AgentDebug: Where LLM Agents Fail and How They can Learn From Failures

Anonymous ACL submission

Abstract

Large Language Model (LLM) agents combine planning, memory, reflection, and tool use, but remain vulnerable to error propagation: one root-cause mistake can cascade through later decisions and cause task failure. Existing evaluations rarely identify such causes systematically or translate them into corrective feedback. We introduce *AgentErrorTaxonomy*, a modular taxonomy of agent failures, and *AgentErrorBench*, a benchmark of 200 expert-annotated failed trajectories from ALFWorld, GAIA, and WebShop. We further propose *AgentDebug*, which decomposes rollouts, isolates minimal root-cause failures, and returns actionable feedback for recovery. To scale supervision, we curate 1,993 proposer–reviewer debugging annotations for supervised fine-tuning of *AgentDebugger-4B*. On *AgentErrorBench*, *AgentDebug* achieves **24%** higher all-correct accuracy and **17%** higher step accuracy than the strongest baseline, while its feedback yields up to **26%** relative improvements in task success across the evaluated environments. The fine-tuned debugger also improves over its base model and outperforms GPT-4.1 on all-correct error detection (**4.7%** vs. **0.3%**).

1 Introduction

LLMs are increasingly capable of interacting with external environments, leveraging tools, and reasoning over memory, which enabled a new paradigm: LLMs Agents (Schick et al., 2023; Qin et al., 2023; Packer et al., 2023; Shinn et al., 2023; Liu et al., 2025). General purposed LLMs agents have driven advances across diverse domains, including embodied control, scientific discovery, open-ended web interaction, and research support (Zhou et al., 2023; Li et al., 2025; Hong et al., 2025; Zhu et al., 2025b). Despite these achievements, current agents remain imperfect and insufficiently robust. They frequently exhibit errors—ranging from misinterpreting instructions and misusing tools to breaking down in long-horizon reasoning. These shortcomings underscore that, while promising, existing LLM Agents still lack the reliability required for real-world deployment. This observation motivates us to ask: ***Where do LLM agents fail*** ?

While prior works have investigated the failures of LLM-based agents (Ji et al., 2024; Sung et al., 2025; Ning et al., 2024; Cemri et al., 2025; Zhang et al., 2025b), they have largely focused on enumerating error types or providing qualitative case studies. However, these analyses stop short of offering systematic mechanisms to trace failures back to their *root causes*, and importantly, do not enable agents to fix these discovered failures based on such insights. To close this gap, we modularize error analysis with the explicit goal of tracing failures to their underlying causes. We conduct a large-scale study over hundreds of trajectories, decomposing each rollout into four operational modules: memory, reflection, planning, and action. By attributing each failure to its root module, we derive the *AgentErrorTaxonomy* — a comprehensive taxonomy of failure modes designed to ground systematic error detection and guide the development of robust mitigation strategies. To support systematical evaluation on whether models can detect critical errors and produce actionable feedback, we construct the *AgentErrorBench*. This benchmark grounds the taxonomy in real-world trajectories, offering the first standardized testbed for error detection and debugging.

Our analysis supports a critical bottleneck in LLM agents: *error propagation*.A single root-cause failure can cascade into successive errors, compounding degradation and leading to task failure. This challenge is especially acute in long-horizon tasks, where early mistakes distort later reasoning and actions, making recovery difficult. These insights motivate our central question: ***How can LLM agents refine from failures***?

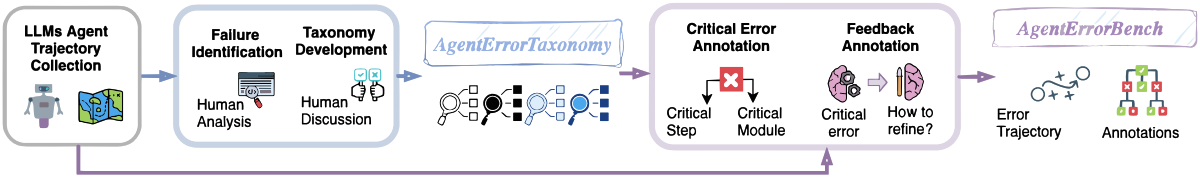


Figure 1: **Pipeline of proposed *AgentErrorTaxonomy* and *AgentErrorBench*.** Failed trajectories are collected, analyzed to develop a taxonomy of errors, and then annotated with root causes and actionable feedback to form the benchmark.

Recent progress in LLMs agent enhancement has largely centered on expanding the reasoning search space—through chain-of-thought (Wei et al., 2023), tree-of-thought (Yao et al., 2023), and graph-of-thought (Besta et al., 2023) approaches—or on incorporating step-level self-reflection (Yao et al., 2022b, 2023; Besta et al., 2023), which allows agents to deliberate more thoroughly and locally revise their actions. While these advances improve flexibility and reasoning depth, they typically treat the trajectory as a sequence of isolated steps rather than as a coherent, interdependent process.

To address this challenge, we propose *AgentDebug*, a debugging framework that decomposes a trajectory into decision steps, isolates the minimal set of root-cause failures, and provides corrective feedback back to the responsible states or actions. Beyond the framework, we also design an automated data curation pipeline to construct AgentDebug-Traj2k, a large-scale dataset generated via proposer–reviewer collaboration, where multiple models propose debugging analyses and consolidates them into high-quality annotations. Build upon this dataset, we train AgentDebugger-4B, a specialized model for agent debugging.

We evaluate *AgentDebug* on *AgentErrorBench*, where it achieves 24% higher all-correct accuracy (24.3% vs. 0.3%) and 17% higher step accuracy (45.0% vs. 28.0%) compared to the strongest baseline, demonstrating its ability to both accurately detect errors and maintain consistency across multi-step trajectories. Beyond detection, the targeted feedback generated by *AgentDebug* enables agents to iteratively recover from failures, yielding up to 26% relative improvements in task success across ALFWorld, GAIA, and WebShop, highlighting the practical impact of our framework in long-horizon, real-world scenarios where error propagation is a critical challenge. In addition, *AgentDebugger-4B* substantially improves over its base model, Qwen3-4B-Instruct, and even outperforms GPT-4.1 in error detection (4.7% vs. 0.3% on *All-Correct*), enabling strong performance without relying on larger proprietary systems.

2 Where do LLM Agents Fail?

To lay the foundation for our study of agent failures and debugging, this section introduces two *proposed* components. Section 2.1 presents the *AgentErrorTaxonomy*, a structured framework that organizes recurring failure modes into coherent modules. Section 2.2 then describes the *AgentErrorBench*, which grounds this taxonomy in systematically annotated trajectories from multiple environments.

2.1 Agent Failure Analysis

To systematically investigate how agents fail, we collected over 500 failed trajectories from ALFWorld, WebShop, and GAIA, and conducted detailed human analyses to uncover recurring patterns. This study revealed the following central insight:

> ***Key Insight.*** *Error propagation is the primary bottleneck in LLM agent reliability.* Early mistakes rarely remain confined; instead, they cascade into subsequent steps, distorting reasoning, compounding misjudgments, and ultimately derailing the entire trajectory.

Building on this insight, we introduce the *AgentErrorTaxonomy*, a structured taxonomy that organizes recurring failure modes into five modules. Four capture the core operations of an agent—*memory*, *reflection*, *planning*, and *action*—while a fifth category accounts for *system-level* errors that arise from external tools or infrastructure. This modular view not only enumerates common error types but also clarifies how weaknesses in one stage can cascade into others, enabling a principled approach to tracing and diagnosing failures.

The *AgentErrorTaxonomy* is designed not merely as a catalog of errors but as a causal lens for understanding how failures originate, propagate, and

interact across modules. Full definitions are provided in Appendix A.1.

2.2 Agent Error Benchmark

To rigorously evaluate whether models can detect critical errors and produce actionable feedback, we construct the *AgentErrorBench*. This benchmark grounds the taxonomy in real-world trajectories, offering the first standardized testbed for error detection and debugging.

Memory	Errors in recalling or retrieving information (false recall, omission, retrieval failure) that distort subsequent reasoning.
Reflection	Failures in monitoring progress or interpreting outcomes (progress misassessment, outcome misinterpretation), blocking course correction.
Planning	Logically unsound or infeasible strategies (impossible actions, constraint ignorance, incoherent subgoals) that cascade into missteps.
Action	Mistakes in executing operations (malformed outputs, incorrect parameters, missing arguments) that are visible but often mask upstream errors.
System-level	Failures outside reasoning, such as tool crashes, API mismatches, or exceeding step limits, highlighting system robustness issues.

The construction pipeline is shown in Figure 1. We curated 200 representative trajectories: 100 from ALFWorld, 50 from WebShop, and 50 from GAIA. Ten expert annotators—graduate students with prior experience in NLP and LLMs agent research—labeled each trajectory using the *AgentErrorTaxonomy* schema. Annotation proceeded at the decision-step level: every action, reflection, or plan was reviewed, and annotators tagged its error type(s) according to the taxonomy. In addition, they were tasked with identifying the minimal set of root-cause failures that explain the downstream error cascade, rather than exhaustively flagging all surface mistakes. This root-cause focus was emphasized through detailed guidelines and calibration examples, refined iteratively over three rounds of pilot annotation.

To ensure consistency, annotators first completed a training phase with feedback from the authors, followed by independent double-annotation on a shared subset of trajectories. Disagreements were adjudicated collectively, leading to several clarifications in category definitions (e.g., distinguishing "retrieval failure" under memory vs. "constraint ignorance" under planning). The final protocol balanced granularity with reliability, aiming for concise but causally meaningful tags. While pairwise agreement is moderate, final labels are obtained via expert adjudication, and most disagreements involve near-neighbor steps rather than qualitatively different root causes (Appendix A.5).

The resulting dataset provides a quantitative view of how agents fail in practice. Figure 15 shows how errors emerge across steps, modules, and error types in the *AgentErrorBench*. It shows most failures cluster in mid-trajectory steps (6–15), where early missteps often cascade downstream. Memory and reflection dominate, with retrieval failures, hallucinations, and progress misjudgments leading to flawed planning. Action and system errors occur less frequently but remain critical, as malformed outputs or step-limit exhaustion can immediately terminate trajectories. The complete error type distribution is attached to Appendix A.10.

3 How to Refine LLM Agents from Failures?

Building on our analysis of agent failures, we aim to develop method that actively refine themselves by learning from mistakes. We first formalize the key modules of an LLM-based agent in Section 3.1, and then present our proposed framework, *AgentDebug*, in Section 3.2.

3.1 Key Modules of an LLM Agent

As illustrated in Figure 2, we consider an agent that interacts with an environment over a sequence of steps. At each step, the agent observes a state and produces an action, forming a trajectory of state–action pairs. Each action is generated through four sequential modules: **Memory**, which recalls relevant past information; **Reflection**, which evaluates progress and interprets feedback; **Planning**, which formulates the next strategy; and **Action**, which executes the low-level operation. Errors can occur in any of these modules, and mistakes made early often propagate through later stages, com-

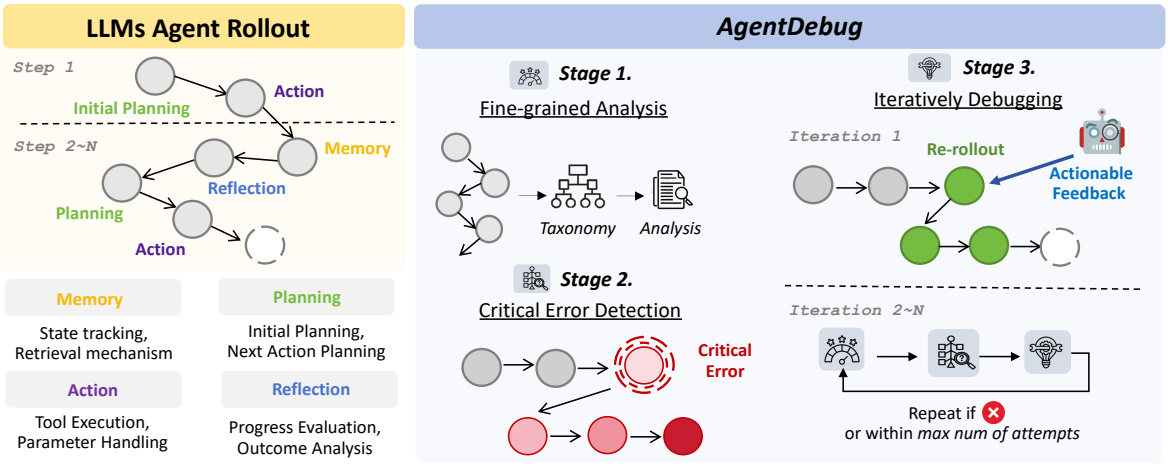


Figure 2: **Overview of *AgentDebug*.** (Left) LLM agent rollouts alternate between memory, planning, reflection, and action. (Right) *AgentDebug* debugs trajectories in three stages: (1) fine-grained analysis across steps and modules, (2) detection of the critical error that triggers failure, and (3) iterative re-rollouts with actionable feedback to turn failures into successes.

pounding into larger failures. This modular rollout design not only mirrors the agent's internal decision process, but also provides a natural structure for human annotators to align errors with specific modules, making the diagnosis of agent failures more transparent and interpretable.

3.2 *AgentDebug* Framework

We propose *AgentDebug*, a debugging framework that enables single-LLM agents to diagnose and recover from their own failures across diverse environments. As illustrated in Fig. 2, *AgentDebug* analyzes complete trajectories to (i) assign fine-grained error types to each step and module, (ii) identify the earliest critical error that directly causes the final failure, and (iii) provide actionable feedback to guide re-rollouts. The central intuition is that correcting a single root-cause mistake can often flip an otherwise failing trajectory into a successful one.

As presented in Algorithm 1, the inference procedure of *AgentDebug* consists of three stages:

Stage 1: Fine-grained Analysis with *AgentErrorTaxonomy*. For each step in the trajectory, we analyze all four modules—memory, reflection, planning, and action. Each module is mapped to an *AgentErrorTaxonomy* error type, grounding the analysis in interpretable categories such as "constraint ignorance" or "format error." This produces a structured, module-level error profile for the trajectory.

Stage 2: Critical Error Detection via Counterfactual Reasoning. If the trajectory is already successful, no debugging is required. Otherwise, we invoke an LLM-based detector that reasons counterfactually over the *annotated* trajectory from Stage 1: for each step, it simulates how the outcome would change *if* the errors at that step were corrected, and infers which step is most likely to flip the final result from failure to success. We define the *critical error* as the earliest step whose hypothetical correction is judged sufficient to prevent the final failure. Unlike superficial mistakes or errors that are later compensated for, the critical error is treated as the root cause that determines whether the overall trajectory ultimately succeeds or fails—capturing both *when* and *why* the agent goes irreversibly off track.

Stage 3: Iterative Debugging with Targeted Feedback. Once the critical error is identified, the system generates feedback that specifies the error type and provides actionable guidance for refining subsequent actions and plans. Grounded in the *AgentErrorTaxonomy* taxonomy, the feedback is both targeted and forward-looking—resolving the root cause while shaping how the agent approaches future steps.

3.3 *AgentDebug*-Traj2k and *AgentDebug*-4B

Beyond the *AgentDebug* framework, we further propose a training recipe that incorporates an automated data curation pipeline to train the specialized model for agent debugging.

Training Data Construction. Our proposed *AgentErrorBench* (Section 2.2) is designed solely for evaluation. To ensure high quality, each tra-

jectory in the benchmark is carefully annotated by human experts. However, its limited scale makes it unsuitable for model training. To address this, we develop an automated data curation pipeline to scale up training data. Inspired by prior works such as Mixture-of-Agents (Wang et al., 2025), ChatEval (Chan et al., 2023), and Multi-Agent-as-a-Judge (Chen et al., 2025), we construct *AgentDebug-Traj2k* using a proposer–reviewer framework. Specifically, multiple models first generate debugging analyses for failure trajectories, and *GPT-4.1* then acts as a meta-reviewer to consolidate and refine these outputs into a final annotation. Further details of the dataset construction are provided in Appendix A.4.

Model Training Building on the constructed dataset, we fine-tune *Qwen3-4B-Instruct-2507* using supervised learning. We train three *AgentDebugger-4B* models, each specialized for a different environment, and additionally obtain a unified variant, *AgentDebugger-4B-Mix*, via mixed-environment training. The resulting *AgentDebugger* analyzes a given trajectory and identifies critical errors, including their corresponding modules and types, along with detailed analyses and actionable suggestions for correction. Further training details are provided in Appendix A.4.9.

4 Experiments and Results

4.1 Critical Error Detection

Dataset. We evaluate critical error localization on *AgentErrorBench* (Sec. 2.2), which consists of 200 annotated failure trajectories. These trajectories are drawn from three representative benchmarks: **ALFWorld** (Shridhar et al., 2020), **GAIA** (Mialon et al., 2023), and **WebShop** (Yao et al., 2022a).

Baselines. We compare against three strategies: (1) **Direct Prompting**, which queries the LLM directly for error localization without any structured search; (2) **Brute Force**, which examines steps sequentially from t=1 to T, stop when the LLM identify the step have critical error; and (3) **Binary Search**, which applies a divide-and-conquer procedure by probing the midpoint of the trajectory and recursively halving the search space until the critical step is isolated. All baseline methods are provided with the same error taxonomy definitions as *AgentDebug*, ensuring that performance differences arise from the structured debugging pipeline rather than access to taxonomy information.

Implementation. We use GPT-4.1 as the backbone model for all baseline methods in Table 1 as well as for the *AgentDebug* framework, with the temperature fixed at 0 to ensure deterministic outputs. In addition, we evaluate Qwen3-4B under the direct prompting setting, which also serves as the backbone for *AgentDebug*-4B and *AgentDebug*-4B-mix. *AgentDebug*-4B is trained on environment-specific datasets, resulting in three separate checkpoints corresponding to the three environments. In contrast, *AgentDebug*-4B-mix is trained on a mixture of all environments. The full prompt template is provided in Appendix A.11, and all methods share the same prompt to ensure fair comparison.

Evaluation Metic. We evaluate critical error detection at multiple granularities, capturing both partial and complete localization ability. Specifically, we measure **Step** accuracy, the ability to identify the exact step where the first critical error occurs; **Step+Module** accuracy, the joint prediction of both the correct step and its module; and **All Correct** (Step+Module+Error Type), a strict metric requiring the exact step, module, and error type to all be correctly identified.

Experiment Results Table 1 shows that *AgentDebug* (stage1+stage2) consistently outperforms all baselines across datasets and evaluation metrics. On average, *AgentDebug* improves the *All-Correct* metric by a large margin (24.3% vs. 0.3%) and achieves substantial gains in Step accuracy (45.0% vs. 28.0%). These improvements are particularly pronounced on GAIA, where Step accuracy nearly doubles (58.0% vs. 30.0%) and *All-Correct* increases more than threefold (38.0% vs. 12.0%), highlighting the effectiveness of our approach in more complex, multi-step settings. We further observe that mixing training data across environments and training a unified AgentDebugger improves step-level and Step+Module (S+M) performance, suggesting better local reasoning and module usage. However, this strategy does not consistently translate to improvements on the stricter *All-Correct* metric, indicating that achieving full trajectory correctness remains challenging. Importantly, *AgentDebugger-4B* significantly outperforms its base model, Qwen3-4B-Instruct, and even surpasses GPT-4.1 on *All-Correct* (4.7% vs. 0.3%), demonstrating the effectiveness of our data curation and training pipeline. The mixed variant, *AgentDebugger-4B-mix*, further improves generalization across environments, achieving higher overall performance than the single-environment model. While some baselines achieve competi-

Method	ALFWorld			WebShop			GAIA			Average		
	S	S+M	ALL	S	S+M	ALL	S	S+M	ALL	S	S+M	ALL
Direct Prompting-GPT4.1	28.0%	14.0%	1.0%	30.0%	6.0%	0.0%	26.0%	10.0%	0.0%	28.0%	10.0%	0.3%
Direct Prompting-Qwen3-4B	11.0%	4.0%	2.0%	10.0%	0.0%	0.0%	24.0%	8.0%	0.0%	15.0%	4.0%	0.7%
Brute Force	10.0%	5.0%	0.0%	8.0%	0.0%	0.0%	18.0%	8.0%	0.0%	12.0%	4.3%	0.0%
Binary Search	20.0%	6.0%	1.0%	14.0%	8.0%	0.0%	22.0%	10.0%	0.0%	18.7%	8.0%	0.3%
***AgentDebug*er-4B**	14.0%	5.0%	4.0%	22.0%	4.0%	0.0%	18.0%	14.0%	10.0%	18.0%	7.7%	4.7%
***AgentDebug*er-4B-mix**	14.0%	7.0%	6.0%	18.0%	4.0%	0.0%	38.0%	18.0%	2.0%	23.3%	9.7%	2.7%
AgentDebug	**35.0%**	**28.0%**	**21.0%**	**42.0%**	**22.0%**	**14.0%**	**58.0%**	**44.0%**	**38.0%**	**45.0%**	**31.3%**	**24.3%**

Table 1: Comparison of methods across three environments (ALFWorld, WebShop, GAIA) and their average performance. Our method consistently outperforms baselines in all settings. Metrics shown are Step Exact (S), Step+Module (S+M), and All Correct.

tive results on certain metrics, this does not contradict our conclusions. Simpler approaches, such as direct prompting or search-based methods, can perform well on less stringent metrics (e.g., Step Exact) by producing locally correct actions or exploiting shortcut patterns. However, they often fail to maintain consistency across steps or properly integrate module-level reasoning, leading to substantially lower performance on more demanding metrics such as S+M and *All-Correct*.

4.2 Downstream Debugging on Single-Agent Benchmarks

We next evaluate whether improved error detection leads to higher task success, testing if localizing root-cause failures enables agents to recover more effectively than baselines.

Baselines. We compare AGENTDEBUG against several strong approaches. The first is Self-Refine, where the agent iteratively revises its outputs without explicit causal diagnosis of errors. The second is a Vanilla Debugger, which applies naive post-hoc corrections to failed trajectories without identifying the critical error taxonomy. Finally, we include strong test-time scaling baselines: Tree-of-Thought (ToT) and Best-of-N. To ensure fairness, the max number of attempts of all baselines is matched to AGENTDEBUG, so any observed gains can be attributed to targeted error recovery rather than higher resource allocation.

Implementation. We implement AGENTDEBUG with up to $N = 5$ re-rollouts, each beginning precisely at the identified critical step. This design enables the agent to explore alternative continuations directly from the point of failure rather than restarting from the beginning of the trajectory, thereby concentrating computational effort where it is most impactful. For backbone models of LLMs Agent, we evaluate across three representative systems of varying scales and architectures: GPT-4o-mini, Qwen3-8B, and Qwen3-Next-80B.

Experiment Results. Figure 3 reports the performance of *AgentDebug* on the ALFWorld benchmark across three backbone LLM agents. On GPT-4o-mini, *AgentDebug* boosts success from 21 (first attempt) to 55; on Qwen3-8B, from 48 to 74; and on Qwen3-Next-80B, from 60 to 84. These results show that *AgentDebug* consistently outperforms all baselines and can effectively help LLM agents improve regardless of the backbone model, with especially large relative gains for smaller models.

To further assess the generalizability of *AgentDebug*, we move beyond ALFWorld and extend our evaluation to two additional benchmarks that stress complementary dimensions of agent reasoning: GAIA, which requires integrating open-domain knowledge and web-based tool use, and WebShop, which evaluates agents in a realistic e-commerce setting with structured constraints and transactional goals. Together, these benchmarks provide a diverse testbed spanning embodied interaction, knowledge-grounded reasoning, and goal-directed web navigation. Given the higher computational demands of these settings, we focus on comparing *AgentDebug* against the two strongest baselines from our earlier experiments (Self-Refine and Best-of-N), which represent state-of-the-art approaches to iterative refinement and test-time scaling, respectively.The results, shown in Figure 4, demonstrate that *AgentDebug* consistently delivers the largest gains across all three benchmarks. In particular, it achieves improvements of up to 26% on ALFWorld and strong average performance on GAIA and WebShop, underscoring its robustness across diverse environments and confirming that targeted error detection and correction can outperform broader but less focused strategies such as scaling rollouts or unguided self-revision.

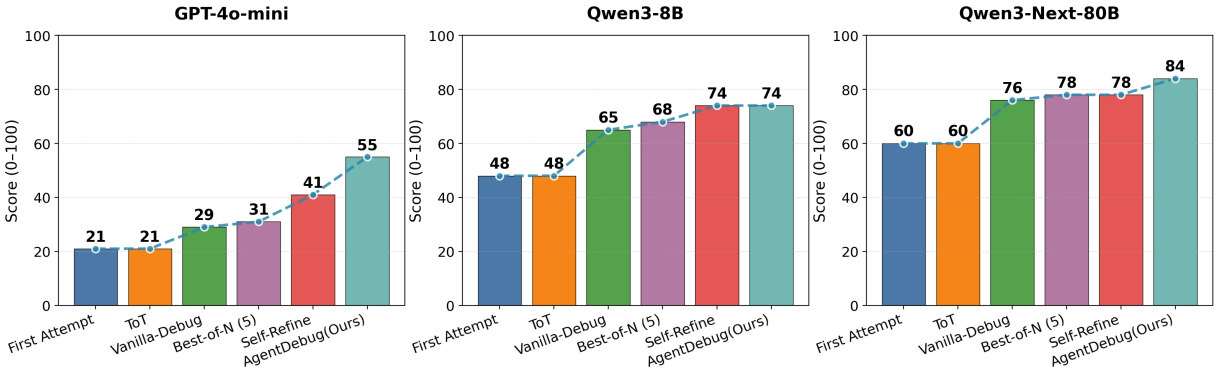


Figure 3: Downstream debugging performance on ALFWorld. *AgentDebug* consistently outperforms strong baselines across backbone models and methods.

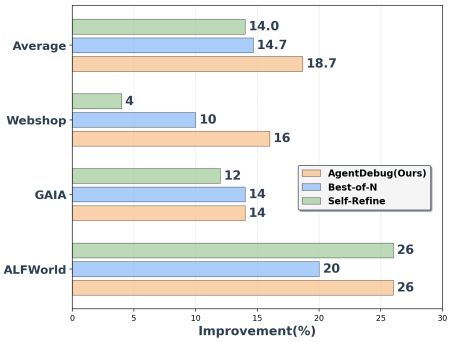


Figure 4: Performance improvements comparison.

Findings. *AgentDebug* consistently outperforms strong baselines in both error detection and downstream task success. Its ability to precisely localize root-cause failures (50.0% step accuracy and 42.5% all-correct accuracy) translates into substantial improvements in task performance, achieving up to 26% relative gains across ALFWorld, GAIA, and WebShop.

4.3 Ablation Study

To better understand the contribution of each component in our framework, we conduct a series of ablation studies. Specifically, we analyze the following factors.

Max Number of Attempts Allowed. We vary the maximum number of attempts of *AgentDebug* with different base models— GPT-4o-mini, Qwen3-8B, and Qwen3-Next-80B—to examine how sensitive performance is to the chosen point of intervention. Results in Figure 5a show that additional attempts yield consistent gains across all backbones, with especially pronounced improvements on smaller models such as GPT-4o-mini. This highlights the value of targeted re-rollouts in boosting task success.

***AgentDebug* Base Models.** We replace the base model of *AgentDebug* with several alternatives to assess their impact on error localization and downstream success. As shown in Figure 5b, GPT-4.1 substantially outperforms other models, achieving 42% step accuracy and 32% strict all-correct accuracy. Competing base models such as Llama-3.3-70B, GPT-4o-mini, and Qwen3-Next-80B perform markedly worse.

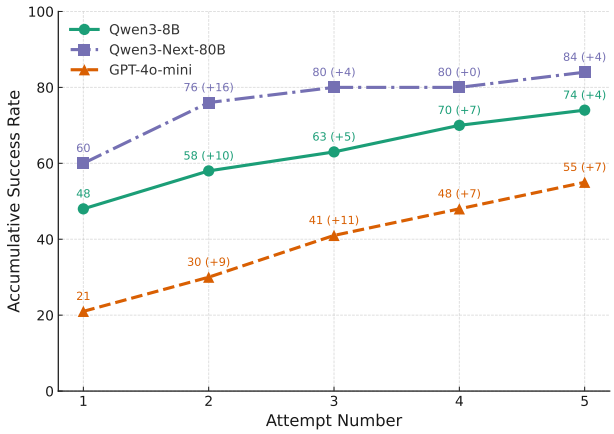
Rollout Strategies. We compare several rollout paradigms—ReAct, Reflection, Act-only, and Memory+ReAct—against our proposed **Modular** strategy on Alfworld under the zero-shot setting. As shown in Table 5c, *AgentDebug*'s Modular rollout achieves the highest score (0.38), outperforming both reasoning-only and memory-augmented baselines. This result highlights the effectiveness of structured modularization is more reliably than alternative rollout designs.

4.4 Error Propagation

Our analysis validates that error propagation is the central bottleneck in LLM agents. As shown in Figure 9, 62% of mid- and late-stage errors on ALFWorld are causally dependent on an earlier mistake, and errors before step 6 increase the probability of three or more downstream failures from 27% to 61%. Memory and reflection errors are the most common sources of cascades, as misremembered facts or misjudged progress systematically distort subsequent planning. These findings motivate two design principles: (1) **early detection and correction are critical**, since cascades are difficult to reverse once initiated, and (2) mechanisms that strengthen memory retrieval accuracy and reflection can substantially reduce propagation risk. Extended analysis is provided in Appendix A.7.

4.5 Error Analysis of AgentDebug

We conduct a systematic error analysis of *AgentDebug* across five backbone models on *AgentEr-*



(a) Success rates on Alfworld across five attempts.

Base Model	Step	Error	Step+M	All
Llama-3.3-70B	16.0	16.0	6.0	2.0
GPT-4o-mini	14.0	10.0	4.0	2.0
Qwen3-Next-80B	4.0	14.0	2.0	2.0
GPT-4.1	**42.0**	**44.0**	**32.0**	**32.0**

(b) Base Model Performance Comparison.

Rollout Strategy	Success Rate
Memory+ReAct	0.34
Reflection	0.32
ReAct	0.26
Act Only	0.10
Modular	**0.38**

(c) Rollout Strategy Comparison.

Figure 5: Ablation study results.

rorBench. Step localization is the dominant bottleneck, accounting for 75–85% of all detection failures (Table 10). Once the correct step is identified, module and error-type classification is comparatively accurate ($\leq$17% and $\leq$3% error rates, respectively). Among modules, memory and system errors are hardest to detect (0–7% All-Correct), while action errors are easiest (up to 33%). Detection accuracy also degrades sharply with step position (Table 15). We additionally evaluate reasoning models (Kimi-K2, MiniMax-M2.5) as backbone detectors; MiniMax-M2.5 matches GPT-4.1 on step-level accuracy (24.7% vs. 24.0%) but trails on full detection (6.7% vs. 11.0%). Full breakdowns by benchmark, module, error type, and step position are in Appendix A.8.

5 Related Work

LLM-based Agentic Systems. Recent advances in LLM-based agents have demonstrated that combining planning, tool use, memory, and self-reflection can substantially improve performance on complex, multi-step tasks. The reasoning-acting paradigm, pioneered by ReAct (Yao et al., 2022b), interleaves chain-of-thought reasoning with grounded environment actions. Test-time search methods such as Tree-of-Thoughts (Yao et al., 2023) and Graph-of-Thoughts (Besta et al., 2023) further expand the deliberation space through structured exploration. These systems are strengthened by external tool integration, ranging from self-supervised tool selection in Toolformer (Schick et al., 2023) to comprehensive tool ecosystems in ToolLLM (Qin et al., 2023), as well as persistent memory mechanisms that maintain context across long horizons (Packer et al., 2023). A critical aspect of robustness is the ability to reflect and self-correct: Reflexion (Shinn et al., 2023) introduces verbal self-reflection to revise future actions. Despite recent advances, the community still lacks a unified view of how agent components interact and fail.

Failure Analyses (Single- and Multi-Agent). Systematic failure analyses are emerging to quantify where agents break and how errors propagate. In multi-agent settings, recent works examine collaboration and competition dynamics (Zhu et al., 2025a), role specialization, and emergent failure modes such as coordination collapse and dialogue drift (Cemri et al., 2025; Zhang et al., 2025b; Zhu et al., 2025a; Hyun et al., 2025; Tran et al., 2025; Zhang et al., 2025a). For single-agent systems, analyses have highlighted planning brittleness, grounding and tool-use errors, and hallucination-induced cascades (Ji et al., 2024; Sung et al., 2025; Ning et al., 2024). Compared to these studies, we focus on actionable debugging: moving from descriptive taxonomies to a two-stage detector that both identifies and helps fix root causes.

6 Conclusion

In summary, our work identifies error propagation as the central bottleneck to building robust LLM agents and introduces principled solutions. By tracing and correcting root-cause failures, *AgentDebug*-4B achieves substantial performance gains and establishes debugging as a foundation for agents that can continuously learn and evolve from their mistakes. While our work has limitations (see Sec 6), it paves the way toward more reliable and adaptive LLMs Agent.

Limitations

Our work is not without limitations. First, while *AgentErrorBench* covers three representative benchmarks (ALFWorld, GAIA, and WebShop), it remains limited in scale and domain diversity; extending to multimodal environments, longer-horizon tasks, or safety-critical applications (e.g., healthcare, finance) is an important direction for future work. Second, collecting sufficient data to train a dedicated debugging model would be prohibitively expensive in low-resource academic settings, given the costs of large-scale human annotation. To mitigate this, we instead designed a cost-efficient workflow that leverages prompt engineering with existing LLMs, though this approach may still fall short of the performance achievable with a fully trained, specialized model. Also, the current AgentDebugger-4B still has a lot of room to improve. Given the hardship of our benchmark and the small model's long-context limitation, this might be addressed through agentic observation condensing and the use of Reinforcement Learning methods in the future.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2023. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya G. Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *Preprint*, arXiv:2308.07201.

Jiaju Chen, Yuxuan Lu, Xiaojie Wang, Huimin Zeng, Jing Huang, Jiri Gesi, Ying Xu, Bingsheng Yao, and Dakuo Wang. 2025. Multi-agent-as-judge: Aligning llm-agent-based automated evaluation with multi-dimensional human evaluation. *Preprint*, arXiv:2507.21028.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *Preprint*, arXiv:2304.05128.

Yining Hong, Rui Sun, Bingxuan Li, Xingcheng Yao, Maxine Wu, Alexander Chien, Da Yin, Ying Nian Wu, Zhecan James Wang, and Kai-Wei Chang. 2025. Embodied web agents: Bridging physical-digital realms for integrated agent intelligence. *Preprint*, arXiv:2506.15677.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Jonathan Hyun, Nicholas R. Waytowich, and Boyuan Chen. 2025. CREW-WILDFIRE: Benchmarking agentic multi-agent collaborations at scale. *arXiv preprint arXiv:2507.05178*.

Zhenlan Ji, Daoyuan Wu, Pingchuan Ma, Zongjie Li, and Shuai Wang. 2024. Testing and understanding erroneous planning in llm agents through synthesized user inputs. *arXiv preprint arXiv:2404.17833*.

Bingxuan Li, Yiwei Wang, Jiuxiang Gu, Kai-Wei Chang, and Nanyun Peng. 2025. Metal: A multi-agent framework for chart generation with test-time scaling. *Preprint*, arXiv:2502.17651.

Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, Yuheng Cheng, Suyuchen Wang, Xiaoqiang Wang, Yuyu Luo, Haibo Jin, Peiyan Zhang, Ollie Liu, Jiaqi Chen, Huan Zhang, and 29 others. 2025. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems. *Preprint*, arXiv:2504.01990.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Kaiwen Ning, Jiachi Chen, Jingwen Zhang, Wei Li, Zexu Wang, Yuming Feng, Weizhe Zhang, and Zibin Zheng. 2024. Defining and detecting the defects of the large language model-based autonomous agents. *arXiv preprint arXiv:2412.18371*.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. 2026. Reasoningbank: Scaling agent self-evolving with reasoning memory. *Preprint*, arXiv:2509.25140.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Yoo Yeon Sung, Hannah Kim, and Dan Zhang. 2025. VeriLA: A human-centered evaluation framework for interpretable verification of llm agent failures. *arXiv preprint arXiv:2503.12651*.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. 2025. Multi-agent collaboration mechanisms: A survey of LLMs. *arXiv preprint arXiv:2501.06322*.

Junlin Wang, Jue WANG, Ben Athiwaratkun, Ce Zhang, and James Zou. 2025. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022b. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. 2025a. Agentracer: Who is inducing failure in the llm agentic systems? *Preprint*, arXiv:2509.03312.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. 2025b. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. 2025a. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. *arXiv preprint arXiv:2503.01935*.

Kunlun Zhu, Jiaxun Zhang, Ziheng Qi, Nuoxing Shang, Zijia Liu, Peixuan Han, Yue Su, Haofei Yu, and Jiaxuan You. 2025b. Safescientist: Toward risk-aware scientific discoveries by llm agents. *Preprint*, arXiv:2505.23559.

A Appendix

Ethical consideration

This work focuses on analyzing and improving the robustness of LLM-based agents. Our study does not involve human subjects, sensitive personal data, or information that could directly identify individuals. All experiments were conducted on publicly available benchmarks (ALFWorld, GAIA, and WebShop) under their respective licenses. While our findings aim to advance reliability and transparency in LLM agents, we acknowledge that more capable debugging frameworks could potentially be misused to strengthen harmful or malicious agents. We encourage responsible use of our methods, with applications limited to domains that respect safety, fairness, and accountability.

LLM Use Statement

LLM were used in two ways during this work. First, LLMs served as the backbone agents evaluated in our experiments (e.g., GPT-4.1, GPT-4o-mini, Qwen3-8B, Qwen3-Next-80B, Llama 3.1-70B). Second, we used LLMs to assist with writing tasks such as grammar error checking, and formatting, but all conceptual contributions, experimental design, and analyses were performed by the authors.

A.1 *AgentErrorTaxonomy*

See Fig. 6 and Table 2.

A.2 Efficiency and Token Cost Analysis

We compare the token consumption of *AgentDebug* with common test-time scaling baselines under matched success budgets.

AgentDebug. *AgentDebug* performs (i) one trajectory-level analysis pass and (ii) up to $N = 5$ targeted re-rollouts starting from the critical step only. This results in approximately $1.3\times$ the token cost of a single rollout.

Best-of-N and Tree-of-Thought. In contrast, Best-of-N and ToT require N or more full-trajectory rollouts, leading to $N\times$ or higher token consumption.

Conclusion. Under matched or lower token budgets, *AgentDebug* achieves superior performance by concentrating compute on the causal failure point rather than uniformly expanding the search space.

A.3 Positioning Against Agent Improvement and Failure Analysis Methods

Table 3 summarizes how *AgentDebug* differs from related agent improvement, prompt optimization, memory/context adaptation, and failure analysis methods. Existing self-correction methods such as Reflexion (Shinn et al., 2023), Self-Debug (Chen et al., 2023), and Self-Refine (Madaan et al., 2023) improve outputs through broad reflection or iterative feedback, but they do not explicitly ask which agent module failed, when the earliest critical error occurred, or whether a correction should restart from that causal point. ReAct-style agents (Yao et al., 2022b) define a reasoning–acting rollout pattern rather than a post-hoc debugging layer. Prompt and context optimization methods such as GEPA (Agrawal et al., 2026), ReasoningBank (Ouyang et al., 2026), and Agentic Context Engineering (Zhang et al., 2026) improve prompts, contexts, memory, or execution harnesses, but their primary object is optimization of future behavior rather than systematic error diagnosis with a modular taxonomy and benchmark. Multi-agent failure attribution work (Cemri et al., 2025; Zhang et al., 2025b,a) analyzes failures in collaborative systems and often asks which agent is responsible. AgenTracer (Zhang et al., 2025a), for example, focuses on identifying the agent that induces failure in multi-agent systems; it does not modularize the internal memory/reflection/planning/action stages of a single agent, nor does it build a benchmark and trained detector for single-agent critical error localization. Our setting instead targets single-agent trajectories and asks which module and step trigger the failure cascade.

To the best of our knowledge, *AgentDebug* is the first work to jointly provide (i) a systematic module-level taxonomy for single-agent failures, (ii) an expert-annotated benchmark for evaluating trajectory-level critical error detection, (iii) a training-free debugging loop that localizes the earliest critical error and re-executes from that point with targeted feedback, and (iv) a scalable proposer–reviewer data construction pipeline that yields supervised data for training an error detector. This focus makes our contribution complementary to agent improvement methods: rather than only improving the agent's policy, prompt, memory, or context, we provide an explicit diagnostic layer that explains *where*, *which module*, and *what type* of error caused the failure, then uses that diagnosis

Figure 6: **Error Taxonomy across all modules**: The hierarchical taxonomy organizes agent errors into five core modules—*Planning*, *Action*, *Reflection*, *Memory*, and *System*. Each outer ring category specifies representative error types, such as constraint ignorance or inefficient planning (Planning), parameter or format errors (Action), causal misattribution or outcome misinterpretation (Reflection), oversimplification or false memory (Memory), and tool execution or environment errors (System). This taxonomy provides a holistic view of where and how failures emerge within modular LLM agents.

Module	Error Type	Definition / Explanation
Memory	Over-simplification / Incomplete Summary	Summarizes past info too crudely, ignoring details; leads to flawed reasoning.
Memory	Hallucination (False Memory)	Recalls events or states that never happened, filling missing gaps with fabricated info.
Memory	Retrieval Failure	Relevant info exists but is not retrieved when needed.
Reflection	Progress Misassessment	Incorrectly evaluates progress (too optimistic, too pessimistic, or misses completion).
Reflection	Outcome Misinterpretation	Executes an action but misreads the immediate result or environment feedback.
Reflection	Causal Misattribution	Correctly notes failure but blames the wrong cause, misguiding subsequent plans.
Reflection	Hallucination	Reflects on events/results that never occurred.
Planning	Constraint Ignorance	Ignores limits (time, budget, space, etc.) when forming plans.
Planning	Impossible Action	Plans a step that is physically/logically impossible given current preconditions.
Planning	Inefficient Planning	Plan is overly long or illogical; wastes steps and risks hitting limits.
Action	Planning–Action Disconnect	Chosen actions do not align with the stated plan intent.
Action	Format Error	Produces syntactically invalid actions.
Action	Parameter Error	Generates unreasonable or malformed parameters.
System	Step Limit Exhaustion	Fails due to reaching the maximum step cap despite reasonable behavior.
System	Tool Execution Error	External tool/API misbehaves or errors, causing downstream failures.
System	LLM Limit	Fails due to API/model constraints (e.g., timeouts, token limits).
System	Environment Error	Simulator/environment breaks expected rules (bug/crash/network), not agent's fault.

Table 2: *AgentErrorTaxonomy* Definitions

to guide self-improvement.

A.4 AgentDebug-Traj-2k: Error Detector SFT Data Construction

To train an error detector model via supervised fine-tuning (SFT), we construct a large-scale dataset of agent trajectories annotated with critical error diagnoses. This appendix describes the full pipeline: trajectory collection, multi-proposer error analysis, meta-reviewer verification, and final data formatting.

A.4.1 Pipeline Overview

Our data construction follows a four-stage pipeline, as shown in Figure 7:

1. **Trajectory Collection (Rollout).** We collect agent trajectories by running a cheap LLM (Llama-3.3-70B-Instruct) across three environments: ALFWorld, WebShop, and GAIA. The rollout model intentionally uses a weaker model to maximize failure trajectories, which are the primary training signal for the error detector.

2. **Multi-Proposer Error Detection.** Three proposer models independently analyze each failed trajectory and produce a critical error diagnosis. Each proposer makes a single API call per trajectory (Phase 2 only, no step-level Phase 1 analysis).

3. **Meta-Reviewer Verification.** A stronger verifier model reads the full trajectory alongside all three proposer analyses and produces the final diagnosis. The verifier acts as a meta-reviewer: it synthesizes proposer insights, resolves disagreements, corrects factual errors, and outputs a structured critical error annotation.

4. **SFT Data Export.** The final dataset pairs each trajectory (as the prompt) with the verifier's diagnosis (as the response), along with full provenance metadata.

A.4.2 Trajectory Collection

Trajectories are collected using the modular rollout framework described in Section 3.1. We use two inference providers to increase throughput and avoid rate-limit bottlenecks:

- **Together AI**: `meta-llama/Llama-3.3-70B-Instruct-Turbo`, concurrency 4
- **FreeInference**: `llama-3.3-70b-instruct` (OpenAI-compatible endpoint), concurrency 2

For each environment, we run rollouts with disjoint task pool slices (controlled by `-seed` and `-pool_offset`) to ensure no overlap between runs. Temperature is set to 0.7 to encourage diverse failure modes. Table 4 summarizes the collection.

Environment	Total Episodes	Failed	Success Rate
ALFWorld	850	810	4.7%
WebShop	850	850	0.0%
GAIA	466	461	1.1%
Total	**2,166**	**2,121**	2.1%

Table 4: Trajectory collection summary. The low success rate is by design: we intentionally use a weaker rollout model to maximize failure trajectories for error detection training.

A.4.3 Multi-Proposer Error Detection

Each failed trajectory is analyzed by three proposer models, each making a single LLM call to identify the critical error:

Proposer	Provider	Model
Proposer 1	OpenAI	`gpt-4o-mini`
Proposer 2	Anthropic	`claude-haiku-4.5`
Proposer 3	Google	`gemini-2.5-flash`

Table 5: Proposer model configuration. All proposers use temperature 0 for deterministic outputs.

Each proposer receives the full trajectory, the *AgentErrorTaxonomy* error taxonomy definitions, and is instructed to identify the earliest critical error. The proposer prompt is:

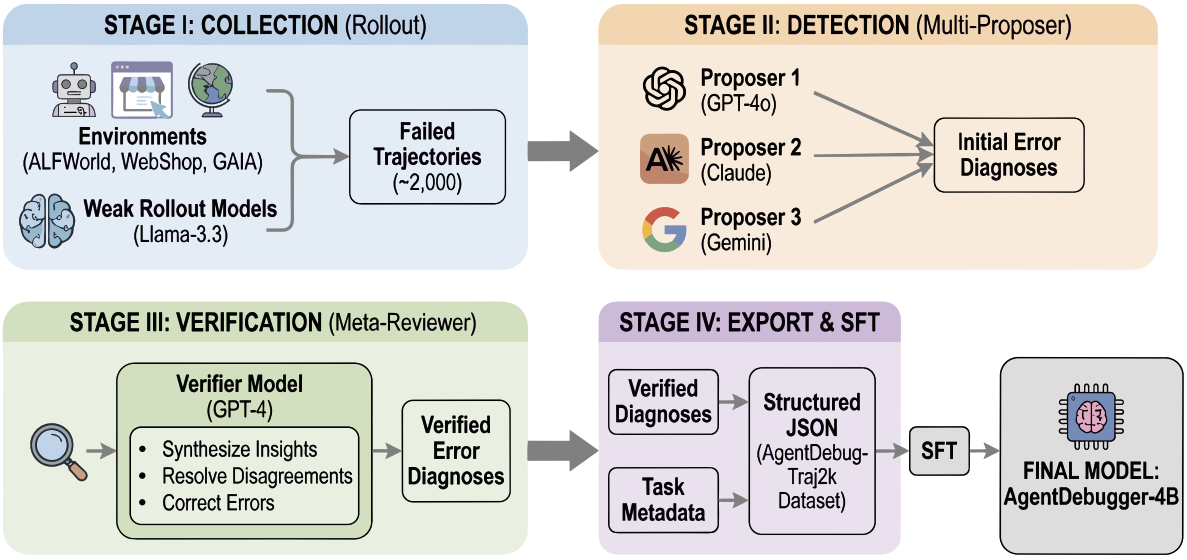


Figure 7: **AgentDebug-Traj2k Data Curation Pipeline.** We first collect failed agent trajectories by rolling out a relatively cheap LLM across ALFWorld, WebShop, and GAIA. Each failed trajectory is then independently analyzed by three proposer models, whose diagnoses are synthesized and verified by a stronger meta-reviewer to produce the final critical error annotation. The resulting dataset pairs each trajectory with the verified diagnosis and provenance metadata, forming supervised fine-tuning data for trajectory-level error detection.

Proposer Prompt

```
Analyze this failed agent trajectory and
identify
the earliest critical error.

Environment: {environment}
Task ID: {task_id}
Task Success: {success}

TRAJECTORY:
{full_transcript}

ERROR TAXONOMY:
{error_definitions}

Identify the critical error -- the earliest
step
where, if corrected, the task would likely
still
have been recoverable.

REQUIRED OUTPUT FORMAT (JSON):
{
  "critical_step": <step number>,
  "critical_module":
  "<memory|reflection|planning|
    action|system|others>",
  "error_type": "<specific error type>",
  "root_cause": "...",
  "evidence": "...",
  "correction_guidance": "...",
  "cascading_effects": [{"step": <n>,
    "impact": "..."}]
}
```

A.4.4 Meta-Reviewer Verification

The verifier model (`gpt-4.1`, temperature 0) receives the full trajectory, the *AgentErrorTaxonomy* taxonomy, and all proposer analyses. It acts as a meta-reviewer: synthesizing insights, resolving disagreements, and producing the final diagnosis.

Verifier (Meta-Reviewer) Prompt

```
You are a meta-reviewer for agent error
analysis.
Multiple reviewers have independently analyzed
a
failed agent trajectory. Your job is to
produce
the final diagnosis by synthesizing their
insights,
resolving disagreements, correcting factual
mistakes, and identifying the earliest
critical
error.

TASK CONTEXT:
Environment: {environment}
Task ID: {task_id}
Task Success: {success}

TRAJECTORY:
{full_transcript}

ERROR TAXONOMY:
{error_definitions}

REVIEWER ANALYSES:
=== Analysis 1 (by openai:gpt-4o-mini) ===
{proposer_1_json}

=== Analysis 2 (by anthropic:claude-haiku-4.5)
===
{proposer_2_json}
```

```
=== Analysis 3 (by gemini:gemini-2.5-flash)
===
{proposer_3_json}

INSTRUCTIONS:
1. Read and understand the trajectory
yourself.
2. Refer to reviewer analyses -- note where
they
   agree and disagree.
3. Form your own judgment and produce the
final
   diagnosis.
4. The critical error is the earliest step
where,
   if corrected, the task would likely still
   have
   been recoverable.

REQUIRED OUTPUT FORMAT (JSON):
{
  "critical_step": <step number>,
  "critical_module":
  "<memory|reflection|planning|
    action|system|others>",
  "error_type": "<specific error type>",
  "root_cause": "...",
  "evidence": "...",
  "correction_guidance": "...",
  "cascading_effects": [{"step": <n>,
    "impact": "..."}],
  "meta_review": {
    "reviewer_agreements": ["..."],
    "reviewer_disagreements": ["..."],
    "final_rationale": "..."
  }
}
```

A.4.5 SFT Data Format

Each SFT training example consists of:

- **Prompt**: The full trajectory formatted as a sequence of `[user]` / `[assistant]` turns, prefixed with task metadata and instructions to identify the critical error.
- **Response**: The verifier's final critical error diagnosis as a JSON object (excluding the `meta_review` field, which is stored separately as metadata).
- **Metadata**: Environment, task ID, per-proposer analysis summaries, and the verifier's meta-review reasoning.

A.4.6 Dataset Statistics

Table 6 summarizes the final SFT dataset.

	ALFWorld	WebShop	GAIA
SFT rows	821	711	461
Has critical error	100%	100%	100%
Has meta_review	100%	100%	100%

Table 6: SFT dataset summary. Total: 1,993 training examples across three environments.

Critical Module Distribution. Table 7 shows which modules are most frequently identified as the source of the critical error. Action errors dominate across all environments (72–82%), followed by planning errors.

Module	ALFWorld	WebShop	GAIA
Action	626 (76.2%)	580 (81.6%)	335 (72.7%)
Planning	71 (8.6%)	65 (9.1%)	117 (25.4%)
Reflection	60 (7.3%)	53 (7.5%)	4 (0.9%)
System	54 (6.6%)	8 (1.1%)	3 (0.7%)
Memory	10 (1.2%)	5 (0.7%)	2 (0.4%)

Table 7: Distribution of critical error modules. Action errors are most frequent, while GAIA shows a notably higher proportion of planning errors due to the complexity of tool-use coordination.

Error Type Distribution. Table 8 shows the top error types per environment.

Environment	Error Type	Count (%)
ALFWorld	invalid_action	314 (38.2%)
	misalignment	160 (19.5%)
	format_error	152 (18.5%)
	outcome_misinterpretation	46 (5.6%)
	impossible_action	42 (5.1%)
WebShop	misalignment	312 (43.9%)
	parameter_error	228 (32.1%)
	outcome_misinterpretation	44 (6.2%)
	inefficient_plan	35 (4.9%)
	invalid_action	22 (3.1%)
GAIA	parameter_error	164 (35.6%)
	misalignment	141 (30.6%)
	constraint_ignorance	55 (11.9%)
	impossible_action	50 (10.8%)
	format_error	25 (5.4%)

Table 8: Top 5 error types per environment. ALFWorld is dominated by invalid actions (format/syntax issues in embodied commands); WebShop by misalignment and parameter errors (wrong product selections); GAIA by parameter errors and misalignment (incorrect tool parameters).

A.4.7 Implementation Details

- **Rollout model**: Llama-3.3-70B-Instruct-Turbo, temperature 0.7, max 30 steps per episode.
- **Proposer models**: gpt-4o-mini, claude-haiku-4.5, gemini-2.5-flash, all at temperature 0.
- **Verifier model**: gpt-4.1, temperature 0.
- **Pipeline concurrency**: 5 trajectories processed in parallel; each trajectory triggers 3 proposer calls + 1 verifier call = 4 API calls total.
- **Retry policy**: Each proposer retries up to 3 times with exponential backoff (2s, 4s, 8s) on API errors.
- **Cost**: Approximately $0.02 per trajectory (4 API calls to lightweight/mid-tier models), totaling roughly $40 for the full 1,993-row dataset.
- **Pool slicing**: Tasks are shuffled with a fixed seed and partitioned via `-pool_offset` to ensure disjoint task pools across runs.

A.4.8 Intended Use

The SFT dataset is designed to train a specialized error detector model that, given a failed agent trajectory, can:

1. Identify the earliest critical error step and module.
2. Classify the error type according to the *AgentErrorTaxonomy* taxonomy.
3. Explain the root cause with evidence from the trajectory.
4. Provide actionable correction guidance.

This trained detector can then be deployed as the analysis backbone in the *AgentDebug* framework (replacing the current prompt-based GPT-4.1 detector), enabling faster and cheaper iterative debugging at inference time.

A.4.9 SFT Training Details

We fine-tune a lightweight open-source model as the error detector using parameter-efficient adaptation on the curated SFT dataset described above.

Base Model. We use `Qwen3-4B-Instruct-2507` as the base model, a 4.15B-parameter instruction-tuned language model with 36 transformer layers, 32 attention heads (8 key-value heads), and a hidden dimension of 2,560. The model supports a maximum context length of 262,144 tokens and uses a vocabulary of 151,936 tokens. We load the model in `bfloat16` precision with scaled dot-product attention (`sdpa`).

LoRA Configuration. We apply Low-Rank Adaptation (LoRA) (Hu et al., 2022) to all linear projection layers in each transformer block, targeting: `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. We use rank r=64, scaling factor α=128, and dropout rate 0.05. This yields 132M trainable parameters out of 4.15B total (3.18%).

Training Hyperparameters. We train separate LoRA adapters per environment as well as a unified mixed-environment model. Training uses AdamW optimizer with a cosine learning rate schedule, peak learning rate 2×10^{-4}, warmup ratio 0.05, weight decay 0.01, and effective batch size 8 (per-device batch size 1 with gradient accumulation over 8 steps). We enable gradient checkpointing to support long sequences. Table 9 summarizes the per-environment training configurations.

Long-Sequence Handling. Agent trajectories can be extremely long—GAIA trajectories reach up to 1.6M tokens before formatting, and WebShop

trajectories up to 74K tokens. To handle sequences beyond the model's context window without out-of-memory errors, we implement two strategies:

1. **Chunked loss computation**: Instead of materializing the full $[B, S, V]$ logits tensor (where $V{=}151,936$), we compute cross-entropy loss in chunks of 256 tokens, accumulating the loss over the sequence. This reduces peak GPU memory from $\mathcal{O}(S \cdot V)$ to $\mathcal{O}(C \cdot V)$ where $C{=}256$.

2. **Trajectory compression at inference**: For evaluation, trajectories exceeding the model's context window are compressed by deduplicating accumulated observation history (each step in the trajectory repeats all prior observations) and truncating long tool outputs while preserving planning, memory, and reflection modules intact.

Hardware. All training is conducted on NVIDIA H100 80GB HBM3 GPUs. Single-environment models use 1–2 GPUs with `device_map=auto` for automatic model parallelism; the mixed model uses 2 GPUs.

Training Curves. Figure 8 shows the training loss curves across the three per-environment SFT runs. All models converge smoothly, with ALFWorld and WebShop exhibiting the fastest convergence due to more structured trajectory formats, while GAIA shows higher initial loss due to the diversity and complexity of web-based tool-use trajectories.

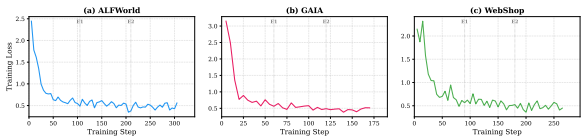


Figure 8: SFT training loss curves for the three per-environment error detector models. Dashed vertical lines indicate epoch boundaries. All models are trained for 3 epochs with cosine learning rate decay.

A.5 Human Annotation Details, Instructions, and Consent

Annotators. All annotations in *AgentErrorBench* were performed by the authors and graduate students affiliated with the project. All annotators are highly educated researchers with prior experience in large language models and LLM-based agent systems. No crowdsourcing platform or external annotators were used. Inter-annotator agreement, measured using Cohen's κ, reached $\kappa = 0.55$ across modules, indicating substantial agreement.

Annotation Instructions (D1). Annotators were provided with detailed written instructions describing the annotation task and expected standards. These instructions included: (i) the full definition of the proposed Agent Error Taxonomy (AET); (ii) illustrative examples for each error category; (iii) multiple fully annotated example trajectories with step-level error labels and identified root causes; and (iv) explicit guidelines emphasizing causal reasoning and the identification of minimal root-cause errors rather than surface-level mistakes. The instructions clearly stated that annotations would be used solely for academic research and benchmark construction.

Recruitment and Compensation (D2). Annotators were recruited internally from the author team and collaborating graduate students. Since all annotators are co-authors or student researchers contributing as part of their academic research activities, no monetary compensation was involved. As a result, concerns regarding inadequate payment or demographic fairness do not apply in this setting.

Data Consent and Ethics (D3). All annotated data consist exclusively of agent-generated trajectories from publicly available benchmarks (ALFWorld, WebShop, and GAIA). No personal, sensitive, or user-generated data were involved. Annotators were informed of how the annotated data would be used, stored, and released as part of a research benchmark. Because the data do not involve human subjects beyond expert annotators acting in a research capacity, and no personal data were collected, the annotation process complies with standard academic research ethics requirements and does not require additional informed consent procedures.

A.6 AgentDebug Algorithm

See Algorithm 1 below.

A.7 Error Propagation case study

See Fig. 9. During annotation, annotators labeled whether an error was causally dependent on a previous step or independent.

On ALFWorld:

Algorithm 1: *AgentDebug* Inference Procedure

Input: Trajectory $\tau = \{(s_t, a_t)\}_{t=1}^{T}$; Error taxonomy $\mathcal{E}_{\text{AET}}$; Critical-error criterion $\mathcal{C}_{\text{crit}}$; Max Iterations I
Output: Corrected trajectory τ^* **or** `Failure`

```
/* Stage 1: Fine-grained Analysis with AET                                  */
for t ← 1 to T do
    foreach m ∈ {mem, plan, refl, act} do
        e_t^m ← MapToAET(s_t, a_t, m, E_AET)          // Assign error type per module
if Eval(τ) = 1 then
    return τ                                          // Trajectory already successful
/* Stage 2: Critical Error Detection via LLM (no rollout/counterfactuals)   */
C ← {τ, {e_t^m}_{t,m}, E_AET, C_crit}                 // Pack analysis, taxonomy & criterion
(T*, M*, Z*, φ^(0)) ← DetectCriticalErrors(C)         // LLM (fine-grained prompt)
if T* = ∅ then
    return Failure                                    // No critical error found
t* ← min(T*)                                          // Earliest critical step
// T*: set of critical steps; M*: modules per critical step; Z*: error-types per
   critical step
/* Stage 3: Iterative Debugging with Targeted Feedback                      */
τ^(0) ← τ
for k ← 1 to I do
    τ^(k) ← ReRollout(τ^(k−1), t*, φ^(k−1))           // Re-execute from t* with feedback
    if Eval(τ^(k)) = 1 then
        return τ^(k)
    φ^(k) ← UpdateFeedback(τ^(k), φ^(k−1))            // Refine guidance if still failing
return Failure                                        // Max Iterations Reached
```

- 62% of mid- and late-stage errors are marked as dependent on an earlier mistake.
- If the first error occurs before step 6, the probability of observing three or more downstream errors increases from 27% to 61%.
- If the first error occurs after step 10, the cascade rate drops to 19%.

Planning errors also contribute heavily, with constraint ignorance or infeasible strategies compounding as the agent attempts to execute them. By contrast, action-level errors are more visible and sometimes recoverable, though malformed outputs or missing parameters can still derail execution. System-level issues such as tool crashes or step-limit exhaustion act as immediate termination points rather than cascades.

These results confirm that error propagation is not merely temporal accumulation but reflects causal dependency.

A.8 Error Analysis of AgentDebug

We provide a detailed breakdown of where *AgentDebug*'s error detection may fails, complementing the summary in Section 4.5. Figure 10 provides a visual overview of the key findings.

A.8.1 Failure Mode Breakdown

Table 10 shows the failure mode distribution across five backbone models. Step localization errors account for 75–85% of all failures. Once the correct step is identified, module classification is relatively accurate (12–17% error rate), and error-type misclassification is rare ($\leq$3%).

A.8.2 Reasoning Models as Backbone Detectors

We evaluate two reasoning models—Kimi-K2 and MiniMax-M2.5—alongside standard models as backbone detectors for *AgentDebug*.

Table 12 provides per-benchmark breakdowns (Step / S+M / All):

Key findings: (1) Reasoning models improve step localization—MiniMax-M2.5 (24.7%) matches GPT-4.1 (24.0%) at the step level. (2) GPT-4.1 retains a clear advantage in full detection (All=11.0% vs. 6.7%). (3) MiniMax-M2.5 is the strongest non-GPT-4.1 model, achieving 8.0% All-Correct on ALFWorld vs. 0–3% for other non-GPT models.

A.8.3 Per-Module Detection Difficulty

Table 13 shows All-Correct accuracy broken down by ground-truth module. Memory and system errors are the hardest to detect across all models, with system errors achieving 0% accuracy universally. Action errors are comparatively easier due to their local, observable nature.

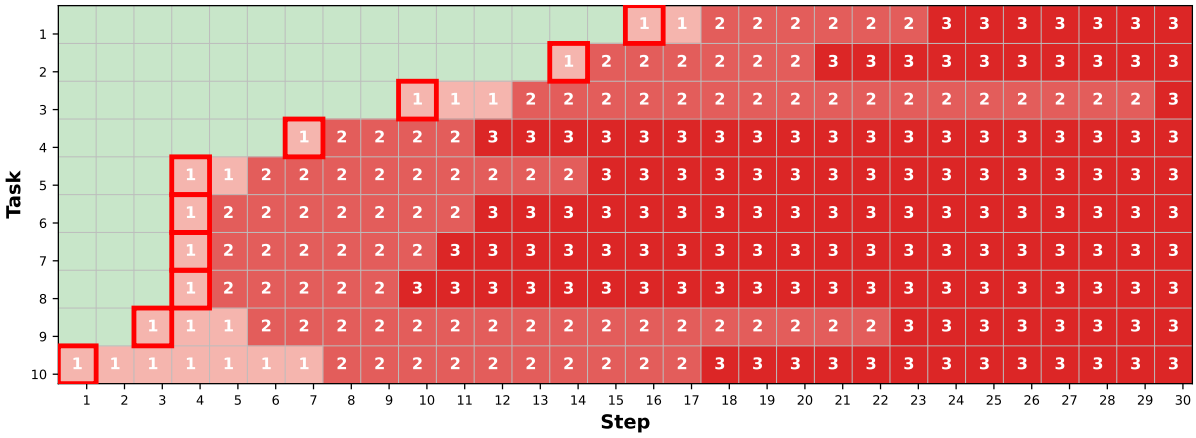


Figure 9: ***Illustration of Error Propagation***: Darker shading indicating compounding failures and highlighting how initial mistakes amplify downstream breakdowns.

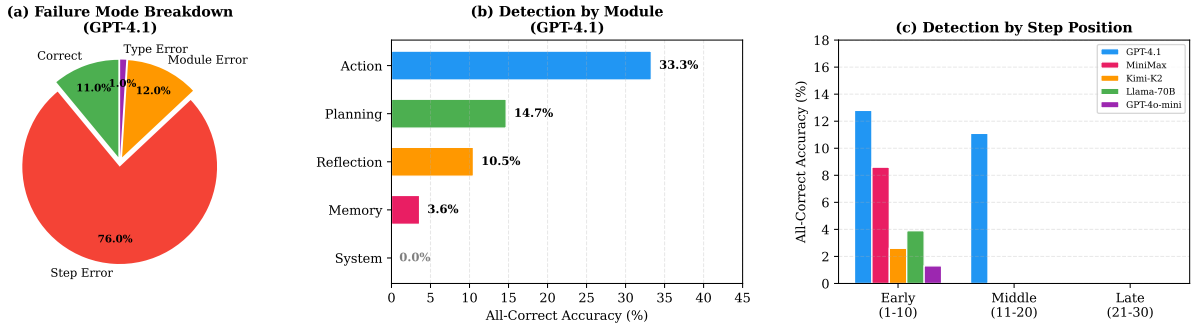


Figure 10: Error analysis of *AgentDebug*. (a) Failure mode breakdown for GPT-4.1: step localization errors dominate at 76%. (b) Per-module detection difficulty: action errors are easiest to detect while system and memory errors remain challenging. (c) Detection accuracy by step position across all backbone models: accuracy degrades sharply with step position, and errors after step 20 are never correctly identified.

A.8.4 Per-Error-Type Detection Difficulty

Table 14 shows GPT-4.1's detection accuracy by error type. Local, observable errors (impossible_action, misalignment) are significantly easier to detect than errors requiring global trajectory reasoning (inefficient_plan, hallucination).

A.8.5 Detection Accuracy by Step Position

Table 15 shows how detection accuracy varies with the ground-truth critical step position.

A.8.6 Summary of Findings

These analyses reveal several actionable insights: (1) **Step localization is the primary bottleneck**—improving step-level accuracy would yield the largest gains in end-to-end detection. (2) **Memory and system errors** are systematically harder to detect, suggesting these modules require specialized detection strategies. (3) **Global reasoning errors** (e.g., inefficient_plan, hallucination) remain largely undetectable even by the strongest models, pointing to fundamental limitations in current trajectory-level analysis. (4) **Late-stage errors** (steps 21+) are never correctly identified, likely because long trajectories dilute the signal for the critical step. (5) **Reasoning models** partially close the gap with GPT-4.1 on step localization but do not yet match its fine-grained classification ability. Planning (33%) and memory (29%) errors together account for over 60% of all detection failures, consistent across all backbone models.

A.9 AgentDebug Example

See Fig. 11.

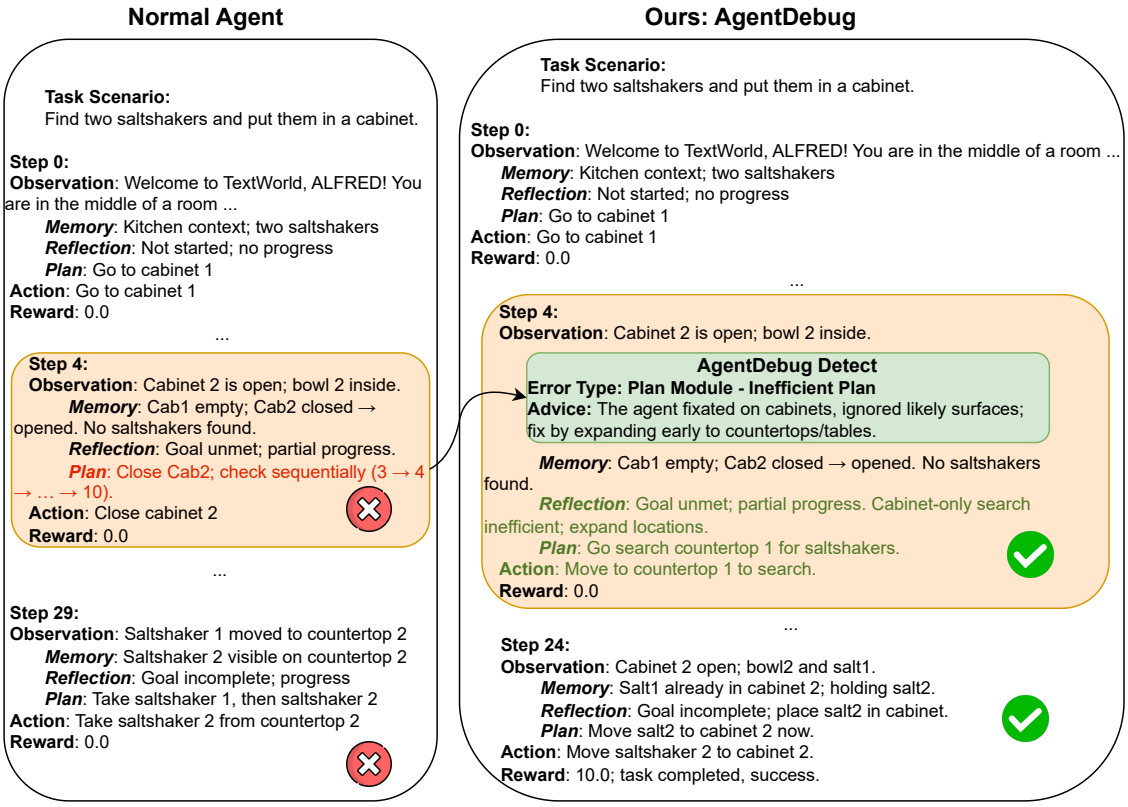


Figure 11: **AgentDebug example on ALFWorld:** Comparison between a normal agent (left) and our method AgentDebug (right). AgentDebug successfully pinpoints the critical error in the *Plan* module (inefficient cabinet-only search) and provides actionable feedback (expand to countertops/tables). This guidance allows the agent to recover from failure and complete the task successfully.

Method family	Representative work	Primary objective	Difference from *AgentDebug*
Reasoning–acting agents	ReAct (Yao et al., 2022b)	Interleave reasoning traces and environment actions during rollout.	Defines how an agent acts, but does not provide a modular failure taxonomy, critical-step benchmark, or targeted re-execution from the earliest root-cause error.
Undirected self-reflection and refinement	Reflexion (Shinn et al., 2023); Self-Debug (Chen et al., 2023); Self-Refine (Madaan et al., 2023)	Improve future attempts by generating verbal feedback or revised outputs.	Reflection is broad and often task-level. *AgentDebug* explicitly detects when and where the trajectory failed, maps the failure to memory/reflection/planning/action/system modules, and grounds feedback in error types.
Search and test-time scaling	Tree-of-Thought (Yao et al., 2023); Best-of-N	Expand candidate trajectories or thoughts with additional inference-time compute.	Allocates compute broadly across candidate rollouts. *AgentDebug* concentrates compute on the localized critical step, reducing unnecessary full-trajectory retries.
Prompt/text optimization	GEPA (Agrawal et al., 2026)	Optimize prompts or text artifacts from feedback and comparisons.	Optimizes artifacts rather than building an agent-error benchmark or detecting step/module/error-type failures in trajectories. *AgentDebug* is error-focused and can be used as a diagnostic signal for such optimizers.
Memory/context self-improvement	ReasoningBank (Ouyang et al., 2026); Agentic Context Engineering (Zhang et al., 2026)	Evolve memory or context from experience to improve future behavior.	Improves context or memory state, but does not center a complete module-level taxonomy or benchmark for critical error detection. *AgentDebug* diagnoses the causal failure first, then uses the diagnosis for recovery or SFT.
Multi-agent failure attribution	Why Do Multi-Agent LLM Systems Fail? (Cemri et al., 2025); Which Agent Causes Task Failures and When? (Zhang et al., 2025b); AgenTracer (Zhang et al., 2025a)	Analyze or attribute failures across multiple interacting agents; AgenTracer identifies which agent induces failure.	Focuses on multi-agent responsibility or system-level collaboration failures. *AgentDebug* instead targets single-agent trajectories, modularizes internal memory/reflection/planning/action/system failures, localizes the earliest critical step, and supports benchmark evaluation plus detector training.
AgentDebug	This work	Detect earliest critical errors, provide taxonomy-grounded feedback, curate SFT data, and train an error detector.	Integrates taxonomy, benchmark, training-free debugging, scalable supervision, and a trained detector for agentic error diagnosis and self-improvement.

Table 3: Positioning of *AgentDebug* relative to agent improvement, prompt/context optimization, and failure analysis methods.

	ALFWorld	GAIA	WebShop	Mixed
Training samples	821	461	711	1,993
Epochs	3	3	3	1
Max seq. length	65,536	131,072	131,072	131,072
Total steps	∼308	∼174	∼267	∼250
GPUs	1×H100	2×H100	2×H100	2×H100
Training time	∼5.5h	∼5.5h	∼24h	∼14h

Table 9: SFT training configuration per environment. All runs use the same base model (Qwen3-4B-Instruct-2507) and LoRA hyperparameters. The mixed model is trained on the concatenation of all three environment datasets for a single epoch.

Model	Correct	Step Err	Module Err	Type Err
GPT-4.1	11.0%	76.0%	12.0%	1.0%
MiniMax-M2.5	6.7%	75.3%	16.9%	1.1%
Kimi-K2	2.0%	79.0%	16.0%	3.0%
Llama-3.3-70B	3.0%	81.8%	12.1%	3.0%
GPT-4o-mini	1.0%	85.0%	14.0%	0.0%

Table 10: Failure mode breakdown (%) across backbone detector models. "Step Err" means the predicted critical step is incorrect; "Module Err" means the step is correct but the module is wrong; "Type Err" means step and module are correct but the error type is wrong.

Model	Type	Step	S+M	All
GPT-4.1	Proprietary	24.0	12.0	11.0
MiniMax-M2.5	Reasoning	24.7	7.9	6.7
Kimi-K2	Reasoning	21.0	5.0	2.0
Llama-3.3-70B	Open-source	18.2	6.1	3.0
GPT-4o-mini	Proprietary	15.0	1.0	1.0

Table 11: Detection accuracy (%) aggregated across all benchmarks. S+M = Step + Module correct. All = Step + Module + Type correct.

Model	ALFWorld	GAIA	WebShop
GPT-4.1	26 / 14 / 14	20 / 16 / 16	24 / 4 / 0
MiniMax-M2.5	22 / 8 / 8	21 / 13 / 8	40 / 0 / 0
Kimi-K2	16 / 2 / 2	32 / 12 / 4	20 / 4 / 0
Llama-3.3-70B	8 / 4 / 0	25 / 8 / 4	32 / 8 / 8
GPT-4o-mini	12 / 0 / 0	20 / 4 / 4	16 / 0 / 0

Table 12: Per-benchmark detection accuracy (%): Step / Step+Module / All-Correct. Reasoning models (MiniMax-M2.5, Kimi-K2) improve step localization but GPT-4.1 retains the strongest full detection performance.

GT Module	N	GPT-4.1	MiniMax	K2	Llama
Action	∼9	33.3	22.2	0.0	0.0
Planning	∼34	14.7	10.7	0.0	2.9
Reflection	∼19	10.5	5.9	10.5	0.0
Memory	∼28	3.6	0.0	0.0	7.1
System	∼10	0.0	0.0	0.0	0.0

Table 13: All-Correct accuracy (%) by ground-truth module. Action errors are easiest to detect; memory and system errors are hardest across all models.

GT Error Type	N	All (%)
impossible_action	9	44.4
misalignment	5	40.0
constraint_ignorance	4	25.0
progress_misjudge	12	16.7
over_simplification	17	5.9
hallucination	10	0.0
inefficient_plan	21	0.0
environment_error	6	0.0

Table 14: Per-error-type All-Correct accuracy for GPT-4.1. Local errors (impossible_action, misalignment) are substantially easier to detect than global reasoning errors (inefficient_plan, hallucination).

Position	GPT-4.1	MiniMax	K2	Llama	4o-mini
Early (1–10)	12.8	8.6	2.6	3.9	1.3
Middle (11–20)	11.1	0.0	0.0	0.0	0.0
Late (21–30)	0.0	0.0	0.0	0.0	0.0

Table 15: All-Correct accuracy (%) by ground-truth step position. Accuracy degrades sharply with step position; errors occurring after step 20 are never correctly identified by any model.

A.10 Failure Analysis Across Representative LLMs Agent Benchmarks

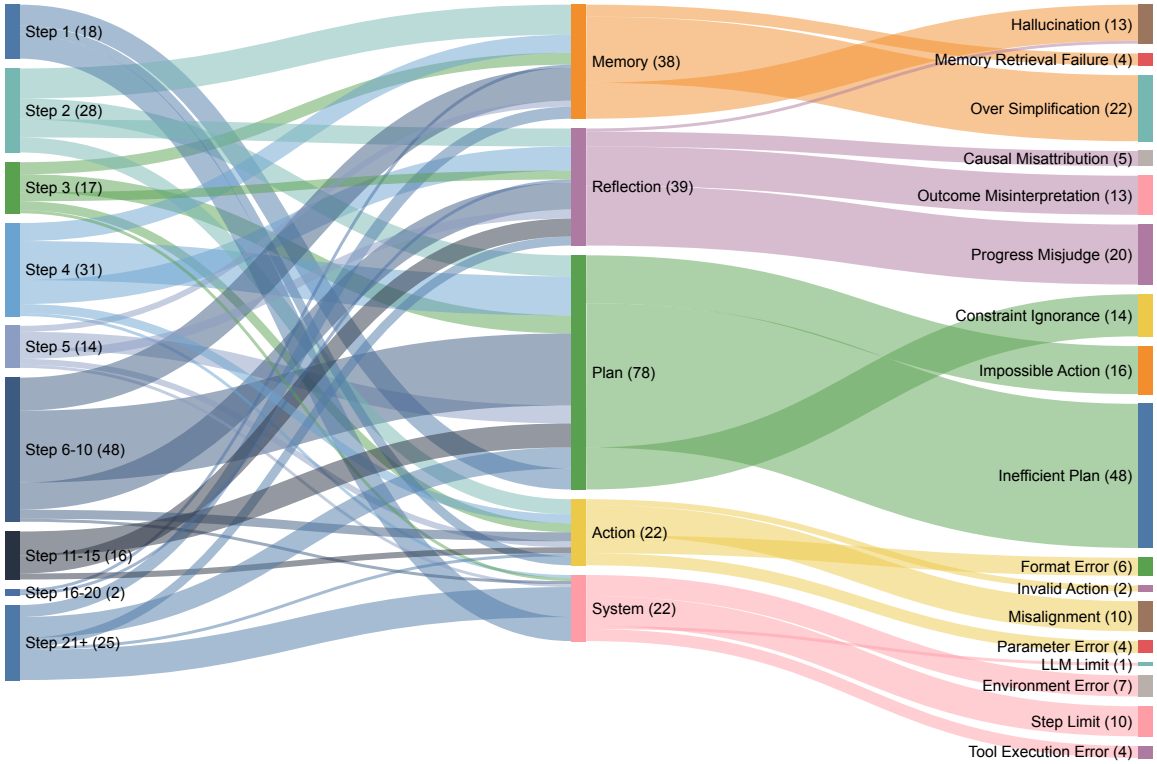


Figure 12: Distribution of failure cases in LLM agents on the *AgentErrorBench*

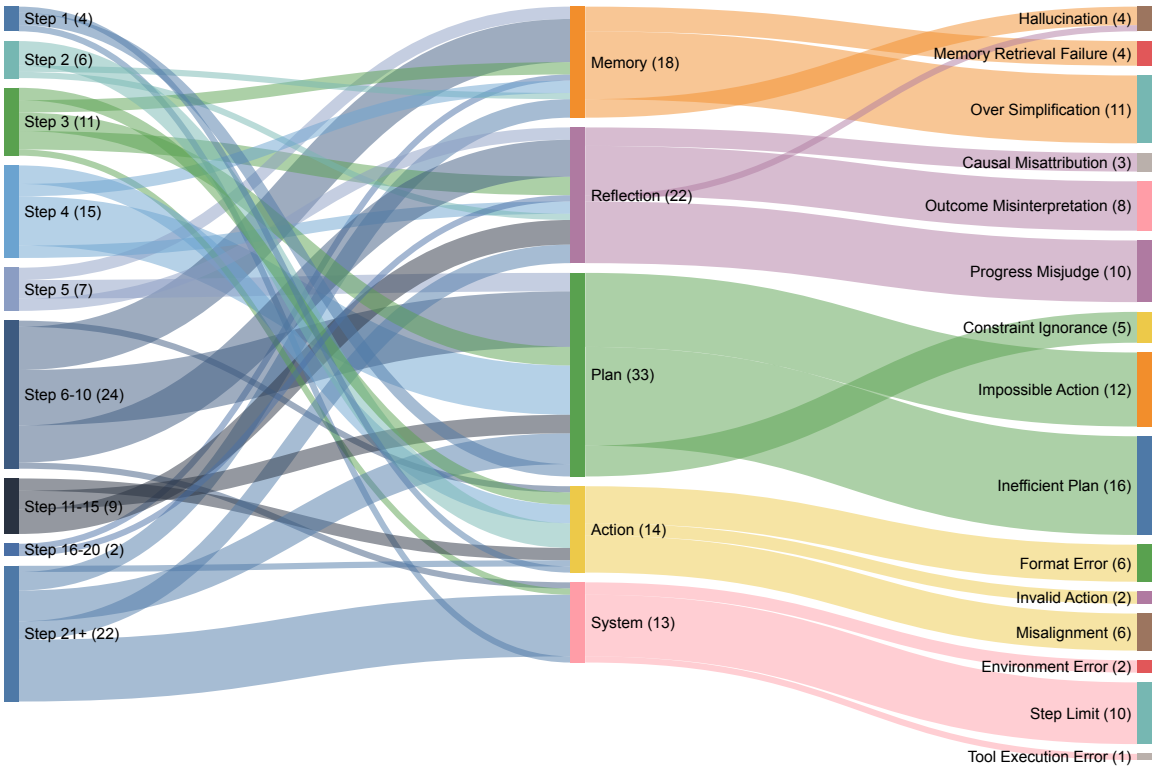


Figure 13: Distribution of failure cases in LLM agents on the Alfworld benchmark.

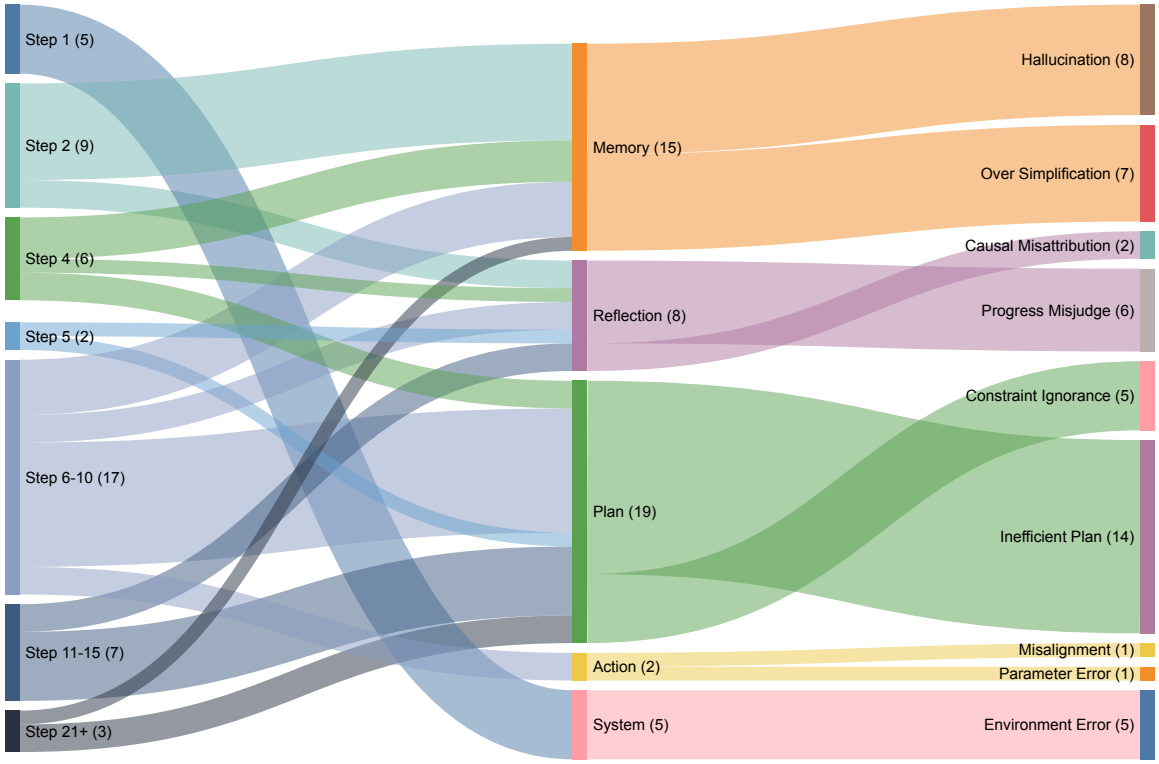


Figure 14: Distribution of failure cases in LLM agents on the Webshop benchmark.

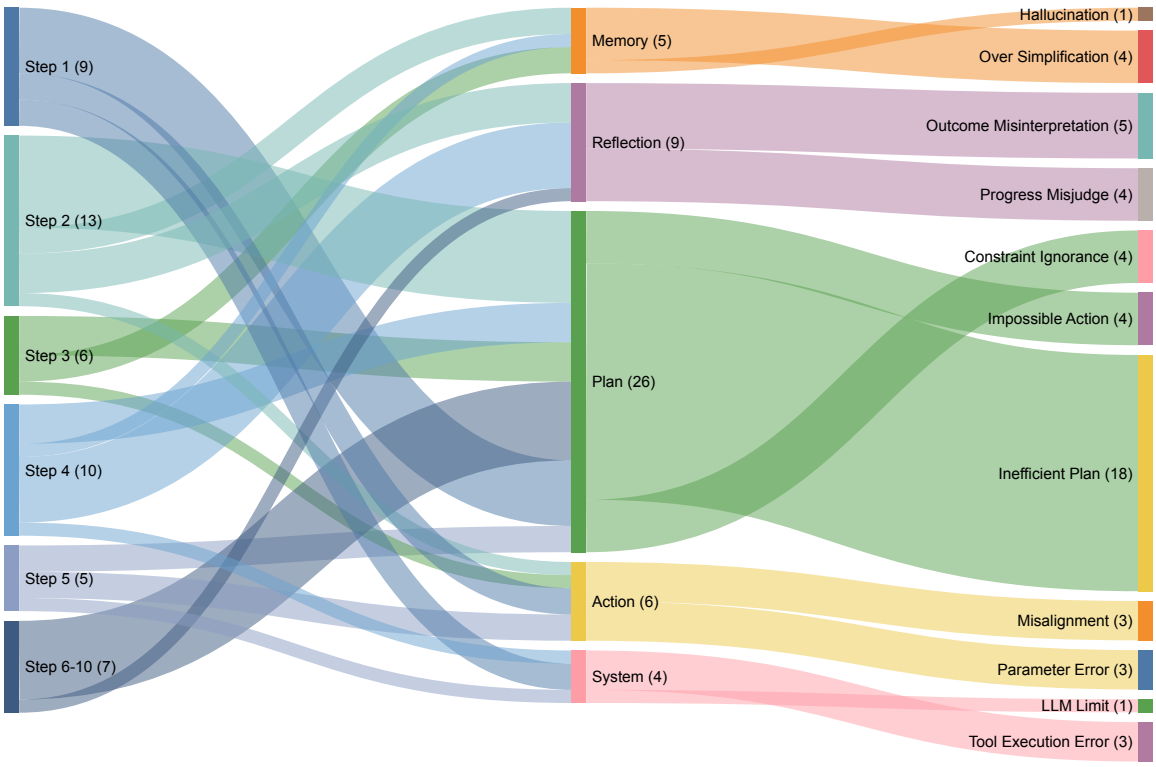


Figure 15: Distribution of failure cases in LLM agents on the GAIA benchmark.

A.11 Prompt

Stage1-Detector Prompt

Prompt Overview: Detect whether a specific module output (Memory, Reflection, Planning, Action, or System) contains a defined error, and justify the decision with evidence and reasoning.

Prompt Content (Verbatim):

```
You are an expert at detecting errors in agent trajectories.

TASK: {task_description}
ENVIRONMENT: {environment}
CURRENT STEP: {step_num}

INPUT AND CONTEXT:
{context}

MODULE TO ANALYZE: {module_name}
MODULE OUTPUT (What the agent produced for this module):
{module_content if module_content else "No content found for this
module"}

ENVIRONMENT RESPONSE AFTER THIS STEP:
{env_response if env_response else "No response"}

{error_definitions}

Based on the SPECIFIC error definitions provided above:
1. Identify if there is an error in this module by checking if the
output
matches any error definition
2. If yes, specify which exact error type based on the definitions
3. Provide evidence from the content that directly relates to the
definition
4. Explain your reasoning showing how it matches the specific
definition criteria

SPECIAL RULES:
- The "Current Step Input" contains the full user message including
  conversation history
- Evaluation criteria for each module:
  * Memory: Should correctly summarize/recall from the current step
    input only
  * Reflection: Should correctly reflect based on current input + this
    step's Memory output
  * Planning: Should plan reasonably based on current input + this
    step's Memory & Reflection outputs
  * Action: Should execute correctly based on current input + this
    step's Planning output
- Each module builds on previous modules' outputs FROM THE SAME STEP
- System errors (step_limit, tool_execution_error, etc.) should be
  identified separately

REQUIRED OUTPUT FORMAT (JSON):
{
    "error_detected": true/false,
    "error_type": "specific_error_type or no_error",
    "evidence": "Quote or description from module content supporting
    the detection",
    "reasoning": "Explanation of why this is (or isn't) an error based
     on the definition"
}

Be precise and base your detection on the actual content and error
definitions.
```

Figure 16: **Detector Prompt** used to identify specific error types with evidence and reasoning, returning a strict JSON schema.

Stage2-AgentDebug Prompt

Prompt Overview: Identify the earliest root-cause error that made success impossible and provide iterative follow-up guidance. Produces a structured JSON report of the critical step, module, error type, evidence, root cause, and cascading effects.

Prompt Content (Verbatim):

```
You are an expert at identifying critical failure points in agent
trajectories and providing high-priority, iterative follow-up
instructions that MUST be followed across all subsequent steps.\n
TASK: {task_description}\nTASK RESULT: FAILED\n\n
DEBUG ITERATION CONTEXT:
- Current debug attempt index: {attempt_index}
- Previously issued follow-up instructions:\n
STEP-BY-STEP ERROR ANALYSIS:\n{all_steps}\n
ERROR DEFINITIONS:\n{error_reference}\n\n
Your job is to identify the CRITICAL ERROR - the earliest and most
important error that led to task failure, and produce an iterative
follow-up instruction that will help avoid similar mistakes in future
attempts.\n\n
CRITICAL ERROR IDENTIFICATION APPROACH:\n You must take a HOLISTIC,
GLOBAL perspective to identify the true root cause of failure.
Do NOT rely on any predetermined severity weights or rankings\n
ANALYSIS GUIDELINES:
1. Consider the ENTIRE trajectory from a global perspective - understand
the task goal and how the agent's path diverged from success
2. Find the EARLIEST point where the agent made a decision or error that
set it on an irreversible path to failure
3. Early exploration steps (steps 1-3) are often normal and should NOT
be marked as critical unless there's a clear, fundamental error
4. An error is critical if:
   - It represents the ROOT CAUSE that made task success impossible
   - It caused a cascade of subsequent errors
   - The trajectory could have succeeded if THIS specific error had
   not occurred
   - IMPORTANT: Correcting this specific error would fundamentally
   change the trajectory toward success
5. Focus on causal chains - trace backwards from the failure to find
the origin point
6. IMPORTANT: Step 1 only has planning and action modules - no
memory or reflection is possible at step 1 since there's no history yet
   - Do NOT mark step 1 memory/reflection as critical errors
   - Early steps without memory/reflection modules are expected
7. Consider System and Others categories as potential critical errors:
   - System errors (step_limit, tool_execution_error, llm_limit,
   environment_error) may also be the true cause of failure
   - For example, if the agent was performing correctly but hit
   step_limit, that IS the critical error
   - Others category captures unusual failures not covered by standard
   error types\n
Identify the TRUE ROOT CAUSE that made the task unrecoverable.\n
REQUIRED OUTPUT FORMAT (JSON):\n{
    "critical_step": <step_number>,
    "critical_module": "<module_name: memory|reflection|planning|action|
    system|others>",
    "error_type": "<specific_error_type_from_definitions>",
    "root_cause": "Concise description of the fundamental problem",
    "evidence": "Specific quote or observation from trajectory
    supporting this identification",
    "correction_guidance": "Actionable advice for the agent to avoid the
    same mistake in that step",
    "cascading_effects": [{ "step": <step_number>,
    "impact": "description" }]}
```

Figure 17: **AgentDebug Prompt** used to locate the root cause of task failure and issue structured correction guidance.

Direct Prompting

Prompt Overview: Identify the earliest root-cause error that made success impossible and provide iterative follow-up guidance. Produces a structured JSON report of the critical step, module, error type, evidence, root cause, and cascading effects.

Prompt Content (Verbatim):

```
"TASK: {task_desc}"
"TASK RESULT: FAILED"
"ERROR DEFINITIONS:\n{error_ref}"
"Your job is to identify the CRITICAL ERROR - the single most "
"important error that led to task failure, and produce an iterative "
"follow-up instruction that will help avoid similar mistakes in future attempts."
"CRITICAL ERROR IDENTIFICATION APPROACH:"
"You must take a HOLISTIC, GLOBAL perspective to identify the true root cause of failure. "
"Do NOT rely on any predetermined severity weights or rankings."
"ANALYSIS GUIDELINES:"
"1. Consider the ENTIRE trajectory from a global perspective - understand "
"the task goal and how the agent's path diverged from success"
"2. The critical error can occur at ANY step in the trajectory, not just early steps. "
"Carefully analyze ALL steps before deciding. Do NOT default to step 1 lazily."
"3. An error is critical if:"
"   - It represents the ROOT CAUSE that made task success impossible"
"   - It caused a cascade of subsequent errors\n"
"   - The trajectory could have succeeded if THIS specific error had not occurred"
"   - IMPORTANT: Correcting this specific error would fundamentally change the
trajectory toward success"
"4. Focus on causal chains - trace backwards from the failure to find the origin point"
"5. IMPORTANT: Step 1 only has planning and action modules - no memory or reflection "
"is possible at step 1 since there's no history yet"
"   - Do NOT mark step 1 memory/reflection as critical errors"
"6. Consider System and Others categories as potential critical errors\n"
"7. IMPORTANT: Do NOT blindly pick step 1 as the critical step. Many failures happen "
"at later steps due to incorrect actions, faulty reasoning, or accumulated errors. "
"Read the FULL trajectory carefully and identify WHERE things actually went wrong."
"Identify the TRUE ROOT CAUSE that made the task unrecoverable.\n\n"
f"TRAJECTORY:\n{transcript}"
"REQUIRED OUTPUT FORMAT (JSON):"
"{"
'  "critical_step": <step_number>,\n'
'  "critical_module": "<module_name: memory|reflection|planning|action|system|others>",'
'  "error_type": "<specific_error_type_from_definitions>",\n'
'  "root_cause": "Concise description of the fundamental problem",\n'
'  "evidence": "Specific quote or observation from trajectory supporting
this identification",'
'  "correction_guidance": "Actionable advice for the agent to avoid the same
mistake in that step",'
'  "cascading_effects": [{ "step": <step_number>, "impact": "description" }]'
"}"
```

Figure 18: **Direct Prompt** used to locate the root cause of task failure and issue structured correction guidance, with fully trajectory

Baseline Prompts

Prompt Overview: Prompts used as baselines, including Tree-of-Thought (ToT) value scoring and proposal, a vanilla debug template, and a Self-Refine feedback prompt.

Prompt Content (Verbatim):

```
1. Prompts for TOT value prompt
base_prompt = (
    f"You are evaluating candidate NEXT actions for an agent in
    {self.env_type}.\n"
    f"Rate how promising each action is for achieving the goal from the
    CURRENT state.\n"
    f"{history_section}Current observation:\n{obs}\n\n"
    f"Candidates (JSON list): {cand_json}\n\n"
    "Scoring rubric (strict):\n"
    "- Use the FULL 0.0-1.0 range. Do NOT give all 1.0 or all equal
    scores.\n"
    "- 0.9-1.0: Directly and obviously advances the goal with minimal
    risk.\n"
    "- 0.7-0.9: Strongly promising next step.\n"
    "- 0.4-0.7: Plausible but uncertain; depends on missing
    preconditions.\n"
    "- 0.2-0.4: Weak progress or likely redundant.\n"
    "- 0.0-0.2: Invalid, circular, or contradicts recent
    history/admissible options.\n"
    "- Penalize repeating the same action that just failed, no-ops,
    or irrelevant moves.\n"
    "Return ONLY a JSON array of floats, aligned to input order, length
    equals number of candidates.")
Propose prompt: prompt = (
    f"You are choosing the agent's next action in {env_type}.\n"
    f"{history_desc}Current observation:\n{obs}\n\n"
    f"Propose up to {k} different, concrete next actions the agent
    could take next.\n"
    f"Each proposal must be a single executable action string in the
    exact format expected by the environment.\n"
    f"Avoid repeating ineffective or redundant actions from the
    history above.\n"
    f"Return only a JSON list of strings, no extra
    text.{diversity_desc}")

2. Prompts for vanilla debug
PROMPT_TEMPLATE = ( "Trajectory :\n{trajectory}\n\n"
    "Your task:\n"
    "1. Identify the earliest step whose action, plan, reflection, or
    memory directly leads the agent off track or repeats ineffective
    behaviour.\n"
    "2. Reference that exact step number (0-based) as shown in the
    trajectory. Do not shift to later steps of that error.\n"
    "3. Explain why the chosen step is wrong, citing relevant
    observation/action details.\n"
    "4. Suggest a concrete alternative for that same step that would
    move the agent toward success
    (e.g., a specific action to take instead).\n\n"
    "Respond strictly in the following format (single spaces around
    colons, no extra text):\n"
    "step: <number>\n"
    "reason: <one concise, specific sentence>\n"
    "suggestion: <one actionable suggestion for that step>\n")

3. Prompts for Self-Refine
PROMPT_TEMPLATE = ( "Current result: {trajectory}\n\n"
    "Why is this trajectory not finished the task?\n\n" "Feedback:")
```

Figure 19: **Baseline Prompts.** Includes Tree-of-Thought value and proposal prompts, a vanilla debug template for earliest error localization, and a Self-Refine feedback prompt.

Environment Rollout Prompt - ALFWorld

Prompt Overview: This is the environment rollout prompt for ALFWorld.

Prompt Content (Verbatim):

```
ALFWORLD_TEMPLATE_NO_HIS = """
You are an expert agent operating in the ALFRED Embodied Environment.
Your task is: {task_description}
Your current observation is: {current_observation}
Your admissible actions of the current situation are:
{admissible_actions}.\n
Please begin by analyzing the situation and planning your approach:\n
<plan>\n Plan the next step:
- Given what I've learned, what should I do next?
- Please explain why this plan is helpful for the next action?
- What do I expect this action to achieve?\n</plan>\n
<action>\n
Finally, choose ONE admissible action for the current step and choose
it within {admissible_actions}. \n</action>
"""

ALFWORLD_TEMPLATE = """ You are an expert agent operating in the ALFRED
Embodied Environment. Your task is to: {task_description} Prior to
this step, you have already taken {step_count} step(s). Below are
the most recent {history_length} observaitons and the corresponding
actions you took: {action_history} You are now at step {current_step}
and your current observation is:  {current_observation} Your admissible
actions of the current situation are: {admissible_actions}.\n
Now it's your turn to take an action.\n
You should first recall relevant past experiences and reason from our
conversation history, then MUST summarize within <memory> </memory>
tags like this:\n
<memory> \n
Look at the past observations and actions from our conversation history.
- Please retrieve the most relavent memory for this step including the
relevant observation and action in a RAG style along with step number.
- These memory shall be helpful milestones to solve this task.</memory>

After that, you should reflect on the last action and its outcome,
then MUST summarize within <reflection> </reflection> tags like this:

<reflection>
Reflect on the last action and its outcome
- Did I complete the task goal?
- Was last action successful or did it encounter issues?
- Am I making progress toward the task goal?
- If the action did not go as expected and did not result in progress,
provide constructive feedback to guide the next planning step.
</reflection>

After that, you should plan next step based on memory and reflection,
then MUST summarize within <plan> </plan> tags like this:

<plan>
Plan the next step based on memory and reflection
- Given what I've learned, what should I do next?
- Please explain why this plan is helpful for the next action?
- What do I expect this action to achieve?
</plan>

<action>
Finally, choose ONE admissible action for the current step and choose it
within {admissible_actions}.
</action>
"""
```

Figure 20: **Environment Rollout Prompt** used for ALFWorld.

Environment Rollout Prompt - Webshop

Prompt Overview: This is the environment rollout prompt for Webshop.

Prompt Content (Verbatim):

```
WEBSHOP_TEMPLATE_NO_HIS = """
You are an expert agent operating in the WebShop e-commerce environment.
Your task is: {task_description}
Your current observation is: {current_observation}
Your admissible actions of the current situation are:
{available_actions}.\n
Please begin by analyzing the situation and planning your approach:

<plan> Plan the next step:
- Given what I've learned, what should I do next?
- Please explain why this plan is helpful for the next action?
- What do I expect this action to achieve? </plan>

<action> Finally, choose ONE admissible action for the current step and
choose it within {available_actions}. </action>
"""

WEBSHOP_TEMPLATE = """
You are an expert agent operating in the WebShop e-commerce environment.
Your task is to: {task_description}
Prior to this step, you have already taken {step_count} step(s). Below
is a compact summary of all steps: {action_history}
You are now at step {current_step} and your current observation is:
{current_observation}
Your admissible actions of the current situation are:
{available_actions}.\n Now it's your turn to take an action.

You should first recall relevant past experience and reason from the
history context, then MUST summarize within <memory> </memory> tags
like this: <memory> Look at the history context above.
- Please retrieve the most relevant memory for this step including the
relevant observation and action in a RAG style along with step number.
- These memory should be helpful milestones to solve this task.</memory>

After that, you should reflect on the last action and its outcome, then
MUST summarize within <reflection> </reflection> tags like this:

<reflection> Reflect on the last action and its outcome
- Did I complete the task goal?
- Was last action successful or did it encounter issues?
- Am I making progress toward the task goal?
- If the action did not go as expected and did not result in progress,
provide constructive feedback to guide the next planning step.
</reflection>

After that, you should plan the next step based on memory and
reflection, then MUST summarize within <plan> </plan> tags like this:

<plan> Plan the next step based on memory and reflection
- Given what I've learned, what should I do next?
- Please explain why this plan is helpful for the next action?
- What do I expect this action to achieve? </plan>

<action> Finally, choose ONE admissible action for the current step and
choose it within {available_actions}. </action>
"""
```

Figure 21: **Environment Rollout Prompt** used for Webshop.

Environment Rollout Prompt - GAIA

Prompt Overview: This is the environment rollout prompt for GAIA.

Prompt Content (Verbatim):

```
TOOL_USE_TEMPLATE_NO_HIS = """
You are an expert research assistant capable of using various tools to
gather information and solve complex problems.\nTask: {task_description}
Available Tools: {available_tools}\n Current Observation:
{current_observation}\n  Instructions:
1. Analyze the task and determine what information you need
2. Use available tools to gather information when needed
3. Reason through the information step by step
4. When you have sufficient information, provide your final answer
in <answer></answer> tags\n Format for tool usage:\n <action> tool:
[tool_name] parameters: {{"param1": "value1", "param2": "value2"}}
</action>\n Now it's your turn to take an action. You shall first reason
step-by-step about the current situation. This reasoning process MUST be
enclosed within <plan></plan> tags. Once you've finished your reasoning,
you should either use a tool or provide your final answer within
<answer> </answer> tags.\n """\n TOOL_USE_TEMPLATE_LAST_STEP = """
You are an expert research assistant capable of using various tools to
gather information and solve complex problems.\n Task:
{task_description}\n Prior to this step, you have already taken
{step_count} step(s). Below are the full {history_length} observations
and the corresponding actions you took: {action_history}\n
You are now at step {current_step} and this is the final step.
Current Observation: {current_observation}
You must provide your final answer within <answer> </answer> tags.
Even if the evidence is incomplete, infer the most plausible answer.
Never respond with "unknown", "cannot determine", or similar phrases."""
TOOL_USE_TEMPLATE = """ You are an expert research assistant capable of
using various tools to gather information and solve complex problems.\n
Task: {task_description}\n Prior to this step, you have already taken
{step_count} step(s). Below are the most recent {history_length}
observations and the corresponding actions you took: {action_history}\n
You are now at step {current_step}. Current Observation:
{current_observation}\n Available Tools:{available_tools}\n  You should
first recall relevant past experiences and reason from our conversation
history, then MUST summarize within <memory_recall> </memory_recall>
tags like this: <memory>
Look at the past observations and actions from our conversation history.
- Please retrieve the most relavent memory for this step including the
relevant observation and action in a RAG style with the step number.
- These memory should be helpful milestones to solve this task.</memory>
After that, you should reflect on the last action and its outcome, then
MUST summarize within <reflection> </reflection> tags like this:
<reflection> Reflect on the last action and its outcome
- Did I complete the task goal?
- Was last action successful or did it encounter issues?
- Am I making progress toward the task goal?
- If the action did not go as expected and did not result in progress,
provide feedback to guide the next planning step. </reflection>\n
Given from the analysis from the memory and reflection, if we
get the final answer,we should provide it within <answer></answer> tags.
If we don't get the final answer, you should plan the next step based on
memory and reflection, then MUST summarize within <plan> </plan> tags
like this: <plan> Plan the next step based on memory and reflection
- Given what I've learned, what should I do next?
- Please explain why this plan is helpful for the next action?
- What do I expect this action to achieve? </plan>\n
Finally, choose ONE admissible action for the current step and present
it within the <action> </action> tags. <action> action: [tool_name]
parameters: {{"param1": "value1", "param2": "value2"}}</action> """
```

Figure 22: **Environment Rollout Prompt** used for Webshop.

A.12 Error Types Examples

Error Type - Over Simplification

task type	webshop
task description	Find me women's sweaters with relaxed fit, long sleeve with color: white, and size: xx-large, and price lower than 40.00 dollars
critical failure module	memory
evidence	Evidence: No previous steps are directly relevant to the current observation as this is step 2 and the first step was a search query without detailed navigation or selection of products. Reasoning: The memory module output oversimplifies the history by stating that 'No previous steps are directly relevant,' ignoring that the initial search query contained all of the task's complex constraints...
critical step	2

step trajectory

```
observation: 'Back to Search' [SEP] 'Page 1 (Total results: 50)' [SEP] 'Next >' [SEP] 'B07DKGJR74'
↪  [SEP] 'Amazon Brand - Daily Ritual Women's 100% Cotton Oversized Fit V-Neck Pullover Sweater'
↪  [SEP] '$20.66 to $29.2' [SEP] 'B09KRLGSC5' [SEP] 'SweatyRocks Women's Oversized Sweater
↪  Crewneck Long Sleeve Knit Tops Pullover Jumper Tops' [SEP] '$25.99' [SEP] 'B07MGB73NJ' [SEP]
↪  'Viracy Women's Short Sleeve V-Neck Casual Flowy Tunic Shirt (M-3XL)' [SEP] '$16.99 to $26.99'
↪  [SEP] 'B09KP78G37' [SEP] 'Women Faux Fur Lined Jacket Coat Winter Warm Thick Fleece Outwear
↪  Trench Zipper Plus Size Long Sleeve Plush Overcoat' [SEP] '$47.41 to $59.07' [SEP] 'B09QT2369P'
↪  [SEP] 'Stylish Women's Long Sleeve Sweatshirt Letters Printed Tops Casual Crewneck Solid
↪  Blouses Loose Fitting Soft Pullover' [SEP] '$11.99 to $14.99' [SEP] 'B09QGK5XHZ' [SEP]
↪  'WENKOMG1 Men's Long Sleeve Undershirt with Mask Turtleneck Hooded T-Shirt Solid Color Workout
↪  Tops Zipper Side Slit Shirts Slim Fit Sweatshirt Spring/Summer Tee Shirts(Gray,)' [SEP] '$8.39'
↪  [SEP] 'B09Q67H373' [SEP] 'JSPOYOU 2022 Newly T-Shirt for Mens Funny 3D Graphics Pattern
↪  Crewneck Short Sleeve Tees Big and Tall Summer Casual Comfy Top' [SEP] '$1.99 to $8.99' [SEP]
↪  'B09ND8P2QR' [SEP] 'InterestPrint Deer and Stars Men's Loungewear Pajama Sets - Long Sleeve Tee
↪  and Jogger Pant' [SEP] '$43.59' [SEP] 'B09ND9DP7J' [SEP] 'InterestPrint Gold Horse Pattern
↪  Men's 2-Piece Sleepwear Set, Long Sleeve Shirt with Pants Loungewear'...
memory: "No previous steps are directly relevant to the current observation as this is step 2 and
↪  the first step was a search query without detailed navigation or selection of products."
reflection: "Reflecting on the last action:\n- The task goal of finding women's sweaters with
↪  specific criteria has not yet been completed.\n- The last action, a search, was successful in
↪  retrieving a list of potential products.\n- Progress toward the task goal is being made as we
↪  now have results to evaluate.\n- No issues were encountered, but further filtering or
↪  exploration is needed to find products matching all the criteria."
plan: "Plan the next step based on memory and reflection:\n- The plan is to click on products that
↪  seem to match the search criteria to gather more details.\n- Clicking on 'B07DKGJR74' (Amazon
↪  Brand - Daily Ritual Women's 100% Cotton Oversized Fit V-Neck Pullover Sweater) will provide
↪  more information to confirm if it matches the desired color, size, and fit.\n- This step is
↪  helpful as it will help verify product details against the specified criteria.\n- I expect this
↪  action to either confirm a match or eliminate this product from consideration."
action: "take soapbottle 2 from cabinet 4"
```

Figure 23: **Error Type: Over Simplification**

Error Type - Impossible Action

task type	alfworld
task description	put two soapbar in toilet
critical failure module	plan
evidence	Task requires 'two soapbar', but at step 10 the plan proposes: 'take soapbottle 2 from cabinet 4 . . . ' Observation at step 10 lists only 'cloth 1' and 'soapbottle 2' in cabinet 4; no soapbar is present.
critical step	10

step trajectory

```
observation: "You open the cabinet 4. The cabinet 4 is open. In it, you see a cloth 1, and a
↪  soapbottle 2.
admissible_actions: 'close cabinet 3'\n 'close cabinet 4'\n 'examine cabinet 3'\n 'examine cabinet
↪  4'\n 'go to cabinet 1'\n 'go to cabinet 2'\n 'go to countertop 1'\n 'go to garbagecan 1'\n 'go
↪  to handtowelholder 1'\n 'go to sinkbasin 1'\n 'go to sinkbasin 2'\n 'go to toilet 1'\n 'go to
↪  toiletpaperhanger 1'\n 'go to towelholder 1'\n 'inventory'\n 'look'\n 'take cloth 1 from
↪  cabinet 4'\n 'take soapbottle 2 from cabinet 4'.
memory: "Looking at the past observations and actions from our conversation history, a relevant
↪  memory for this step is from Observation 9 and Action 9. At step 9, the action was to 'open
↪  cabinet 4' because it was a potential location for soap bars and cabinets 1 through 3 had
↪  already been checked. This action resulted in Observation 10, where the cabinet 4 is open and
↪  contains a cloth 1 and a soapbottle 2. This is a helpful milestone because it indicates that
↪  cabinet 4 contains items that could be relevant to the task, but it does not contain soap bars
↪  as hoped. Instead, it contains a soapbottle, which might not be what we're looking for since
↪  the task specifies \"soapbar.\""
reflection: "Reflecting on the last action and its outcome, the task goal of putting two soap bars
↪  in the toilet has not been completed. The last action to 'open cabinet 4' was successful in the
↪  sense that it provided new information about the environment, but it did not directly
↪  contribute to achieving the task goal because it did not yield any soap bars. Progress toward
↪  the task goal is limited because the primary objective is to find and relocate soap bars, and
↪  so far, none have been found. Constructive feedback for the next planning step would be to
↪  consider locations that might more likely contain soap bars or to re-evaluate the items found
↪  for potential use or clues."
plan: "Planning the next step based on memory and reflection, given what's been learned, the next
↪  action should focus on examining the contents of cabinet 4 more closely or moving to a new
↪  location that could potentially contain soap bars. Since cabinet 4 is already open and its
↪  contents are known, examining it further might not yield additional useful information.
↪  However, before moving on, it's essential to consider if the soapbottle 2 could be used or if
↪  there's a way to obtain soap bars from the current location. A helpful next step would be to
↪  take the soapbottle 2 from cabinet 4, not because it directly achieves the task goal, but
↪  because it might provide a clue or an alternative solution, or simply to inventory the item for
↪  future reference. This action is helpful because it engages with the environment in a way that
↪  could potentially reveal more information or provide a resource that could be used creatively
↪  to achieve the task goal."
action: "take soapbottle 2 from cabinet 4"
```

Figure 24: **Error Type: Impossible Action**

Error Type - Constraint Ignorance

task type	webshop
task description	Find me wash cold, machine wash men's shirts with polyester heathers, heathers cotton, cotton heather, needle sleeve, classic fit with color: navy, and fit type: women, and size: small, and price lower than 50.00 dollars
critical failure module	plan
evidence	Evidence from step 1: 'Planning ignores the explicit fit type: women, size: small, and price lower than 50.00 dollars constraints in its summary of what it needs to search for, instead stating 'search for men's shirts that match specific criteria: fabric types (polyester heathers, heathers cotton, cotton heather), design features (needle sleeve, classic fit), color (navy), fit type (women), size (small), and a price below 50 dollars.' ... it lists all user constraints but does not acknowledge potential incompatibility or how to prioritize or filter for those constraints, especially given that 'fit type: women' is not standard for 'men's shirts.' This shows a lack of attention to task constraints and possible resource or logical conflicts, fitting constraint ignorance.'
critical step	1

step trajectory

```
admissible_actions: 'search[<your query>]',\n'click[search]'
plan: To find the desired product, I need to search for men's shirts that match specific criteria:
↪  fabric types (polyester heathers, heathers cotton, cotton heather), design features (needle
↪  sleeve, classic fit), color (navy), fit type (women), size (small), and a price below $50. The
↪  current observation allows me to perform a search query, which is necessary to locate products
↪  matching these specific attributes.
action: search[\"men's shirts polyester heathers heathers cotton cotton heather needle sleeve
↪  classic fit navy color women's fit small size under 50 dollars\"]
```

Figure 25: **Error Type: Constraint Ignorance**

Error Type - Inefficient Plan

task type	alfworld
task description	find two pencils and put them in drawer.
critical failure module	plan
evidence	At step 4, LLM chose the action 'look' despite already confirming there was only one pencil on the desk and knowing that the room contained multiple drawers and shelves likely to hold another pencil. This resulted in repeated 'Nothing happens' outcomes in subsequent steps.
critical step	4

step trajectory

```
observation: "You pick up the pencil 2 from the desk 1."

admissible_actions: 'examine desk 1'\n 'examine pencil 2'\n 'go to bed 1'\n 'go to drawer 1'\n 'go
↪  to drawer 2'\n 'go to drawer 3'\n 'go to drawer 4'\n 'go to drawer 5'\n 'go to drawer 6'\n 'go
↪  to drawer 7'\n 'go to drawer 8'\n 'go to dresser 1'\n 'go to garbagecan 1'\n 'go to shelf 1'\n
↪  'go to shelf 2'\n 'go to shelf 3'\n 'go to shelf 4'\n 'go to shelf 5'\n 'inventory'\n 'look'\n
↪  'move pencil 2 to desk 1'

memory: "Looking at the past observations and actions from our conversation history, the most
↪  relevant memory for this step is: \nAt step 2, the observation was 'You arrive at desk 1. On
↪  the desk 1, you see a book 2, a cd 2, a creditcard 2, and a pencil 2.' and the action was 'take
↪  pencil 2 from desk 1'. This memory is helpful because it indicates that there is at least one
↪  pencil on the desk, and we have already taken it. Now, we need to find another pencil and put
↪  both pencils in a drawer."

reflection: "Reflecting on the last action and its outcome, we can see that the last action 'take
↪  pencil 2 from desk 1' was successful, and we have picked up the pencil 2 from the desk 1. We
↪  have made progress toward the task goal, but we still need to find another pencil and put both
↪  pencils in a drawer. The task is not yet complete."
plan: "Planning the next step based on memory and reflection, we should examine the desk 1 again to
↪  see if there is another pencil. This plan is helpful because it allows us to gather more
↪  information about the objects on the desk and potentially find the second pencil. We expect
↪  this action to achieve a better understanding of the objects on the desk and possibly find the
↪  second pencil."
action: "examine desk 1"
```

Figure 26: **Error Type: Inefficient Plan**

Error Type - Misalignment

task type	GAIA
task description	
critical failure module	action
evidence	Evidence: Plan: 'I will modify the approach to directly explore the Tropicos database or a reliable botanical taxonomy reference to find the Tropicos ID for the Order Helotiales.' Action: tool: wikipedia knowledge searcher parameters: "query": "Helotiales Tropicos ID"
critical step	3

step trajectory

```
observation: Tool 'google_search' executed.\nResult: [\n  {\n    \"title\": \"Untitled\",\n
↪  \"link\": \"https://www.tropicos.org/name/Search?name=Helodiaceae\",\n    \"snippet\": \"\"\n
↪  }\n]

memory: "In step 1, I attempted to find the Tropicos ID for the Order Helotiales using a Google
↪  search, which resulted in a link that was not directly relevant.\n- In step 2, I used the
↪  Wikipedia knowledge searcher to find the Tropicos ID for Helotiales, but no results were found
↪  for the query."

reflection: "The last action did not yield the desired Tropicos ID for the Order Helotiales, as no
↪  results were found on Wikipedia. This indicates that Wikipedia might not have the specific
↪  identifier information needed, and it confirms the need to access more specialized or direct
↪  resources for taxonomic data. Although there has been no progress in finding the Tropicos ID,
↪  identifying the limitations of current methods is valuable for refining the search approach."

plan: "I will use the Google search tool again, but with a refined query to specifically target the
↪  Tropicos database or another authoritative botanical taxonomy site that might list the Tropicos
↪  ID for Helotiales. This plan is likely to be effective because it focuses on accessing
↪  specialized databases where the Tropicos ID is more likely to be documented. The goal of this
↪  action is to obtain the Tropicos ID so that I can proceed with calculating the ISBN-10 check
↪  digit."
action: "tool: google_search\nparameters: {\"query\": \"Helotiales Tropicos ID site:tropicos.org\",
↪  \"num_results\": 1}"
```

Figure 27: **Error Type: Misalignment**

Error Type - Format Error

task type	alfworld
task description	clean some plate and put it in shelf.
critical failure module	action
evidence	Step 15 has a non-empty plan but the recorded action is empty. Step 16 observation returns 'Nothing happens'. From Step 17 onward, admissible actions at the dining table never include 'take plate 1 ...', yet attempts persist.
critical step	15

step trajectory

```
observations: "On the diningtable 1, you see a apple 1, a bread 1, a butterknife 1, a cup 3, a fork
↪  1, a houseplant 1, a knife 1, a lettuce 3, a lettuce 2, a peppershaker 2, a plate 2, a plate 1,
↪  a saltshaker 1, a soapbottle 2, a soapbottle 1, a spoon 2, a spoon 1, a winebottle 3, and a
↪  winebottle 2."

admissible_actions: "'examine diningtable 1'\n 'examine dishsponge 1'\n 'go to cabinet 1'\n 'go to
↪  cabinet 10'\n 'go to cabinet 11'\n 'go to cabinet 12'\n 'go to cabinet 13'\n 'go to cabinet
↪  2'\n 'go to cabinet 3'\n 'go to cabinet 4'\n 'go to cabinet 5'\n 'go to cabinet 6'\n 'go to
↪  cabinet 7'\n 'go to cabinet 8'\n 'go to cabinet 9'\n 'go to coffeemachine 1'\n 'go to
↪  countertop 1'\n 'go to countertop 2'\n 'go to drawer 1'\n ..."

memory: "Looking at the past observations and actions from our conversation history, a relevant
↪  memory for this step is from step 10, where the observation was \"You open the drawer 1. The
↪  drawer 1 is open. In it, you see a dishsponge 1.\" and the action was \"take dishsponge 1 from
↪  drawer 1\". This memory is helpful because it shows that the dishsponge 1, which is likely
↪  needed for cleaning, has been retrieved and is now in the agent's possession. Another relevant
↪  memory is from step 14, where the observation was ..."

reflection: "Reflecting on the last action and its outcome, the last action was \"examine
↪  diningtable 1\" and the outcome was that the agent observed the current state of the
↪  diningtable 1. The task goal is to clean some plate and put it in a shelf, and the last action
↪  did not directly contribute to completing this goal. However, it did provide information about
↪  the current state of the diningtable 1, which is necessary for planning the next step. The
↪  agent is making progress toward the task goal by gathering information, but the actual cleaning
↪  and putting away of the plate has not started yet. To make progress, the agent should use the
↪  dishsponge 1 to clean a plate and then put it away in a shelf.",

plan: "Planning the next step based on memory and reflection, the agent should use the dishsponge 1
↪  to clean a plate. Since the agent is currently at the diningtable 1 and there are plates
↪  present, the agent should take a plate from the diningtable 1 and clean it using the dishsponge
↪  1. After cleaning the plate, the agent can put it away in a shelf. This plan is helpful because
↪  it directly addresses the task goal and uses the information gathered from previous actions.
↪  The expected outcome of this action is that the agent will have a clean plate that can be put
↪  away in a shelf."

action: "take plate 1 from diningtable 1."
```

Figure 28: **Error Type: Format Error**

Error Type - Progress Misjudge

task type	alfworld
task description	put a clean bowl in shelf.
critical failure module	reflection
evidence	Step 16 reflection: 'This means that the task goal of putting a clean bowl in the shelf has been completed. The last action was successful, and progress was made toward the task goal. However, to confirm that the bowl is indeed in the shelf and the task is fully completed, further verification might be necessary.'
critical step	16

step trajectory

```
observations: ""You move the bowl 1 to the shelf 1."

admissible_actions: "'examine shelf 1'\n 'go to cabinet 1'\n 'go to cabinet 2'\n 'go to cabinet
↪  3'\n 'go to cabinet 4'\n 'go to cabinet 5'\n 'go to cabinet 6'\n 'go to cabinet 7'\n 'go to
↪  cabinet 8'\n 'go to cabinet 9'\n 'go to coffeemachine 1'\n 'go to countertop 1'\n 'go to
↪  countertop 2'\n 'go to countertop 3'\n 'go to drawer 1'\n 'go to drawer 2'\n 'go to drawer 3'\n
↪  'go to drawer 4'\n 'go to drawer 5'\n 'go to drawer 6'\n 'go to drawer 7'\n 'go to drawer 8'\n
↪  'go to drawer 9'\n 'go to fridge 1'\n 'go to garbagecan 1'\n 'go to microwave 1'\n 'go to shelf
↪  2'\n 'go to shelf 3'\n 'go to sinkbasin 1'\n 'go to stoveburner 1'\n 'go to stoveburner 2'\n
↪  'go to stoveburner 3'\n 'go to stoveburner 4'\n 'go to toaster 1'\n 'inventory'\n 'look'\n
↪  'take bowl 1 from shelf 1'."

memory: "Looking at the past observations and actions from our conversation history, a relevant
↪  memory for this step is from Step 13 and Step 14. \nAt Step 13, the observation was: \"You
↪  arrive at countertop 1. On the countertop 1, you see a bowl 1, a butterknife 2, a butterknife 1,
↪  a dishsponge 1, a fork 1, and a kettle 1.\" \nThe action taken was: \"take bowl 1 from
↪  countertop 1\". \nThen at Step 14, the observation was: \"You pick up the bowl 1 from the
↪  countertop 1.\" \nThe action taken was: \"go to shelf 1\". \nThis memory is helpful because it
↪  shows how the bowl was obtained and moved to the shelf, which is the goal of the task."

reflection: "Reflecting on the last action and its outcome, the last action was to move the bowl 1
↪  to the shelf 1. \nThe outcome was that the bowl was successfully moved to the shelf. \nThis
↪  means that the task goal of putting a clean bowl in the shelf has been completed. \nThe last
↪  action was successful, and progress was made toward the task goal. \nHowever, to confirm that
↪  the bowl is indeed in the shelf and the task is fully completed, further verification might be
↪  necessary.",

plan: "Planning the next step based on memory and reflection, given what has been learned, the next
↪  action should be to verify that the bowl is indeed on the shelf. \nThis plan is helpful because
↪  it ensures that the task goal has been fully achieved and that there are no unexpected issues.
↪  \nThe expected outcome of this action is to confirm the presence of the bowl on the shelf,
↪  thereby confirming the completion of the task."

action: "examine shelf 1."
```

Figure 29: **Error Type: Progress Misjudge**