior to task execution.

A.2.7 Feature Drift / Concept Drift

The statistical properties of data or their underlying business meanings change over time.

- **Statistical distribution shift:** The distribution of input data changes significantly over time.
- **Concept mismatch over time:** The definition of the target variable or its relationship with features changes over time.

Correction Strategy: Monitor feature statistics and retrain models as needed.

A.3 Method Failures

Even with correct data and clear objectives, the inappropriate application of analytical models, statistical methods, or algorithms can yield flawed conclusions.

A.3.1 Method–Data Mismatch

The selected analytical method is unsuitable for the current data type or data characteristics.

- **Method unsuitable for data type:** Applying methods designed for specific data types to incompatible ones.

Correction Strategy: Check method assumptions against data type and distribution.

A.3.2 Assumption Violation

The prerequisite assumptions relied upon by the chosen model or statistical method are not satisfied by the current data.

- **Model assumptions not satisfied:** Failing to verify or forcibly using models when statistical assumptions (e.g., normality, independence) are violated.

Correction Strategy: Validate assumptions before applying methods.

A.3.3 Parameter Mis-specification

Setting incorrect parameter values when invoking models or methods.

- **Wrong parameter values:** Assigning inappropriate hyperparameters or configuration parameters, causing the model to fail or perform poorly.

Correction Strategy: Use parameter validation, cross-validation, or sensitivity analysis.

A.3.4 Inappropriate Comparison

Selecting invalid baselines or using incorrect statistical inference methods during comparative analysis.

- **Invalid baseline:** Choosing an incomparable control group or baseline for comparison.
- **Improper statistical inference:** Using incorrect statistical testing methods when comparing groups, or misinterpreting p-values.

Correction Strategy: Select appropriate baseline and test methods.

A.3.5 Method Overfitting / Underfitting

The complexity of the model does not align with the underlying patterns of the data.

- **Model too specific or too general:** Overfitting noise and details, or failing to capture latent data patterns due to over-simplicity.

Correction Strategy: Evaluate model performance on validation datasets and adjust complexity.

A.4 Execution Failures

Execution failures involve technical breakdowns when the agent attempts to run code, invoke tools, or render outputs.

A.4.1 Code Logic Failure

The code written by the agent may be syntactically correct, but its internal logic fails.

- **Algorithm mistake:** Flaws in the core logic when implementing specific algorithms.
- **Loop or conditional failure:** Incorrectly set loop or conditional logic.

Correction Strategy: Code review, unit testing, and verification of algorithm correctness.

A.4.2 Runtime Failure

Code execution is interrupted due to exceptions.

- **Crash, infinite loop, memory overflow:** The program encounters unhandled failures and crashes, gets stuck, or requests excessive memory.

Correction Strategy: Monitor runtime logs, set timeouts, and handle exceptions.

A.4.3 Rendering Failure

Display issues occurring during the generation of final visual artifacts.

- **Visualization or output misrendered:** Generated chart elements overlap, labels are incomplete, or the final document layout is chaotic.

Correction Strategy: Validate visualizations against expected output formats.

A.4.4 Environment Dependency Failure

Issues with the environments, libraries, or external services required to execute the code.

- **Missing libraries:** The code attempts to import Python libraries that are not installed.
- **API unavailable:** External or internal API services are temporarily inaccessible.

Correction Strategy: Verify all dependencies and environments before execution.

A.4.5 Hardware/Resource Limit Failure

The computational resources required to execute the task exceed the allocated system limits.

- **Memory, CPU/GPU limitation:** Processing large-scale data consumes resources beyond the environment's upper limits.

Correction Strategy: Estimate resource requirements and allocate sufficient resources.

A.5 Insight Failures

Insight failures occur when the agent struggles to distill, interpret, and present conclusions from the analytical results.

A.5.1 Shallow Insight

Conclusions remain at a simple descriptive level of data phenomena.

- **Only describes patterns, lacks reasoning/prediction:** The report only states "What" happened, without explaining "Why" it happened or offering "How" suggestions.

Correction Strategy: Encourage causal analysis, predictive reasoning, and attribution.

A.5.2 Evidence–Claim Inconsistency

Internal logical contradictions exist among the conclusions, charts, and text within the report.

- **Conclusion inconsistent with data:** The textual conclusion contradicts the information shown in the cited data.
- **Graph-text mismatch:** The chart itself does not match its title or surrounding text.
- **Cross-chapter contradictions:** Different sections of the report provide contradictory descriptions.

Correction Strategy: Cross-check claims with evidence and visualizations.

A.5.3 Overgeneralization

Drawing overly broad or inappropriate conclusions from limited or biased data.

- **Over-interpretation:** Extending interpretations beyond what the data supports.
- **Ignoring limitations:** Drawing conclusions without considering inherent data limitations.
- **Statistical misjudgment:** Incorrectly interpreting statistical test results.

Correction Strategy: Include uncertainty, confidence intervals, and note limitations.

A.5.4 Irrelevance

Generated insights or information, while possibly correct, are unrelated to the user's core analytical objective.

- **Deviates from main goal:** The analysis strays off-topic and fails to answer the user's core question.
- **Redundant information:** The report contains excessive "correct nonsense" unrelated to the goal.
- **Wrong focus:** Focusing excessively on minor details while ignoring primary findings.

Correction Strategy: Align insights with objectives and remove redundant details.

A.5.5 Omission

Key findings or necessary analytical steps are missing from the final report or conclusion.

- **Missing key findings:** Failing to report or emphasize the most important insights.
- **Missing comparison or risk assessment:** Lacking necessary comparisons for context or failing to assess potential risks.

Correction Strategy: Cross-check analysis plan and ensure coverage of all critical points.

A.5.6 Hallucination

The agent generates content containing entirely fabricated information.

- **Invented facts:** Fabricating events, data, or facts that do not exist.
- **Fabricated details:** Adding plausible-sounding details to invented facts.
- **False sources:** Citing non-existent reports, studies, or experts.

Correction Strategy: Verify all facts and sources; mark uncertain claims.

A.5.7 Redundant Insight

Identical findings or conclusions appear repeatedly in different parts of the report.

- **Repeated findings:** The same insight is reiterated without providing new angles or incremental information.

A.5.8 Misattributed Insight

Providing an incorrect explanation or attribution for a valid observation.

- **Wrong causal attribution:** The observed phenomenon is valid, but it is attributed to an incorrect cause.

A.6 Communication Failures

Communication failures pertain to failures in the structure, terminology, formatting, or narrative of the final report.

A.6.1 Structural Incoherence

The overall organizational structure of the report is chaotic and lacks logical flow.

- **Missing sections:** The report lacks necessary components, such as a conclusion or abstract.
- **Chapter/section disorder:** The arrangement of chapters does not follow a logical progression.

Correction Strategy: Use standardized templates and verify section completeness and order.

A.6.2 Terminology Inconsistency

Different or ambiguous terms are used for the same concept throughout the report.

- **Ambiguous terms:** Using terms with unclear meanings without providing explicit definitions.
- **Inconsistent naming:** Using different names for the same person, product, or metric.

Correction Strategy: Standardize terminology and verify consistency.

A.6.3 Formatting Failure

Issues with the layout, charts, or table formats of the report.

- **Layout / table / figure issues:** Unappealing document layout, misaligned table borders, or illegibly small chart fonts.

Correction Strategy: Use formatting templates and automated style checks.

A.6.4 Evidence Chain Breakage

The argumentative process lacks traceability, making conclusions appear as if they were drawn out of thin air.

- **No traceable reasoning:** Readers cannot understand how conclusions were derived step-by-step from the raw data.

Correction Strategy: Ensure every claim is linked to verifiable evidence.

A.6.5 Visualization Ambiguity

Poorly designed charts or graphs make their intended meaning difficult to understand.

- **Graph/figure meaning unclear:** Charts lack necessary legends or axis labels, preventing accurate interpretation.

Correction Strategy: Add labels, legends, and descriptions to clarify visualizations.

A.6.6 Reference Missing

Data, methods, or perspectives cited in the report lack source attribution.

- **Missing citations for data or methods:** Utilizing external data without providing corresponding references.

Correction Strategy: Verify all claims are backed by references.

A.6.7 Narrative Bias

Inappropriate subjectivity or bias is introduced when describing facts or interpreting data.

- **Subjective or biased description:** Using emotionally charged or leading language to describe analytical results.

Correction Strategy: Review narrative for neutrality and objectivity.

B Extended Methodology and Empirical Analysis

This section provides the formal mathematical definitions, extended quantitative results, and deeper mechanistic analysis to supplement the empirical findings presented in the main text.

B.1 Detailed Distribution of Failure Types

While the main text highlights the dominance of the *Insight* (40.8%, $N = 394$) and *Data* (30.3%, $N = 293$) phases, our macroscopic analysis reveals a broader taxonomy. *Cognitive planning* failures represent a substantial portion (15.6%, $N = 151$), while failures in *Execution* (6.8%, $N = 66$), *Method* selection (3.9%, $N = 38$), and *Communication* (2.5%, $N = 24$) occur less frequently but still contribute to pipeline breakdowns.

At the finer-grained level (L2), the distribution exhibits a clear long-tailed pattern. Beyond *Schema Failure* (2.1), the most frequent L2 failure types include *Evidence-Claim Inconsistency* (5.2, $N = 130$), *Omission* (5.5, $N = 97$), and *Step Planning Failure* (1.5, $N = 93$).

As shown in the main text’s Table 1, zooming in to the leaf-level (L3) categories reveals a highly skewed distribution. Aside from the dominant *Field misinterpretation* (2.1.1) and *Missing key findings* (5.5.1), other notable high-frequency L3 failures include specific forms of *Evidence-Claim Inconsistency* (5.2.1 *Conclusion inconsistent with data* and 5.2.3 *Cross-chapter contradictions*) and *Step Planning Failure* (1.5.3 *Redundant steps*). This concentration confirms that failures are heavily localized in a few specific types of execution and reasoning breakdowns.

B.2 Formalization of Single-Failure Impact

To systematically quantify the marginal impact of each failure category discussed in the main text, we utilize Cliff’s Delta (Cliff, 1993). Given trajectories containing failure e ($\mathcal{D}_e$) and those without ($\mathcal{D}_{-e}$), the effect size is formally defined as:

$$\delta_e = \frac{\#(y_i > y_j) - \#(y_i < y_j)}{|\mathcal{D}_e| \cdot |\mathcal{D}_{-e}|}$$

where y_i denotes the final task score. As illustrated in Fig. 5a and Fig. 5b, the mathematical formulation confirms the statistical significance ($p < 0.05$ (Pearson, 1900)) of Insight-level failures (e.g., 5.1 and 5.4). It further highlights that failures deviating from the core analytical objective disproportionately penalize the agent compared to minor formatting or communication issues.

B.3 Metrics for Failure Co-occurrence and Cascades

Building upon the pipeline effects introduced in the main text, we mathematically quantify structural dependencies using the Jaccard Similarity Index. Given two failure types e_i and e_j , let $\mathcal{T}_{e_i}$ and $\mathcal{T}_{e_j}$ denote the sets of agent trajectories where each failure respectively occurs. The index (Jaccard, 1901) is defined as:

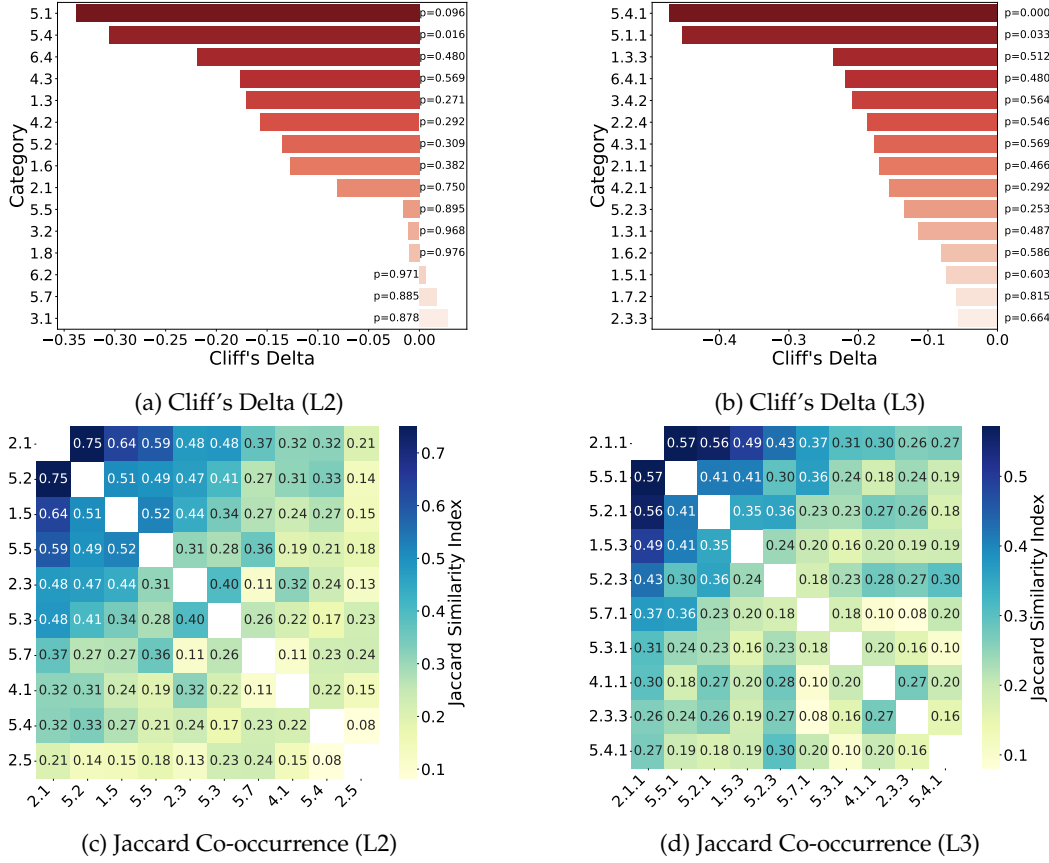


Figure 5: Extended statistical analysis of single-failure impacts and failure co-occurrences at L2 and L3 granularity.

$$J(e_i, e_j) = \frac{|\mathcal{T}_{e_i} \cap \mathcal{T}_{e_j}|}{|\mathcal{T}_{e_i} \cup \mathcal{T}_{e_j}|}$$

Beyond the strong 2.1 and 5.2 association (0.75) noted in the main text, the L2 heatmap (Fig. 5c) reveals that 2.1 *Schema Failure* also heavily co-occurs with 1.5 *Step Planning Failure* (0.64) and 5.5 *Omission* (0.59).

Zooming into the L3 fine-grained level (Fig. 5d) provides a clearer mechanism: the tight coupling between 2.1.1 *Field misinterpretation* and 5.5.1 *Missing key findings* (0.57) demonstrates that misinterpreting a specific column systematically blinds the agent to critical data trends. Additionally, 1.5.3 *Redundant steps* shows moderate to high overlap with the aforementioned failures (e.g., 0.49 with 2.1.1), implying that when agents struggle with data interpretation, they engage in repetitive, non-productive exploratory loops.

B.4 Extended Synergistic Analysis

To capture the non-linear interactions between failures, we define a synergy index over co-occurring pairs:

$$S(e_a, e_b) = \mathbb{E}[y \mid e_a, e_b] - \min(\mathbb{E}[y \mid e_a], \mathbb{E}[y \mid e_b])$$

As shown in Fig. 6a and Fig. 6b, the combination of *Schema Failure* (2.1.1) and *Insight Failures* (e.g., 5.1.1) yields the most negative synergy. Adjacency analysis of these trajectories reveals a consistent behavioral cascade: Patterns Only → Repeated findings →

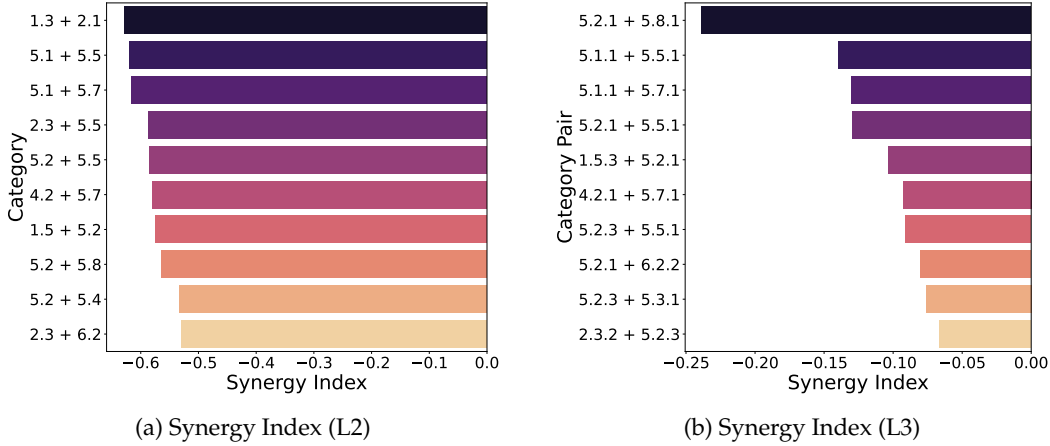


Figure 6: Synergistic effects of failure combinations, illustrating how co-occurring failures amplify negative impacts on performance.

Missing key findings. This suggests that failing to derive reasoning-based insights early logically guarantees repetitive observations and omitted critical information downstream, creating a compounding failure state that is exceedingly difficult for the agent to self-correct.

B.5 Formalization of Cross-Category Cascades

To construct the transition graphs presented in the main text (Figure 3b), we define the L1 transition matrix as:

$$M_{ij} = P(C_j | C_i)$$

where C_i denotes high-level failure categories. Complementing the transition probabilities, our temporal analysis (main text Figure 3a) partitions trajectories into chronological stages. The data confirms that early stages are dominated by Cognitive and Data failures, while late stages are overwhelmingly dominated by Insight failures. This temporal separation proves that failures introduced early often remain latent during the coding/execution steps, only manifesting catastrophically during final reasoning and report generation.

C Detailed Experimental Settings

C.1 Baselines

To comprehensively evaluate the effectiveness of our proposed TGT-Critic framework, we compare it against several strong baseline agents:

- **ReAct** (Yao et al., 2022): A foundational agent paradigm that interleaves reasoning and acting to solve complex tasks.
- **Pandas Agent** (LangChain, 2024): A widely used standard agent optimized for tabular data manipulation via Python’s Pandas library.
- **AgentPoirot** (Sahu et al., 2025): A recent agent framework specifically designed for data analytics and insight generation.
- **Aime** (Shi et al., 2025): A hierarchical Planner-Actor agent system capable of handling long-horizon tasks.
- **Baseline (Standard)**: Our underlying two-stage agent scaffold consisting of a high-level planner and a tool-using actor, without any runtime critic.

- **Baseline (+ Generic Critic):** The same Planner–Actor scaffold augmented with a generic runtime critic prompt, but without taxonomy-grounded retrieval or structured failure typing.

C.2 Dataset

Our evaluation spans two primary datasets designed for evaluating data analysis agents: **InsightBench** (Sahu et al., 2025) and **InsightEval** (Zhu et al., 2025). Both datasets feature multi-step, open-ended data analysis tasks. To systematically assess the agents’ capabilities under varying cognitive loads, the tasks in both datasets are categorized into three difficulty levels: **Easy**, **Medium**, and **Hard**. These levels are defined based on the complexity of the required reasoning, the depth of data manipulation, and the length of the analytical trajectory.

C.3 Implementation Details

This section provides additional implementation details for reproducibility.

Model Configuration. We use a controlled model configuration to ensure that performance differences are attributable to the critic design rather than backbone changes.

- **Agent backbone.** For our **Baseline (Planner–Actor)**, **Baseline (+ Generic Critic)**, and **Baseline (TGT-Critic)**, all agent-side modules use **DeepSeek-V3-250324**, including the planner, actor, diagnostician, and failure critic when applicable.
- **Evaluation model.** We use **GPT-4o-2024-11-20** only as the LLM judge in the G-Eval evaluation pipeline.
- **Sampling.** The sampling temperature for agent generation is fixed at **0.2**.

Critic Variants. To isolate the contribution of taxonomy grounding, we compare three variants built on the same underlying Planner–Actor scaffold:

- **Baseline (Planner–Actor):** no runtime critic.
- **Baseline (+ Generic Critic):** a generic runtime critic without taxonomy-grounded retrieval.
- **Baseline (TGT-Critic):** our full taxonomy-guided runtime critic.

Retrieval Hyperparameter. For TGT-Critic, the number of retrieved examples is set to $k = 3$ by default. As shown in Table 3, this choice provides the best balance between useful diagnostic guidance and retrieval noise.

Evaluation Protocol. We evaluate generated analytical reports with the **G-Eval** framework and report **Summary-level** following prior work Sahu et al. (2025). This metric measures the overall quality of the final reports in terms of accuracy, comprehensiveness, and helpfulness.

D Case Studies on Critic Interventions

In this section, we present detailed case studies extracted from empirical execution trajectories to qualitatively illustrate how the Taxonomy-Guided Trajectory Critic (TGT-Critic) intervenes in the agent’s reasoning loop. Based on the analysis of over 100 execution trajectories from InsightBench, these cases demonstrate how the critic maps observed anomalies to our predefined Agent Failure Taxonomy, dynamically adjusting both high-level plans and low-level actions.

D.1 Identifying Missing Dependencies and Structural Flaws

During the formulation of the initial plan, the TGT-Critic acts as a forward-looking safeguard. It ensures that the agent does not blindly execute tasks without establishing necessary prerequisites, effectively preventing “garbage in, garbage out” scenarios and scope creep.

Large language models frequently assume the existence of specific data schemas without empirical verification. For instance, in **Case 12**, the agent formulated a plan assuming direct access to specific incident assignment fields. The TGT-Critic intercepted this as a *Missing Data Dependency*, forcing the agent to inject an exploratory data analysis (EDA) step to validate schema availability. Similarly, in **Case 16**, the critic identified that the agent intended to calculate correlations between cost and warranty data without first standardizing the financial metrics or checking for missing values, thereby preventing a flawed analytical pipeline. Furthermore, in **Case 6**, the agent attempted to bypass macro-level workload drill-downs to directly perform text mining on specific “Software” cases. The critic issued a severe warning for *Premature Narrowing of Scope*, forcing the agent to revert to a systematic, top-down investigation to rule out systemic inefficiencies first before diving into granular categories.

Table 4 presents direct excerpts from the execution trajectories corresponding to these cases.

Table 4: Trajectory Excerpts of Critic Interventions during Structural Planning

Case	Identified Anomaly	Agent’s Original Trajectory (Excerpt)	Critic’s Feedback (Excerpt)
12	Missing Data Dependency	"Plan_Step_1": "Filter dataset by 'assignment_group' and 'assigned_to' fields to calculate resolution times."	"[Critic-Reject]: The fields 'assignment_group' and 'assigned_to' have not been verified in the schema. Missing prior EDA step to check column existence."
16	Assumption Hallucination	"Action": "df['cost'].corr(df['warranty_period'])" "Thought": "Calculating direct Pearson correlation between asset cost and warranty period."	"[Critic-Halt]: Currency formatting ('\$') and null values in 'cost' column not handled. Must standardize metrics before correlation."
6	Premature Narrowing of Scope	"Task": "Analyze 'Software' category descriptions using TF-IDF to find common themes."	"[Critic-Warning]: Bypassing macro-analysis. You must first analyze the distribution of ALL categories to justify why 'Software' is the bottleneck."

D.2 Enforcing Methodological Rigor and Bridging Execution Gaps

Once execution begins, the TGT-Critic monitors the trajectory to ensure the agent does not settle for superficial descriptive statistics, fall into redundant loops, or omit critical comparative dimensions.

While attempting to detect fraudulent expense claims (**Case 42**), the agent executed an exact string-matching function, found no duplicates, and inexplicably scheduled the exact same check for the next step. The TGT-Critic pruned this redundant loop and criticized the simplistic methodology (*Flawed Assumption*), forcing the agent to escalate to fuzzy matching for near-duplicates to ensure analytical rigor. Furthermore, agents often fail to conduct cross-sectional comparative analysis. In **Case 17**, the agent observed that HR department equipment costs were significantly higher than those of other departments and prematurely logged this as an anomaly. The critic intervened, pointing out a critical missing temporal dimension: the HR equipment might simply be newer. This forced the agent to standardize the cost metrics against asset “Age”, thereby preventing a highly misleading conclusion.

Table 5 highlights the specific execution steps where the TGT-Critic intervened.

Table 5: Trajectory Excerpts of Critic Interventions during Methodological Execution

Case	Identified Anomaly	Agent’s Original Trajectory (Excerpt)	Critic’s Feedback (Excerpt)
42	Redundant Loop & Flawed Assumption	"Action": "df[df.duplicated(subset=['expense_amount'])]" "Next_Step": "Re-run duplicate check on subset."	"[Critic-Reject]: Exact match found 0 results. Scheduling the same exact match is a redundant loop. Escalate methodology to fuzzy matching (e.g., Levenshtein distance) for near-duplicates."
17	Omission of Synthesis	"Observation": "HR department mean equipment cost is \$2400, highest among all. Log as primary cost anomaly."	"[Critic-Halt]: Missing cross-sectional synthesis. Higher cost may correlate with newer procurement dates. You must standardize cost against 'asset_age' before concluding it is an anomaly."

D.3 Rectifying Analytical Fallacies and Hallucinations

At the final reporting stage, the TGT-Critic serves as a strict statistical and logical gatekeeper, preventing the generation of hallucinated, speculative, or statistically invalid business insights.

The agent frequently misinterprets statistical significance. In **Case 8**, the agent compared the resolution times of two dimensions (173.20 hours vs. 178.59 hours) and hastily concluded that the former was “faster.” The critic immediately flagged this as an *Insight Generation Failure*, noting that against a standard deviation reaching hundreds of hours, a 5-hour difference is statistically negligible. Additionally, agents tend to hallucinate causal relationships. In **Case 44**, the agent observed a lower approval rate (26.8%) for certain asset expenses. Rather than exploring policy constraints, it generated an insight accusing the company of systemic “bias.” The critic rigorously dismantled this claim, categorizing it as *Evidence-Claim Inconsistency*. It reminded the agent of the difference between correlation and causation, requiring it to investigate underlying policy contexts rather than making unproven extrapolations. Finally, in **Case 40**, the agent encountered an impossibly fast processing period (-6.13 days). Instead of questioning the anomaly, it began drafting insights attributing this “speed” to departmental efficiency. The critic flagged this fundamental data validity concern (*Schema/Data Quality Failure*), forcing the agent to clean the corrupted timestamps before synthesizing any final metrics.

Table 6 displays the trajectory snippets demonstrating the TGT-Critic’s correction of flawed insights.

Table 6: Trajectory Excerpts of Critic Interventions during Insight Generation

Case	Identified Anomaly	Agent's Original Trajectory (Excerpt)	Critic's Feedback (Excerpt)
8	Insight Generation Failure	"Insight": "Dimension A is faster than Dimension B (173.2h vs 178.6h)."	"[Critic-Reject]: Statistically invalid claim. With a standard deviation of >100 hours, a 5-hour delta is within the margin of error (insignificant)."
44	Evidence-Claim Inconsistency	"Insight": "Systemic bias exists against Category X due to its low 26.8% approval rate."	"[Critic-Halt]: Unproven causation hallucinated from correlation. You cannot claim 'bias' without checking policy parameters (e.g., budget caps, missing receipt flags)."
40	Schema/Data Quality Failure	"Insight": "Dept Y is highly efficient, processing requests in an average of -6.13 days."	"[Critic-Warning]: Logical impossibility detected. A negative processing time indicates corrupted timestamps. Do not generate insights; trigger data cleaning protocol immediately."