# ParaRecover: A Process-Level Benchmark for Error Localization and Recovery in Parallel Tool-Use Agents

**Bowen Guan**[1], **Zhentao Yin**[1], **Yanming Shen**[1*]
[1]School of Computer Science and Technology, Dalian University of Technology, China
20201071138@mail.dlut.edu.cn, yinzhentao2003@mail.dlut.edu.cn
shen@dlut.edu.cn

## Abstract

Existing agent benchmarks mainly evaluate final task success or tool-call correctness, providing limited insight into whether agents can reliably diagnose and recover from intermediate execution failures. This limitation becomes particularly critical in multi-turn parallel tool-use scenarios, where errors may propagate across dependent branches and trigger cascading failures. We introduce ParaRecover, a process-level benchmark for evaluating error localization and recovery in multi-turn parallel tool-use agents. Built upon a fine-grained taxonomy of 14 error types covering planning dependencies, tool selection, and argument matching, the benchmark comprises 10,626 instances spanning two difficulty levels. To enable fine-grained, process-oriented evaluation, we further propose the SDE rubric, which measures structural integrity, diagnostic reasoning, and evolutionary strategy during agent execution. Experiments across more than ten mainstream LLMs reveal that even state-of-the-art models still struggle with multi-turn error propagation, implicit tool-use failures, and precise replanning. Moreover, we demonstrate that the SDE rubric provides effective supervision signals for improving agents' reflective recovery capabilities. Our data and code are available at https://github.com/gbw206/ParaRecover.

## 1 Introduction

Large language model agents are increasingly used to solve complex tasks through planning, tool use, and interaction with external environments (Yao et al., 2022; Qin et al., 2023; Huang et al., 2024). In real-world scenarios such as scientific research, code generation, and data analysis (Ma et al., 2024; Yang et al., 2024; Hu et al., 2024), tasks are often not simple sequential processes. Instead, they are usually multi-turn parallel processes, which involve multiple parallel subgoals, branching tool calls, and cross-step dependencies (Huang et al., 2024; Wang et al., 2024). Therefore, systematically evaluating agents' planning, decision-making, and error-recovery abilities in multi-turn parallel tool-use scenarios has become an important problem.

Existing benchmarks for agent and tool use mainly focus on final task outcomes, such as answer correctness and overall task completion rate (Mialon et al., 2024; Yao et al., 2024; Liu et al., 2025). However, successful task completion does not necessarily imply reliable agent behavior. An agent may still complete the task through redundant calls, repeated trial and error, or incorrect recovery strategies, leading to high cost in real systems. These end-to-end metrics are useful for measuring final performance, but they provide limited insight into whether agents can correctly localize failures, diagnose their causes, and recover from corrupted intermediate states.

As shown in Table 1, although some recent studies consider process-level behaviors, they typically focus on tool-call success rate or provide only coarse-grained or indirect judgments of intermediate decisions or format (Li et al., 2023; Qin et al., 2023; Patil et al., 2025; He et al., 2025; Lu et al., 2025). As a result, they may fail to capture the fine-grained behaviors required in multi-turn parallel execution, such as deciding concurrent subtasks, identifying cross-step dependencies, assessing the impact of failures on downstream branches, and replanning after errors. Existing evaluation paradigms rarely characterize these recovery challenges in a systematic way, making it difficult to understand how and where agents fail. In practical applications, however, robust error diagnosis, reflection, and replanning capabilities are crucial for reliable long-horizon agent execution.

Motivated by these limitations, we construct a fine-grained error taxonomy based on agent execution trajectories in real-world multi-turn parallel scenarios, covering 14 error types across planning dependencies, tool selection, and argument



*Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, pages 22479–22496
October 24-29, 2026 ©2026 Association for Computational Linguistics

| Benchmark | Structured Trajectory Eval. | Parallel Calls | Error-State Eval. | Reflective Localization | Replanning Decision | Process-Level Metrics |
|---|---|---|---|---|---|---|
| API-Bank | ✗ | ✗ | ✓ | ✗ | ✗ | △ |
| Tool-Bench | ✗ | ✗ | △ | ✗ | ✗ | △ |
| BFCL | ✗ | ✓ | ✗ | ✗ | ✗ | △ |
| ToolSandbox | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ |
| TRAJECT-Bench | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ |
| ParaRecover | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 1: Comparison between previous agent benchmarks and ParaRecover in evaluation dimensions. ✓ denotes full support, ✗ denotes no support, and △ denotes indirect support.

matching. Building upon this taxonomy, we develop ParaRecover, a benchmark designed for complex multi-turn parallel agent execution scenarios. ParaRecover covers two difficulty levels: LEVEL-1 only includes errors introduced in the most recent execution, while LEVEL-2 involves multi-turn propagation and various error types, comprehensively evaluating the agent's error localization and replanning capabilities. To further refine the evaluation, we introduce the **SDE** evaluation rubric, which measures agent behavior from three dimensions: **S**tructural Integrity, **D**iagnostic Reasoning, and **E**volutionary Strategy. Our main contributions are as follows:

- We introduce a process-level benchmark for multi-turn parallel tool use, comprising 10,626 instances across two difficulty levels, explicitly evaluating agents' intermediate decision-making, error localization, and corrective replanning.
- We develop and validate a fine-grained error taxonomy for parallel agent execution. Based on real rollout trajectories from multiple models, we further characterize how these errors emerge and distribute in practice.
- We propose the SDE Rubric, which evaluates agent from Structural Integrity, Diagnostic Reasoning, and Evolutionary Strategy, enabling fine-grained assessment of reflection, backtracking, and decision capability after errors.
- We reveal a substantial gap between final task success and process reliability in current agents. Although state-of-the-art models often achieve high task completion rates, they still exhibit severe weaknesses in dependency reasoning, implicit failure localization, and efficient recovery under multi-turn error propagation.

## 2 Problem Definition

We consider real-world tasks, which contain a substantial number of subtasks that are conditionally independent and therefore can be executed in parallel. Although parallel execution can significantly improve efficiency, it also imposes substantially greater demands on agent planning and decision-making capabilities. To capture this more realistic and challenging setting, we formulate agent execution as a **multi-turn parallel decision process**.

To formalize parallelism, we represent the execution process as a DAG $G = (V, E)$, where each node $v_i \in V$ denotes a subtask or tool-use operation, and each edge $(v_i, v_j) \in E$ indicates that $v_j$ depends on the completion of $v_i$. Under this formulation, the prerequisite set of node $v_j$ can be written as

$$Dep(v_j) = \{v_i \in V | (v_i, v_j) \in E\}. \quad (1)$$

A node $v_j$ is executable only when all of its dependencies have been satisfied, i.e.,

$$\forall v_i \in Dep(v_j), \quad s(v_i) = 1, \quad (2)$$

where $s(v_i) \in \{0, 1\}$ represents the state of node $v_i$, indicating whether it has been completed. Consequently, tool invocations are executed layer-wise strictly along the topological structure of the DAG. Originating from the root, the graph naturally partitions into hierarchical layers: nodes directly dependent on the root constitute the first layer, nodes dependent on the first layer form the second, and so forth.

Since the topological depth reached in any given round is non-deterministic, the agent must dynamically infer the execution state. At each interaction step $t$, the agent receives the partially executed DAG $G^{(t-1)}$ from the previous round, along with the execution observations $O^{(t)}$ from the recently invoked tools. A critical capability required of the agent is to deduce the executable frontier for

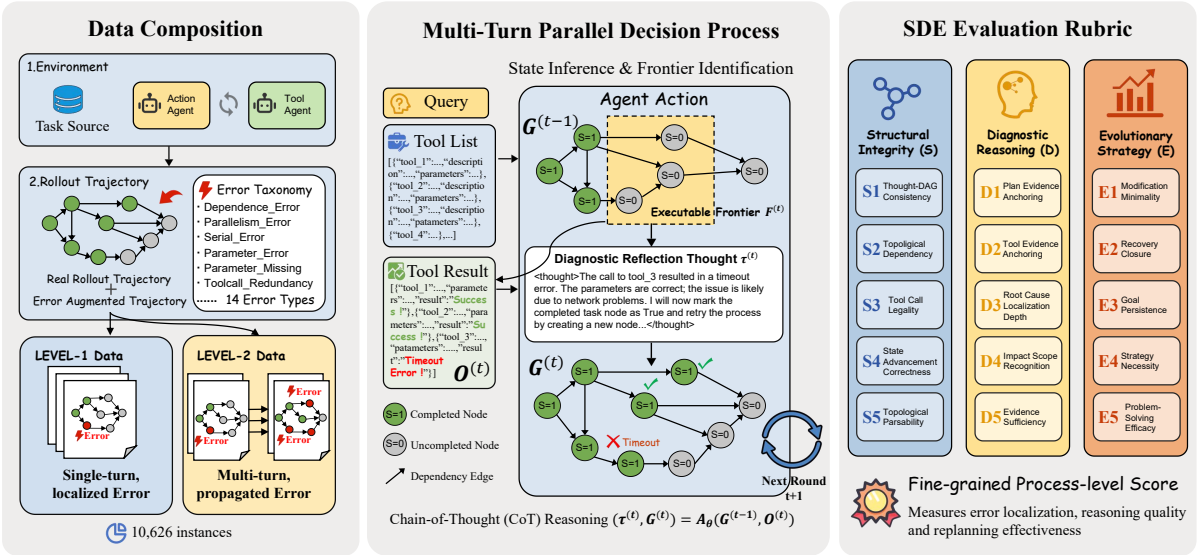


Figure 1: Overview of the ParaRecover Benchmark

the current round. This frontier, denoted as $F^{(t)}$, comprises all pending nodes whose prerequisite dependencies have been strictly satisfied.

The core objective of the model at step $t$ is multifaceted: it must critically analyze the structural correctness of the preceding DAG, identify the target nodes within $F^{(t)}$, and evaluate the observations $O^{(t)}$ to diagnose which tool invocations succeeded or failed. To formalize this diagnostic reflection and decision-making process, we constrain the agent to follow a Chain-of-Thought (CoT) (Wei et al., 2023) reasoning paradigm. Specifically, the agent $A_\theta$ is prompted to generate a textual reflection $\tau^{(t)}$ (thought) followed by an updated graph $G^{(t)}$, defined as the mapping:

$$(\tau^{(t)}, G^{(t)}) = A_\theta(G^{(t-1)}, O^{(t)}). \quad (3)$$

The overall problem definition is shown in the middle part of Fig. 1.

## 2.1 ParaRecover

As shown in Fig. 1, ParaRecover is a process-level benchmark evaluating error localization and recovery in parallel tool-use agents via DAG.

### 2.1.1 Task Source

We source our initial tasks from BUTTON (Chen et al., 2024a), which constructs complex trajectories by introducing dependencies among diverse atomic tasks. This compositional structure exhibits a natural correspondence with the edges and subtask nodes inherent in DAG planning. Regarding the tool execution environment, we follow Chen et al. (2024a) and adopt the tool agent-based simulation paradigm. Since our primary objective is to evaluate the model's capacity for reflection and decision-making amidst various intermediate error states rather than assessing the stability of external APIs, this controlled simulated environment serves as an ideal execution backend.

### 2.1.2 Error Taxonomy

A core challenge is to determine what types of errors should be recovered. Previous work has identified several common failure modes (Zhuang et al., 2023; Chen et al., 2025), such as incorrect tool names, invalid arguments, and non-executable calls. However, these error types mainly capture local mistakes at the function-call level and do not fully characterize the failures that emerge in multi-turn parallel execution. In a DAG-based agent workflow, the model must additionally maintain valid dependency structures, identify which nodes can be executed in parallel and update the global plan after partial completion.

Therefore, beyond inherited tool-level errors, we extract additional error categories from large-scale rollout traces, including parallelization errors, dependency errors, and so on. Finally, we construct a fine-grained error taxonomy consisting of three dimensions and 14 specific error types. Specific error types and their descriptions are provided in Appendix C. We further report the empirical distribution of these errors across models of different parameter scales in Appendix D, showing that they are not artificial annotations but naturally occurring

| Family | Model Name | LEVEL-1 | | | | | LEVEL-2 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | SI | DR | ES | Avg. | Pass@1 | SI | DR | ES | Avg. | Pass@1 |
| OpenAI | GPT-4o | 56.08 | 59.10 | 39.16 | 51.45 | 92.21 | 47.90 | 53.73 | 32.59 | 44.74 | 91.50 |
| | GPT-4o mini | 49.91 | 46.20 | 29.83 | 41.98 | 89.32 | 44.85 | 40.41 | 25.45 | 36.90 | 88.84 |
| | GPT-5 nano | 43.88 | 55.31 | 25.07 | 41.42 | 89.50 | 40.53 | 51.61 | 22.56 | 38.23 | 88.92 |
| | GPT-5.2 | 57.51 | **81.81** | 41.21 | 60.24 | 92.61 | 55.24 | **80.75** | 41.61 | 59.20 | 91.84 |
| Anthropic | Claude Sonnet 4.6 | 69.12 | 74.36 | 50.27 | 64.58 | 94.10 | 65.84 | 71.22 | 48.63 | 61.90 | 93.50 |
| | Claude Opus 4.6 | 73.56 | 78.94 | 55.18 | **69.23** | **94.78** | 70.38 | 76.41 | 53.76 | **66.85** | **94.05** |
| Google | Gemini 3.1 flash | 68.35 | 65.87 | 47.19 | 60.47 | 93.82 | 63.50 | 61.13 | 45.04 | 56.56 | 93.25 |
| | Gemini 3.1 pro | 66.90 | 61.72 | 48.23 | 58.95 | 93.68 | 63.22 | 62.08 | 47.18 | 57.49 | 93.22 |
| Qwen | Qwen 3 8B | 47.61 | 57.08 | 25.72 | 43.47 | 91.84 | 45.41 | 51.12 | 26.69 | 41.07 | 91.37 |
| | Qwen 3 32B | 52.26 | 56.94 | 31.17 | 46.80 | 92.08 | 50.42 | 54.55 | 31.94 | 45.64 | 91.86 |
| | Qwen 3.5 9B | 56.58 | 72.78 | 38.35 | 55.90 | 92.21 | 53.12 | 68.65 | 37.27 | 52.98 | 92.14 |
| | Qwen 3.5 122B-A10B | 72.48 | 77.87 | 52.28 | 67.54 | 94.50 | 69.12 | 74.77 | 51.34 | 65.08 | 93.04 |
| ZHIPU | GLM 4.5 Air | 62.46 | 61.03 | 42.62 | 55.37 | 93.50 | 58.43 | 59.33 | 41.67 | 53.14 | 92.72 |
| | GLM 4.6 | 66.20 | 55.91 | 47.80 | 56.64 | 93.81 | 63.42 | 52.68 | 48.47 | 54.86 | 93.06 |
| DeepSeek | DeepSeek-V4-flash | 73.74 | 75.80 | 54.57 | 68.04 | 94.13 | 71.04 | 73.82 | 54.23 | 66.36 | 93.17 |
| | DeepSeek-V4-flash* | 74.23 | 74.53 | 56.49 | 68.42 | 94.67 | 71.50 | 72.90 | 55.30 | 66.57 | 93.86 |
| | DeepSeek-V4-pro | **74.30** | 72.35 | **58.21** | 68.29 | 94.13 | **71.47** | 71.10 | **57.03** | 66.50 | 93.01 |

Table 2: Performance of different models on LEVEL-1 and LEVEL-2 tasks

failure patterns in realistic agent executions. We then use this unified error taxonomy as the basis for subsequent trajectory perturbation and error-aware data augmentation.

### 2.1.3 Data Composition

Our benchmark consists of two complementary sources: manually curated real rollout trajectories with errors and augmented error trajectories constructed by injecting the 14 predefined error types into correct trajectories. The former ensures that the benchmark reflects naturally occurring failure patterns in realistic agent rollouts, while the latter enables controlled expansion of error diversity and difficulty.

Based on the propagation range of errors, we organize the benchmark into two difficulty levels, namely LEVEL-1 and LEVEL-2. LEVEL-1 contains both manually selected and augmented single-round error trajectories, where the error is restricted to the immediately preceding round, either in the planning result or in the tool return. In this setting, the error has not yet propagated across multiple rounds, and the agent is mainly required to identify and correct a recent error.

In contrast, LEVEL-2 is designed to capture more difficult failure cases. In practice, early errors are often not detected immediately. Instead, they continue to affect later execution. Moreover, an agent may introduce new errors in different rounds, and the accumulation of errors can jointly cause the final failure. LEVEL-2 is intended to model such conditions: multiple co-existing errors and cross-round error propagation. Specifically, from real rollout trajectories, we extract instances where errors are not immediately resolved but propagate to subsequent rounds, or where new errors are introduced later. For augmented data, we randomly inject various error types into the downstream steps of already flawed trajectories. So this level evaluates not only whether the agent can detect an error, but also recover under long-horizon corrupted contexts. Detailed data distributions are provided in Appendix F.

### 2.1.4 The SDE Rubric

To overcome the limitations of traditional binary metrics in evaluating complex agent trajectories, we propose a fine-grained, multi-dimensional evaluation framework: the **SDE** (**S**tructural, **D**iagnostic, and **E**volutionary) Rubric. These three sub-dimensions correspond respectively to multi-turn parallel execution, error localization, and replanning capabilities.

**Structural Integrity (S)** This dimension primarily evaluates the formal correctness and executability of the generated DAG, and is divided into 5 sub-dimensions ($S1 - S5$).

**S1**: Thought-DAG Consistency (1/0.5/0): Evaluates whether the reasoning expressed in thought $\tau$ is perfectly mapped onto the generated DAG $\theta(G)$.

**S2**: Topological Dependency (1/0.5/0): Assesses

| Error Type | LEVEL-1 | | | LEVEL-2 | | |
|---|---|---|---|---|---|---|
| | DR | SI | ES | DR | SI | ES |
| **Structural-related Errors** | | | | | | |
| Dependence_Error | 34.89 | 46.89 | 25.22 | 33.71$_{\downarrow 3.4\%}$ | 43.80$_{\downarrow 6.6\%}$ | 24.85$_{\downarrow 1.5\%}$ |
| Dependence_Null | 39.34 | 48.15 | 22.20 | 34.61$_{\downarrow 12.0\%}$ | 44.07$_{\downarrow 8.5\%}$ | 23.05$_{\uparrow 3.8\%}$ |
| Parallelism_Error | 34.19 | 54.26 | 29.79 | 32.53$_{\downarrow 4.9\%}$ | 47.48$_{\downarrow 12.5\%}$ | 26.83$_{\downarrow 9.9\%}$ |
| Serial_Error | 41.26 | 49.78 | 26.24 | 33.99$_{\downarrow 17.6\%}$ | 48.09$_{\downarrow 3.4\%}$ | 26.26$_{\uparrow 0.01\%}$ |
| **Parameter-related Errors** | | | | | | |
| Parameter_Error | 57.35 | 51.95 | 32.80 | 48.82$_{\downarrow 14.9\%}$ | 44.88$_{\downarrow 13.6\%}$ | 27.05$_{\downarrow 17.5\%}$ |
| Parameter_Invalid | 49.63 | 49.21 | 29.95 | 45.67$_{\downarrow 8.0\%}$ | 44.82$_{\downarrow 8.9\%}$ | 28.51$_{\downarrow 4.8\%}$ |
| Parameter_Missing | 57.93 | 49.63 | 31.90 | 48.53$_{\downarrow 16.2\%}$ | 45.86$_{\downarrow 7.6\%}$ | 29.81$_{\downarrow 6.6\%}$ |
| Type_Error | 55.19 | 49.04 | 31.56 | 46.68$_{\downarrow 15.4\%}$ | 47.71$_{\downarrow 2.7\%}$ | 31.18$_{\downarrow 1.2\%}$ |
| **Tool-related Errors** | | | | | | |
| Null_Error | 46.12 | 44.48 | 23.88 | 37.74$_{\downarrow 18.2\%}$ | 37.20$_{\downarrow 16.4\%}$ | 17.14$_{\downarrow 28.2\%}$ |
| Timeout_Error | 44.17 | 41.98 | 21.57 | 37.28$_{\downarrow 15.6\%}$ | 37.27$_{\downarrow 11.2\%}$ | 16.39$_{\downarrow 24.0\%}$ |
| Tool_name_Error | 41.91 | 35.03 | 16.80 | 35.61$_{\downarrow 15.0\%}$ | 34.99$_{\downarrow 0.1\%}$ | 17.72$_{\uparrow 5.5\%}$ |
| Tool_select_Error | 42.07 | 52.66 | 28.98 | 40.38$_{\downarrow 4.0\%}$ | 44.64$_{\downarrow 15.2\%}$ | 22.64$_{\downarrow 21.9\%}$ |
| Toolcall_Missing | 35.81 | 54.35 | 26.65 | 37.11$_{\uparrow 3.6\%}$ | 32.36$_{\downarrow 40.5\%}$ | 16.91$_{\downarrow 36.6\%}$ |
| Toolcall_Redundancy | 31.85 | 52.01 | 27.06 | 27.99$_{\downarrow 12.1\%}$ | 29.79$_{\downarrow 42.7\%}$ | 18.36$_{\downarrow 32.2\%}$ |

Table 3: Performance of GPT-4o-mini across different error types

the correctness of node dependencies, ensuring that parallel nodes have no interdependencies and sequential nodes have valid causal links.

**S3**: Tool Call Legality (1/0.5/0): Verifies the validity of the invoked tools and their parameters.

**S4**: State Advancement Correctness (1/0.5/0): Checks whether the status attributes of all nodes are accurately updated based on the prior execution feedback.

**S5**:Topological Parsability (1/0): Determines whether the generated DAG structure can be successfully parsed by the execution engine.

**Diagnostic Reasoning (D)** This dimension assesses the depth and accuracy of the agent's failure analysis, and is divided into 5 sub-dimensions ($D1 - D5$).

**D1**: Plan Evidence Anchoring (1/0.5/0): Evaluates whether the thought $\tau$ demonstrates a correct understanding and accurate referencing of the previous round's planning graph.

**D2**: Tool Evidence Anchoring (1/0.5/0): Assesses whether the thought $\tau$ accurately interprets the methods and execution results of previous tool calls.

**D3**: Root Cause Localization Depth (1/0.5/0): Determines if the reasoning identifies the fundamental root cause of the failure rather than merely describing surface-level symptoms.

**D4**: Impact Scope Recognition (1/0.5/0): Checks if the agent successfully identifies downstream nodes and dependency chains affected by a specific error.

**D5**: Evidence Sufficiency (1/0.5/0): Evaluates whether the reasoning provides adequate justification while avoiding over-inference.

**Evolutionary Strategy (E)** This dimension measures the efficiency and precision of the agent's replanning, and is divided into 5 sub-dimensions ($E1 - E5$).

**E1**: Modification Minimality (1/0.5/0): Assesses whether the updated plan only alters necessary components without introducing irrelevant revisions.

**E2**: Recovery Closure (1/0.5/0): Evaluates whether the agent not only fixed the explicit point of failure, but also the previous or subsequent faulty workflows associated with it.

**E3**: Goal Persistence (1/0.5/0): Verifies that the new plan remains strictly aligned with the original task objective without goal drift.

**E4**: Strategy Necessity (1/0.5/0): Determines whether the new step has a clear and reasonable necessity.

**E5**: Problem-Solving Efficacy: Quantifies the practical utility of the updated plan when executed in a real-world environment. It measures whether the task can be completed within finite steps and evaluates the margin of difference from the optimal trajectory. For each query $q$, we define the following variables: $G_q$, which is the optimal number of rounds required by the ground truth, $T_q$, which is the actual number of rounds required to complete the task after substituting the current plan into the subsequent workflow, and $S_q \in \{0, 1\}$, which is the final success status, where 1 indicates success and 0 indicates failure. The resulting score

| Error_Type | LEVEL-1 | | | LEVEL-2 | | |
|---|---|---|---|---|---|---|
| | DR | SI | ES | DR | SI | ES |
| **Structural-related Errors** | | | | | | |
| Dependence_Error | 72.56 | 79.22 | 53.37 | $71.71_{\downarrow 1.2\%}$ | $73.45_{\downarrow 7.3\%}$ | $52.23_{\downarrow 2.1\%}$ |
| Dependence_Null | 74.02 | 76.09 | 50.87 | $72.75_{\downarrow 1.7\%}$ | $70.88_{\downarrow 6.8\%}$ | $51.60_{\uparrow 1.4\%}$ |
| Parallelism_Error | 71.12 | 81.06 | 56.12 | $69.46_{\downarrow 2.3\%}$ | $73.14_{\downarrow 9.8\%}$ | $55.44_{\downarrow 1.2\%}$ |
| Serial_Error | 73.55 | 78.42 | 55.86 | $71.77_{\downarrow 2.4\%}$ | $70.74_{\downarrow 9.8\%}$ | $54.09_{\downarrow 3.2\%}$ |
| **Parameter-related Errors** | | | | | | |
| Parameter_Error | 84.32 | 67.11 | 54.39 | $80.47_{\downarrow 4.6\%}$ | $70.08_{\uparrow 4.4\%}$ | $55.50_{\uparrow 2.0\%}$ |
| Parameter_Invalid | 79.74 | 72.11 | 54.21 | $79.03_{\downarrow 0.9\%}$ | $72.26_{\uparrow 0.2\%}$ | $56.82_{\uparrow 4.8\%}$ |
| Parameter_Missing | 81.80 | 63.92 | 52.59 | $79.84_{\downarrow 2.4\%}$ | $68.22_{\uparrow 6.7\%}$ | $54.36_{\uparrow 3.4\%}$ |
| Type_Error | 83.10 | 68.40 | 55.24 | $79.68_{\downarrow 4.1\%}$ | $67.15_{\downarrow 1.8\%}$ | $56.04_{\uparrow 1.4\%}$ |
| **Tool-related Errors** | | | | | | |
| Null_Error | 73.13 | 72.24 | 54.81 | $70.77_{\downarrow 3.2\%}$ | $70.71_{\downarrow 2.1\%}$ | $51.21_{\downarrow 6.6\%}$ |
| Timeout_Error | 71.51 | 68.07 | 53.21 | $70.74_{\downarrow 1.1\%}$ | $71.57_{\uparrow 5.1\%}$ | $52.50_{\downarrow 1.3\%}$ |
| Tool_name_Error | 76.89 | 63.77 | 52.75 | $74.71_{\downarrow 2.8\%}$ | $64.35_{\uparrow 0.9\%}$ | $52.84_{\uparrow 0.2\%}$ |
| Tool_select_Error | 75.85 | 81.70 | 56.01 | $73.80_{\downarrow 2.7\%}$ | $72.21_{\downarrow 11.6\%}$ | $53.74_{\downarrow 4.1\%}$ |
| Toolcall_Missing | 74.03 | 81.34 | 58.98 | $72.39_{\downarrow 2.2\%}$ | $75.86_{\downarrow 6.7\%}$ | $53.98_{\downarrow 8.5\%}$ |
| Toolcall_Redundancy | 69.31 | 83.44 | 55.53 | $66.13_{\downarrow 4.6\%}$ | $77.41_{\downarrow 7.2\%}$ | $52.84_{\downarrow 4.9\%}$ |

Table 4: Performance of DeepSeek-V4-flash across different error types

$Score(E5)$ for the query is defined as:

$$Score(E5) = S_q \cdot \min\left(1, \frac{G_q}{T_q}\right). \quad (4)$$

## 3 Experiment

We comprehensively evaluate agents' reflection and decision-making capabilities under erroneous states on our benchmark. Specifically, we focus on the following research questions:

**RQ1**: How do mainstream LLMs perform on our benchmark?

**RQ2**: How does agent performance vary across different types of errors?

**RQ3**: Is it reliable to evaluate abstract rubrics using a LLM-as-a-Judge paradigm?

**RQ4**: Can fine-grained evaluation metrics provide actionable guidance for improving model capabilities?

### 3.1 RQ1: How do mainstream large language models perform on our benchmark?

We benchmark six major model families: (1) the OpenAI models, including GPT-4o (Hurst et al., 2024), GPT-4o-mini (Hurst et al., 2024), GPT-5-nano (Singh et al., 2026), and GPT-5.2 (OpenAI, 2025); (2) the Anthropic models, including Claude Sonnet 4.6 (Anthropic, 2026b) and Claude Opus 4.6 (Anthropic, 2026a); (3) the Google Gemini models, including Gemini-3.1-Flash (DeepMind, 2026a) and Gemini-3.1-Pro (DeepMind, 2026b); (4) the Qwen models, including Qwen3-8B, Qwen3-32B (Yang et al., 2025), Qwen3.5-9B, and Qwen3.5-122B-A10B (Team, 2026); (5) the Zhipu GLM models, including GLM-4.5-Air (Zeng et al., 2025) and GLM-4.6 (Organization, 2025); and (6) the DeepSeek models, including DeepSeek-V4-Flash, DeepSeek-V4-Flash*(thinking mode), and DeepSeek-V4-Pro (DeepSeek-AI, 2026).

**Settings.** For all models, we adopt a consistent CoT prompt (see Appendix E) to guide model reasoning and decision-making. Evaluations are conducted under the SDE rubric, with results reported in Table 2. In the table, SI, DR, and ES denote the average scores for Structural Integrity, Diagnostic Reasoning, and Evolutionary Strategy, respectively. The five sub-dimensions within each category are equally weighted. All reported scores are normalized to the 0~100 scale for ease of interpretation. Meanwhile, we report the pass@1 metric, where the updated plan will be adopted in subsequent rounds. A score of 1 is assigned upon successful task completion within specified steps (see Appendix H), and 0 otherwise.

The results show that mainstream LLMs achieve average scores below 70 on our benchmark. Moreover, models with larger parameter sizes substantially outperform smaller ones, indicating that our evaluation effectively captures differences in model capability. The generally low scores also suggest that state-of-the-art models still struggle with attribution and decision-making in multi-turn parallel tasks involving errors. Models consistently achieve higher scores across all dimensions on LEVEL-1 than on LEVEL-2, demonstrating that multi-turn errors pose a greater challenge than single-turn errors. Among all models, Claude Opus 4.6 achieves

| Dimension | EMA | MAE | Human Mean | LLM Mean |
|---|---|---|---|---|
| Diagnostic Reasoning | | | | |
| D1 | 0.8225 | 0.1062 | 0.6363 | 0.6700 |
| D2 | 0.8125 | 0.1100 | 0.5550 | 0.5725 |
| D3 | 0.8275 | 0.1100 | 0.5725 | 0.5775 |
| D4 | 0.8250 | 0.1162 | 0.5300 | 0.5688 |
| D5 | 0.8100 | 0.0950 | 0.3250 | 0.3950 |
| Evolutionary Strategy | | | | |
| E1 | 0.8125 | 0.1150 | 0.4475 | 0.4625 |
| E2 | 0.8025 | 0.0988 | 0.2350 | 0.2463 |
| E3 | 0.7950 | 0.1088 | 0.4238 | 0.3925 |
| E4 | 0.7925 | 0.1338 | 0.4963 | 0.5100 |
| Structural Integrity | | | | |
| S1 | 0.8275 | 0.1075 | 0.5125 | 0.5150 |
| S2 | 0.7875 | 0.1425 | 0.4750 | 0.4725 |
| S3 | 0.7875 | 0.1562 | 0.4938 | 0.4775 |
| S4 | 0.7850 | 0.1700 | 0.4538 | 0.3963 |
| Overall | 0.8067 | 0.1208 | 0.4736 | 0.4813 |

Table 5: Consistency between human annotations and LLM-as-a-Judge

the highest average scores on both levels, obtaining 69.23 and 66.85, respectively. Further analysis reveals that low scores are most frequently observed in the Evolutionary Strategy dimension, which measures a model's ability to replan effectively in response to errors. While models tend to be relatively proficient at localizing errors, they show limited capacity for generating precise and efficient plan updates.

We can also observe that Pass@1 of all models is above 89%, indicating that the plans generated by agents can mostly complete the tasks in subsequent rounds. However, all models score poorly on ES (where E5 measures the deviation from optimal execution rounds), revealing severe inefficiencies caused by redundant calls and execution errors. This highlights that final-outcome metrics alone fail to capture intermediate planning quality, whereas our evaluation system successfully identifies these process-level deficiencies. For further details, please refer to Appendix B

### 3.2 RQ2: How does agent performance vary across different types of errors?

To investigate which types are difficult to localize or recover, we evaluate two representative models with contrasting parameter scales—GPT-4o-mini (small) and DeepSeek-V4-flash (large)—on tasks labeled with different error types. Experimental results are presented in Table 3 and Table 4, reporting overall dataset metrics for LEVEL-1 (single-turn) and LEVEL-2 (multi-turn). Red/green arrows indicate the relative change of metrics on LEVEL-2 compared to LEVEL-1.

Overall, multi-turn errors are more challenging than single-turn errors for both small and large models. Notably, DeepSeek-V4-flash shows relatively small changes from LEVEL-1 to LEVEL-2 across most metrics, with some dimensions even showing slight improvements, whereas GPT-4o-mini suffers substantial drops on most dimensions. This suggests that models with stronger reasoning capabilities can better handle multi-turn error scenarios and remain robust.

| Model | LEVEL-1 | | LEVEL-2 | |
|---|---|---|---|---|
| | DeepSeek | GPT-4o | DeepSeek | GPT-4o |
| GPT-4o | 51.45 | 52.31 | 44.74 | 45.41 |
| Claude Opus 4.6 | 69.23 | 69.20 | 66.85 | 67.24 |
| Qwen3-8B | 43.47 | 43.76 | 41.07 | 41.34 |
| DeepSeek-V4-flash | 68.04 | 68.47 | 66.36 | 67.67 |
| DeepSeek-V4-pro | 68.29 | 68.87 | 66.50 | 67.36 |

Table 6: Cross-judge comparison of average SDE scores using DeepSeek-V4-flash thinking and GPT-4o as judges.

For both models, Toolcall_Redundancy has the lowest scores in the DR dimension, indicating that redundant call is the hardest to identify. Unlike other errors, it typically does not cause explicit execution failures but manifest as "plausible yet ineffective" implicit errors, making them difficult to detect—a common challenge for agents.

Different error types impose distinct challenges. Parameter-related errors yield relatively high scores for two models. However, when difficulty escalates to LEVEL-2, GPT-4o-mini experiences a broad decline, while DeepSeek-V4-flash shows few declines and even improvements. This indicates that large model has become insensitive to this error type, effectively localizing and correcting them in both simple and complex calls.

Structure-related errors and tool-related errors pose significant challenges for both models. Models score relatively higher on parameter-related explicit errors. In real-world scenarios, explicit failures are often easy to resolve, whereas errors that accumulate implicitly—such as redundant calls, state drift, and unreasonable planning—remain difficult to handle.

### 3.3 RQ3: Is it reliable to evaluate abstract rubrics using a LLM-as-a-Judge paradigm?

Many dimensions in our evaluation rubric rely on abstract criteria, making manual annotation impractical. We therefore adopt LLM-as-a-Judge, leveraging the reasoning capability of large language mod-

| Model | LEVEL-1 | | | | LEVEL-2 | | | |
|---|---|---|---|---|---|---|---|---|
| | SI | DR | ES | Avg. | SI | DR | ES | Avg. |
| Base(Qwen3-8B) | 47.35 | 57.75 | 26.48 | 43.86 | 46.33 | 51.06 | 27.72 | 41.70 |
| Few-shot | 48.75 | 57.82 | 26.83 | 44.47 | 47.62 | 51.20 | 27.91 | 42.24 |
| SFT | 49.35 | 58.42 | 27.05 | 44.94 | 48.53 | 51.73 | 27.85 | 42.70 |
| ParaRecover-DPO | **53.42** | **62.71** | **31.16** | **49.10**$_{\uparrow 5.24}$ | **53.02** | **55.27** | **30.95** | **46.41**$_{\uparrow 4.71}$ |

Table 7: Model Performance Comparison

els to automatically score these abstract dimensions. Prior work shows that fine-grained, low-cardinality, and easily discernible rules substantially improve accuracy and consistency (Hashemi et al., 2024; Zhuge et al., 2024), and our three-level (1/0.5/0) scoring per dimension aligns well with this principle.

After evaluating multiple LLMs in preliminary experiments, we selected DeepSeek-V4-flash thinking, which achieved the highest initial agreement with human annotations. For each of the 13 LLM-judged dimensions, we randomly sampled 400 instances, conducted multiple rounds of human annotation, and iteratively refined the evaluation prompt based on this standard. Four metrics are used to measure human-model agreement: EMA, MAE, Human Mean, and Model Mean.

EMA (Exact Match Accuracy) is defined as:

$$\text{EMA} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{\hat{y}_i = y_i}, \tag{5}$$

where $N$ is the total number of samples, $\hat{y}_i$ and $y_i$ are the model-predicted and human-annotated scores, and $\mathbf{1}$ is the indicator function. MAE (Mean Absolute Error) is defined as:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} \left|\hat{y}_i - y_i\right|. \tag{6}$$

As reported in Table 5, the average EMA across all dimensions is 0.8067, and the average MAE is 0.1208, demonstrating that the LLM evaluations achieve a trustworthy level of agreement. The evaluation prompt is provided in the Appendix E.

To further examine the sensitivity to judge choice and potential same-family preference, we additionally calibrate prompt of GPT-4o as an independent judge and use it to re-evaluate representative models. As shown in Table 6, the two judges produce highly consistent scores and overall performance patterns across model families. The LEVEL-1 ranking remains unchanged, while LEVEL-2 shows only minor rank permutations among the top-performing models. These results suggest that our main conclusions are robust to the choice of judge and are unlikely to be driven by same-family preference.

### 3.4 RQ4: Can fine-grained evaluation metrics provide actionable guidance for improving model capabilities?

To evaluate whether the proposed fine-grained intermediate reflection and decision metrics can provide effective supervision to improve agent performance, we use Qwen3-8B as the base model and conduct SFT and DPO (Rafailov et al., 2023), respectively. We first randomly construct a fixed test set. These test instances are excluded from all subsequent training data construction to avoid data leakage. The remaining tasks are used to construct the training data.

**Few-shot Prompting.** To examine how much capability can be elicited through prompting alone, we additionally introduce a few-shot baseline by augmenting the original prompt with three high-quality demonstrations.

**Supervised Fine-Tuning.** For SFT, we collect trajectories generated by multiple models and score them using the SDE evaluation metrics. Trajectories with an average score greater than 10 (15 sub-dimensions in total, with a total score of 15) are selected as candidate high-quality trajectories and are further manually cleaned. This process results in 4,924 SFT training instances.

**ParaRecover-DPO.** Our SDE evaluation framework, as a point-wise scoring rubric, allows us to construct preferences based on score differences, thus making it compatible with preference-learning objectives such as DPO (and variants). Based on the SFT model, we further train with DPO. For DPO, we construct preference pairs from trajectories generated for the same task. The trajectory with a higher total score is treated as the preferred response, while the lower-scoring trajectory is treated as the rejected response. To ensure that

| Model | LEVEL-1 | | | | LEVEL-2 | | | |
|---|---|---|---|---|---|---|---|---|
| | Tool Use Eff.↑ | E5↑ | Invalid DAG Rate↓ | Pass@1↑ | Tool Use Eff.↑ | E5↑ | Invalid DAG Rate↓ | Pass@1↑ |
| Base (Qwen3-8B) | 20.43 | 36.50 | 10.25 | 91.84 | 17.07 | 30.21 | 13.22 | 91.37 |
| SFT | 21.29 | 37.12 | 6.71 | 92.42 | 18.42 | 31.64 | 9.41 | 92.02 |
| ParaRecover-DPO | **26.20** | **40.85** | **5.90** | **94.30** | **21.31** | **33.10** | **9.08** | **93.97** |

Table 8: Judge-free evaluation of Base, SFT, and ParaRecover-DPO models on LEVEL-1 and LEVEL-2. Higher is better for Tool Use Efficiency, E5, and Pass@1, while lower is better for Invalid DAG Rate.

the preference signal is reliable, we require all trajectories to satisfy a minimum threshold 3 and the score gap between the preferred and rejected trajectories to be greater than 5. After manual cleaning, we obtain 3,187 preference pairs for DPO training.

As shown in Tab. 7, few-shot prompting provides moderate improvements over the base model, achieving average scores of 44.47 and 42.24 on LEVEL-1 and LEVEL-2, respectively. This suggests that high-quality demonstrations can partially elicit better recovery behavior without parameter updates. Also, both SFT and ParaRecover-DPO improve the SDE scores, with ParaRecover-DPO achieving the best performance on both LEVEL-1 and LEVEL-2.

Since the training data are constructed using the SDE rubric, however, improvements under the same rubric alone may not provide fully independent evidence of enhanced recovery capability. We therefore further evaluate the models using judge-free, execution-based metrics, including tool-use efficiency, execution-round efficiency (E5), invalid DAG rate, and Pass@1. These metrics are computed directly from execution traces or environment outcomes and do not involve the LLM judge.

As shown in Tab. 8, ParaRecover-DPO consistently improves all judge-free metrics across both difficulty levels. On LEVEL-1, tool-use efficiency increases from 20.43 to 26.20 and E5 from 36.50 to 40.85, while the invalid DAG rate decreases from 10.25 to 5.90 and Pass@1 rises from 91.84 to 94.30. Similar gains are also observed on LEVEL-2. These judge-independent results provide further evidence that SDE-guided preference learning improves actual execution and recovery behavior rather than merely fitting the judge's evaluation preferences.

## 4 Conclusion

We present ParaRecover, a novel process-level benchmark for evaluating error localization and recovery in parallel tool-use agents. Moving beyond previous benchmarks, our framework introduces a fine-grained error taxonomy, challenging error trajectories, and the SDE rubric to measure agent capabilities. Experimental results reveal that high task completion rates do not necessarily imply reliable recovery and replanning capabilities, especially under multi-turn error propagation. Furthermore, we demonstrate that SDE-based supervision can effectively improve agents' reflective recovery behaviors.

## 5 Limitation

While ParaRecover offers a controlled and fine-grained framework for evaluating process-level error recovery in parallel tool-use agents, it has several limitations. First, our benchmark is built on a simulated execution environment rather than live real-world APIs. Although this design ensures reproducibility and controlled error injection, it may not fully capture the unpredictability, latency, and dynamic state changes inherent in real-world systems. Second, our SDE rubric employs an LLM-as-a-Judge paradigm to evaluate abstract reasoning dimensions. Despite achieving relatively high agreement with human annotations, this approach may still introduce bias in complex or ambiguous cases. Finally, ParaRecover models agent execution using DAG-structured workflows. While this formulation effectively captures dependency relationships in parallel multi-turn scenarios, it does not fully represent more dynamic real-world patterns such as loops, conditional branching, or asynchronous interactions. Future work may further improve ParaRecover through more realistic execution environments, hybrid human-AI evaluation protocols, and more general execution structures.

## 6 Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant 62276044, and also in part by the 2025 Scientific Research Projects of the General Administration of Customs of China under Grant No. 2025HK184 and 2025HK209.

## References


Anthropic. 2026a. Claude Opus 4.6. https://www.anthropic.com/news/claude-opus-4-6. Accessed: 2026-05-16.

Anthropic. 2026b. Claude Sonnet 4.6. https://www.anthropic.com/claude/sonnet.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Wang Xinzhi, and Wu Liu. 2025. ACEBench: A comprehensive evaluation of LLM tool usage. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 12970–12998, Suzhou, China. Association for Computational Linguistics.

Mingyang Chen, Haoze Sun, Tianpeng Li, Fan Yang, Hao Liang, Keer Lu, Bin Cui, Wentao Zhang, Zenan Zhou, and Weipeng Chen. 2024a. Facilitating multi-turn function calling for llms via compositional instruction tuning. *arXiv preprint arXiv:2410.12952*.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024b. T-eval: Evaluating the tool utilization capability of large language models step by step. *Preprint*, arXiv:2312.14033.

Google DeepMind. 2026a. Gemini 3.1 Flash. https://deepmind.google/models/gemini/flash/.

Google DeepMind. 2026b. Gemini 3.1 Pro. https://deepmind.google/models/gemini/pro/.

DeepSeek-AI. 2026. Deepseek-v4 collection. https://huggingface.co/collections/deepseek-ai/deepseek-v4. Accessed: 2026-05-16.

Zikang Guo, Benfeng Xu, Chiwei Zhu, Wentao Hong, Xiaorui Wang, and Zhendong Mao. 2026. Mcp-agentbench: Evaluating real-world language agent performance with mcp-mediated tools. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30888–30896.

Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. 2024. Llm-rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13806–13834.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890.

Pengfei He, Zhenwei Dai, Bing He, Hui Liu, Xianfeng Tang, Hanqing Lu, Juanhui Li, Jiayuan Ding, Subhabrata Mukherjee, Suhang Wang, Yue Xing, Jiliang Tang, and Benoit Dumoulin. 2025. Traject-bench:a trajectory-aware benchmark for evaluating agentic tool use. *Preprint*, arXiv:2510.04550.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. 2024. Infiagent-dabench: Evaluating agents on data analysis tasks. *Preprint*, arXiv:2401.05507.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, and 399 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 3102–3116.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2025. Agentbench: Evaluating llms as agents. *Preprint*, arXiv:2308.03688.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *Preprint*, arXiv:2408.04682.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J

Pal, and Siva Reddy. 2025. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*.

Yubo Ma, Zhibin Gou, Junheng Hao, Ruochen Xu, Shuohang Wang, Liangming Pan, Yujiu Yang, Yixin Cao, and Aixin Sun. 2024. Sciagent: Tool-augmented language models for scientific reasoning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 15701–15736.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pages 9025–9049.

OpenAI. 2025. Introducing gpt-5.2. https://openai.com/zh-Hans-CN/index/introducing-gpt-5-2/.

ZAI Organization. 2025. GLM-4.6. https://huggingface.co/zai-org/GLM-4.6. Accessed: 2026-05-16.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

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. *Preprint*, arXiv:2307.16789.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

Shuo Ren, Can Xie, Pu Jian, Zhenjiang Ren, Chunlin Leng, and Jiajun Zhang. 2025. Towards scientific intelligence: A survey of llm-based scientific agents. *arXiv preprint arXiv:2503.24047*.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 467 others. 2026. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Qwen Team. 2026. Qwen3.5: Towards Native Multimodal Agents. https://qwen.ai/blog?id=qwen3.5.

Jiaxuan Wang, Yulan Hu, Wenjin Yang, Zheng Pan, Xin Li, and Lan-Zhe Guo. 2026. Aligning agents via planning: A benchmark for trajectory-level reward modeling. *arXiv preprint arXiv:2604.08178*.

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

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, and Eugene Siow. 2025. Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers. *Preprint*, arXiv:2508.20453.

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.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

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

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen

Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, and 151 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *Preprint*, arXiv:2508.06471.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. *Preprint*, arXiv:2307.13854.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36:50117–50143.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. 2024. Agent-as-a-judge: Evaluate agents with agents. *Preprint*, arXiv:2410.10934.

# A Related Work

## A.1 LLM Agents

LLM agents extend language models from text generators to goal-driven systems that can plan, act and interact with external environments (Yao et al., 2022; Shinn et al., 2023). An important direction is tool-augmented agents, where LLMs learn to invoke external search engines, calculators or domain-specific tools to overcome the limitations of parametric knowledge (Schick et al., 2023; Qin et al., 2023). These design have made agents effective and flexible across domains. For example, research agents automate literature collection, scientific experiments, and paper writing (Ren et al., 2025); Code agents iteratively edit and debug in complex projects (Yang et al., 2024; Hong et al., 2024); Web agents interact with browsers and operating systems to complete complex tasks (Zhou et al., 2024; He et al., 2024). These scenarios involve complex tool calls and long-range decision-making, therefore establishing high-quality benchmarks is crucial for expanding the capabilities of LLM agents.

## A.2 Agent Tool-use and Trajectory Evaluation.

Existing tool-use benchmarks mainly evaluate tool selection, parameter filling, and task completion. API-Bank and Tool-Bench evaluate API retrieval, and execution over tool-use tasks (Li et al., 2023; Qin et al., 2023), while BFCL focuses on function-call correctness across real-world APIs, including serial and parallel function calls (Patil et al., 2025). More recent benchmarks such as GAIA, MCP-Bench and MCP-AgentBench extend evaluation to multi-step tool use, cross-tool coordination, protocol-mediated environments, and realistic agent workflows (Mialon et al., 2024; Wang et al., 2025; Guo et al., 2026). However, these benchmarks are still mainly outcome-oriented or call-level oriented, making it difficult to diagnose where an agent fails within a long trajectory.

| Model | LEVEL-1 | | LEVEL-2 | |
|---|---|---|---|---|
| | E5 | Pass@1 | E5 | Pass@1 |
| GPT-4o | 42.19 | 92.21 | 40.26 | 91.50 |
| GPT-4o-mini | 36.17 | 89.32 | 30.05 | 88.84 |
| Claude Opus 4.6 | 60.61 | 94.78 | 57.50 | 94.05 |
| Gemini 3.1 pro | 52.81 | 93.68 | 49.32 | 93.22 |
| Qwen 3 8B | 36.50 | 91.84 | 30.21 | 91.37 |
| Qwen 3 32B | 39.94 | 92.08 | 34.10 | 91.86 |
| DeepSeek-V4-flash | 57.42 | 94.13 | 55.68 | 93.17 |

Table 9: E5 score and pass@1 performance of different models on LEVEL-1 and LEVEL-2

Recent trajectory-aware benchmarks attempt to provide more fine-grained evaluation. T-Eval decomposes tool utilization into instruction following, planning, reasoning, retrieval, understanding, and review (Chen et al., 2024b). TRAJECT-Bench evaluates whether tools are selected, parameterized, and ordered correctly along trajectories (He et al., 2025). Reward-oriented benchmarks such as AgentRewardBench and Plan-RewardBench evaluate agent trajectory preferences (Lù et al., 2025; Wang et al., 2026). Although these work moves beyond final-answer accuracy, they mainly focus on step correctness, tool-use correctness or trajectory-level preference judgment.

As shown in Table 1, our work differs by targeting error-aware trajectory evaluation in parallel tool-calling scenarios. Instead of only asking whether a tool call is correct, whether the final answer is correct, or which trajectory is preferred, our benchmark evaluates whether an agent can inspect flawed intermediate trajectories, localize erroneous steps, and decide how to recover and replan. This setting complements existing tool-use and trajectory-evaluation benchmarks by focusing on the agent's reflection and recovery decision ability under multi-turn parallel tool-use failures.

## B Analysis of the Discrepancy Between Pass@1 and ES Scores

Table 9 presents the disparity between E5 scores and Pass@1 performance for several representative models across different task levels. We observe that while Pass@1 metrics remain consistently high (over 88%) across all models, E5 scores are generally low (the scores are shown after normalization to 0~100), with significant performance gaps among models of different parameter sizes. Since the E5 metric quantifies the deviation of actual execution rounds from ground-truth rounds, the contrasting trends of these two metrics indicate that although models can complete our benchmark tasks within a limited number of rounds (see Appendix H), their execution processes are plagued by erroneous planning and redundant tool calls, leading to severe inefficiency and high costs.

The case study in Fig. 2 illustrates this phenomenon. In this relatively simple task, Node 3 returns an empty response due to an incorrect parameter type. However, instead of performing valid error attribution and retrying the call, the agent misattributes the failure to an upstream task. Consequently, it creates Node 7 to retry the entire workflow, triggering a cascade of subsequent errors. The actual execution takes 18 rounds, compared to the ground-truth requirement of 6 rounds, resulting in an E5 score of 0.33. Such cases are prevalent, demonstrating that agents struggle with effective error attribution in multi-turn parallel scenarios, which frequently generates redundant steps. This reveals the limitations of relying solely on final task completion rates, underscoring the critical importance of evaluating agents' error localization and replanning capabilities when handling flawed trajectories.

## C Error Description

Table 12 shows the 14 error types and their corresponding descriptions.

## D Error Type Distribution

As shown in Fig. 3, we analyze the error distributions of three different models during real-world execution. The most frequent errors encountered are incorrect parameter types, invalid parameter values, redundant tool calls, and missing critical tool calls. Notably, models with larger parameter counts and stronger reasoning capabilities consistently exhibit lower error rates compared to smaller models. This distribution also demonstrates that all error categories within our taxonomy occur in real-world agent execution scenarios.

## E Related Prompts

In this section, we present prompts used across various stages of our experiments. Specifically, these prompts comprise the DAG-based CoT agent execution planning prompt, the tool agent prompt used to simulate tool execution results in our controlled environment, and the LLM-as-a-Judge evaluation prompts across the three dimensions of the SDE rubric.

**Prompt for Agent**

You are an expert at leveraging various tools to solve problems. You focus on observing the [Front-Wheel Plan] and [Front-Wheel Tool Call Results], and then make appropriate reflections and plans based on the [Original User Query] and [Available Tools List].
## Your task:
1. Analyze the user's original query, combine it with the DAG list of the previous plan and the results of the previous tool calls, and consider what needs to be done to complete the task.
2. Based on the tools that still need to be called, update the current complete task list to guide the entire subsequent process.
## Output requirements
- First, output the <thought> section. Include reflections on the original user task and perform a requirement analysis: first, identify which subtasks have already been completed toward answering the user's question and which subtasks still need to be completed. Then, provide a detailed sequence of steps required to accomplish the original task, along with a sound justification for the rationale behind each step. In your reasoning, include a detailed analysis of the reasonableness of the previous round's planning, as well as a detailed analysis of the reasonableness of the previous round's tool invocation arrangements (including the tools used and their parameters).
- Second, output a parseable JSON representing the task chain list of tool calls. 1. Clearly specify the dependency relationships for each subtask. A dependency exists only if the return result of a preceding task is required to supplement or complete the parameters of a subsequent tool call. For any subtask that does not require preceding information, its dependency should be set to "root". 2. For subtask nodes that have not yet been executed, you are free to modify, add, or delete them as needed to better invoke tools and complete the task. Note that subtask nodes which have already been executed must not be altered. 3. The first subtask node has been determined as the "root" node, with the subtask set to:

{"id": "root", # Root node, representing the starting point of the task. "desc": {query}, # Original query content provided by the user. "dep": [], # The root node does not need to depend on other nodes. "status": True # The root node has already been executed by default.} The last subtask node has been determined as the "response" node, with the subtask set to: {"id": "response", # The final node indicates the end point of the task. "desc": "Analyze and summarize, and then respond.", # Summarize and analyze the results of all subtasks, and then respond. "dep": [...], # All nodes that are not dependent on by other subtasks. "status": False # The last response node has not been executed by default.}
##The output format is as follows (strictly output the reasoning section and the task list JSON. Do not include comments. Please output a compact JSON format, removing all unnecessary line breaks and indentation spaces, while preserving necessary spaces within string content):
<thought>Output your thoughts on the user's current query; you need to re-analyze and rethink it. Conduct a requirements analysis, first analyzing what you have already accomplished to complete the user's original query, and what tools are still needed. Then, detail the steps required to answer the user's query, and reasonably explain the rationale for each step. Your thinking should include a detailed analysis of the rationality of the previous planning, as well as the rationality of the tool usage arrangement (including tools used and parameters). Note that your thinking should not exceed 500 words.</thought>

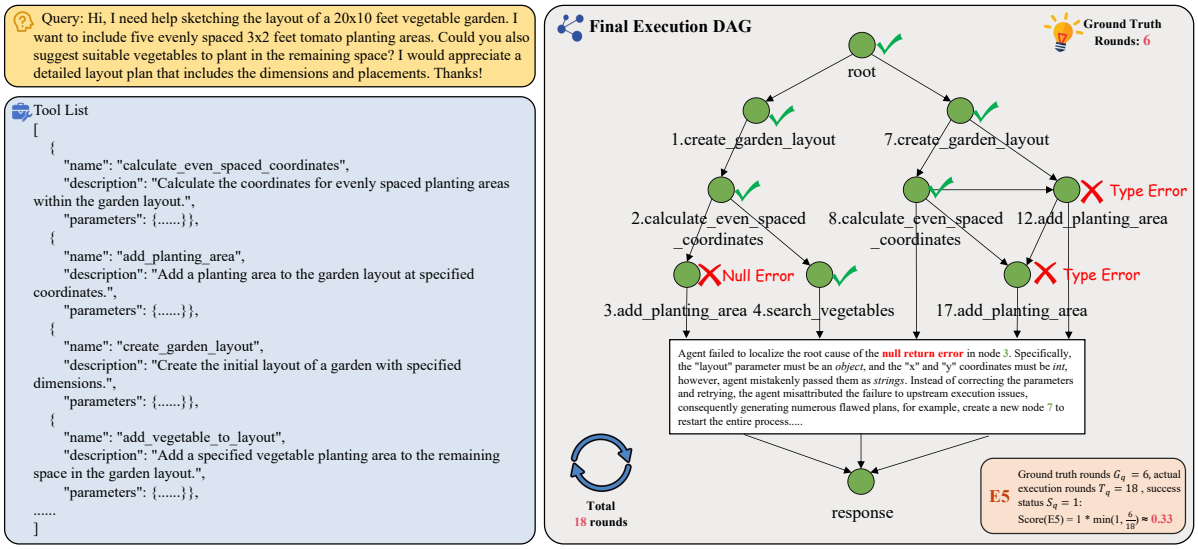


Figure 2: Case study

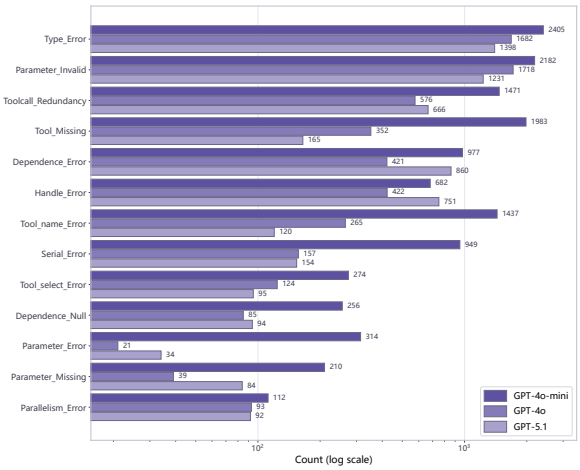


Figure 3: Comparative analysis of error type occurrences across GPT-4o-mini, GPT-4o, and GPT-5.1. The horizontal bars represent the frequency of each error category on a logarithmic scale.

{"tasks": [ # A list of subtasks, where each element is a tool call subtask node, containing the task ID, the name of the called tool, the task description, and dependencies. All subtask nodes must be provided (regardless of whether they have been executed), including the root node and the response subtask node. {"id": string, # A unique identifier for a subtask, which is identified by a number "n", starting from "1". "name": string, # The name of the tool called in the subtask. "para": [string], # The parameters used in the tool call are in the format {"parameter1 name":"parameter1 value","parameter2 name":"parameter2 value", .....}. "desc": string, # Description of task nodes "dep": [string], # The subtask dependencies, where each element in the list is the id of the dependent subtask. "status": bool # This indicates whether the subtask has been executed. True means it has been executed, and False means it has not been executed.}, .....]}
##Original User Query
{query}
##Available Tools List
{tool_list}
##Front-Wheel Plan
{plan_dag}
##Front-Wheel Tool Call Results
{tool_call_dag}

## Prompt for Tool Agent

You are an expert skilled in simulating agent tool invocation. You need to generate simulated execution results based on the User Task and Available Tool List.
##Your Task
Carefully identify the pending executable nodes in the previous planned DAG graph, namely nodes marked False with all dependency requirements satisfied. Verify the validity of tool invocations in the DAG graph. Check whether tool names, parameter names and parameter types comply with standard specifications defined in the Available Tool List. Return reasonable error prompts immediately if any irregularities are detected. If the invocation is valid, generate plausible returned results corresponding to the description and assigned tool of each task node. All simulated results shall be logical and consistent with real-world scenarios. Deliver accurate calculation outputs for computational tools. Return reasonable summarized search contents for retrieval tools. Realistic fabricated values associated with the given task are acceptable.
##DAG Graph Description
The previous plan is presented as a DAG graph starting from the root node and ending at the response node. DAG structure format:
{"tasks": [{"id": string, "name": string, "para": {}, "desc": string, "dep": [string], "status": bool}, ...]} Nodes ready for execution are unexecuted nodes with False status, whose all dependent predecessor nodes have been fully completed with True status. Output Format First write your reasoning process within <thought></thought>.Then output tool responses in standard JSON array format: ’’’json [ {

"name": string, # Consistent with the name field in DAG graph "arguments": {"Parameter1": "Value1"}, # Tool calling parameters "results": string # Simulated tool execution feedback }, ... ]
##User Task
{query}
##Available Tool List
{tool_list}
##Previous Planned DAG Graph
{replan_dag}

## Prompt for Structural Integrity Evaluation

##Role
You are a highly strict, conservative, and fine-grained evaluation expert for replanner decision-making.
##General Requirements
You must evaluate whether the given replanner's thought and replan DAG are truly high-quality.You must default to strict scoring rather than moderate or high scores by default.
##Evaluation Dimension: Structural Integrity
Strictly evaluate the following 4 sub-items judged by LLM from the perspective of plan structure and executability consistency.An additional script-based judgment item will be automatically supplemented in the program:
##S1 Thought-DAG Consistency: Whether the actions, repairs, and progress claimed in the thought are truly reflected in the replan DAG. 1: Key repair actions, downstream processing, and DAG nodes in the thought correspond one-to-one.
0.5: Partially consistent, but with omissions, misalignments, or claims only in the thought without implementation in the DAG.
0: Clearly inconsistent.
S2 Topological Dependency & Overall Executability: Whether node dependencies, execution order, and upstream/downstream relationships are correct, and whether the overall structure forms a real executable DAG. 1: Correct and reasonable dependencies with strong overall executability. 0.5: Minor dependency issues or execution risks exist, but the main chain is generally understandable. 0: Obvious dependency errors or overall difficulty in execution.
S3 Tool Call Legitimacy: Whether tool names, parameter names, and parameter value formats strictly comply with the constraints in the tool list. 1: All tool names, parameter names, and parameter value formats are legal and accurate. 0.5: Minor non-standard formats or potential risks exist, but not completely unusable. 0: Invalid tools, wrong parameter names, missing key parameters, or obviously illegal values.
S4 State Progression Correctness: Whether the statuses of executed nodes, unexecuted nodes, retained nodes after failure, and response nodes are reasonable. 1: State annotations are consistent with the actual execution status. 0.5: Minor non-standard state annotations. 0: Obvious state errors that will mislead the executor.
##Output Format
Output ONLY a JSON object, NO extra content: "'json
{ "S1_score": "1 / 0.5 / 0", "S1_reason": "Reason for this score", "S2_score": "1 / 0.5 / 0", "S2_reason": "Reason for this score", "S3_score": "1 / 0.5 / 0", "S3_reason": "Reason for this score", "S4_score": "1 / 0.5 / 0", "S4_reason": "Reason for this score", }
"'
##Input
##Original User Task
{query}
##Available Tool List
{tool_list}
##Previous Plan
{prev_plan} ##Execution Results & Tool Responses of Previous Plan
{tool_result}
##Output of ReplannerThought
{thought}
##Updated Plan
{new_plan}

## Prompt for Diagnostic Reasoning Evaluation

##Role
You are a highly strict, conservative, and fine-grained evaluation expert for replanner decision-making.
##General Requirements
You must evaluate whether the given replanner's thought and replan DAG are truly high-quality.You must default to strict scoring rather than moderate or high scores by default.
##Evaluation Dimension: Diagnostic Reasoning
Strictly assess the following five sub-indicators from the perspective of diagnostic quality.
D1 Plan Evidence AnchoringWhether the thought explicitly references and accurately understands specific nodes, dependencies, statuses or steps in the previous plan. 1: Clearly cites verifiable details such as node IDs, upstream and downstream nodes and status values, and conducts problem analysis based on valid evidence. 0.5: Contains partial references but fails to specify verifiable node IDs, dependencies or statuses, or lacks precise supporting evidence. 0: Barely relies on plan evidence and only provides general descriptions.
D2 Tool Evidence AnchoringWhether the thought explicitly references and correctly interprets specific error messages, return values, parameter names or tool behaviors from tool results. 1: Precisely specifies exact tool names, parameter names, error fields or return values, and explains their implications. 0.5: Detects existing tool abnormalities but fails to reference specific parameters or error details, or presents incomplete evidence. 0: Makes no reference to concrete tool outputs or contains obvious misinterpretation.
D3 Root Cause Localization AccuracyWhether the actual fundamental cause is identified instead of merely describing superficial symptoms. 1: Points out failure manifestations and accurately locates essential causes including invalid parameters, faulty dependencies and improper state transition. 0.5: Roughly pinpoints problematic sections but stays at superficial descriptions such as failed execution and parameter adjustment demand. 0: Delivers incorrect root cause judgment or fails to identify underlying reasons.
D4 Impact Scope RecognitionWhether subsequent nodes, dependency chains, state transitions and response generation affected by errors are recognized. 1: Clearly names at least one affected downstream node, relevant dependency chain or abnormal response progress. 0.5: Acknowledges potential impacts but fails to specify affected nodes or covers incomplete influence scope. 0: Fails to identify any affected scope.
D5 Evidence Sufficiency and RestraintWhether reasoning is sufficiently supported by facts without fabricating non-existent information. 1: Draws conclusions based on at least two independent verifiable evidence sources such as plan and tool results; actively detects remaining flaws and risks; distinguishes authentic root causes from irrelevant misleading explanations. 0.5: Partially meets requirements but shows noticeable deficiencies in evidence adequacy, false cause elimination or residual risk detection. 0: Lacks solid supporting grounds, contains obvious speculation and fabrication, or lacks rigorous verification awareness.
##Output Format
Output only a single JSON object with no extra content. "'json
{ "D1_score": "1 / 0.5 / 0", "D1_reason": "State the reason for the score", "D2_score": "1 / 0.5 / 0", "D2_reason": "State the reason for the score", "D3_score": "1 / 0.5 / 0", "D3_reason": "State the reason for the score", "D4_score": "1 / 0.5 / 0", "D4_reason": "State the reason for the score", "D5_score": "1 / 0.5 / 0", "D5_reason": "State the reason for the score", }
"'
##Input
##Original User Task
{query}
##Available Tool List
{tool_list}
##Previous Plan
{prev_plan} ##Execution Results & Tool Responses of Previous Plan
{tool_result}
##Output of ReplannerThought
{thought}
##Updated Plan
{new_plan}

## Prompt for Diagnostic Evolutionary Strategy

##Role
You are a highly strict, conservative, and fine-grained evaluation expert for replanner decision-making.
##General Requirements
You must evaluate whether the given replanner's thought and replan DAG are truly high-quality.You must default to strict scoring rather than moderate or high scores by default.
##Evaluation Dimension: Evolutionary Strategy Strictly assess the following five sub-items in terms of the quality of revision and improvement strategies.
E1 Minimal ModificationWhether the replan only revises necessary parts without irrelevant rewriting. 1: Only essential nodes and dependencies are adjusted, and changes directly correspond to

specific errors. 0.5: Revisions follow a general correct direction yet remain superficial, with redundant or excessive adjustments lacking sufficient justification. 0: Uncontrolled modifications made or key issues left unaddressed.
E2 Closed-loop RevisionWhether the revision fixes current errors and complements relevant downstream nodes, parameters, statuses and response chains. 1: A complete closed loop is formed with explicit handling of affected downstream nodes, parameters and statuses. 0.5: Partial errors get fixed, yet gaps remain in downstream links, status transition and response chains. 0: No executable closed loop established.
E3 Goal ConsistencyWhether the updated plan still faithfully fulfills the original user task without deviation, incomplete delivery or missing core objectives. 1: Original goals fully retained with complete pathways reserved for final response generation. 0.5: Basic goals maintained but partial coverage missing or minor deviation exists, and complete final response loop cannot be guaranteed. 0: Noticeable deviation or omission of core objectives.
E4 Strategy RationalityWhether added, removed or reordered steps and dependencies possess clear practical necessity. 1: All strategic adjustments are fully justified and necessary. 0.5: Most adjustments are reasonable while a few changes lack adequate necessity. 0: Critical adjustments serve no practical purpose and appear as mechanical rewriting.

##Output Format
Output only a single JSON object with no extra content.
"'json
{ "E1_score": "1 / 0.5 / 0", "E1_reason": "State the reason for the score", "E2_score": "1 / 0.5 / 0", "E2_reason": "State the reason for the score", "E3_score": "1 / 0.5 / 0", "E3_reason": "State the reason for the score", "E4_score": "1 / 0.5 / 0", "E4_reason": "State the reason for the score", }
##Input
##Original User Task
{query}
##Available Tool List
{tool_list}
##Previous Plan
{prev_plan} ##Execution Results & Tool Responses of Previous Plan
{tool_result}
##Output of ReplannerThought
{thought}
##Updated Plan
{new_plan}

## F Data Distribution

Fig. 4 illustrates the data distribution of our constructed benchmark. The benchmark comprises a total of 10,626 instances categorized into two levels: LEVEL-1 and LEVEL-2. Specifically, LEVEL-1 contains 4,033 instances, including 1,421 error instances collected from real-world rollouts and 2,612 instances generated via synthetic error augmentation on correct trajectories. LEVEL-2 consists of 6,593 instances, comprising 1,391 error instances from real-world rollouts and 5,202 instances derived through error augmentation on correct trajectories. Table 10 and Table 11 present the data distribution across different error types within the LEVEL-1 and LEVEL-2 tasks of ParaRecover. Specifically, LEVEL-1 tasks contain only single-round errors, whereas LEVEL-2 tasks encompass multi-turn error types.

## G Training Configuration

All experiments were conducted on a single server equipped with 8 NVIDIA RTX 3090 GPUs. We adopted Low-Rank Adaptation (LoRA) (Hu et al., 2021) for both the SFT and DPO stages. The model used is Qwen3-8B. The LoRA hyperparameters were uniformly set to a rank of $r = 32$, a scaling factor of $\alpha = 64$, and a dropout rate of 0.05. For the SFT stage, we trained the base model for 3 epochs with a maximum sequence length of 4096 tokens to accommodate multi-turn agent execution trajectories. We employed a cosine learning rate scheduler with a peak learning rate of $1 \times 10^{-4}$ and a 3% warmup ratio. The global batch size was set to 64, achieved through a per-device batch size of 2 and 4 gradient accumulation steps. For the DPO stage, we initialized the policy and reference models with the resulting SFT weights. To prevent overfitting on the preference data, DPO training was limited to 1 epoch. The learning rate was reduced to $5 \times 10^{-6}$ with a cosine scheduler and a 10% warmup ratio. The KL penalty coefficient was set to $\beta = 0.1$, and the global batch size was maintained at 64.

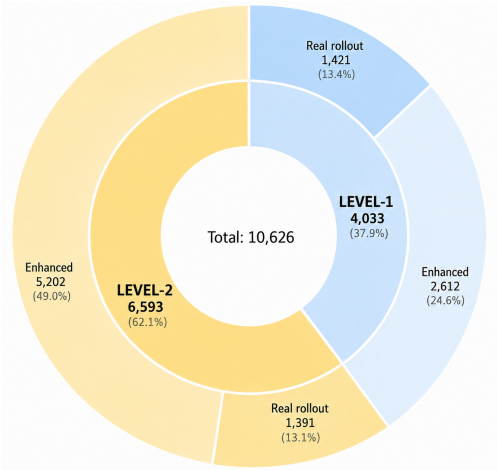


Figure 4: Data distribution

| Error Type | Frequency | Ratio(%) |
|---|---|---|
| Parameter_Missing | 333 | 8.26 |
| Parameter_Invalid | 341 | 8.46 |
| Parameter_Error | 276 | 6.84 |
| Type_Error | 322 | 7.98 |
| Tool_name_Error | 258 | 6.40 |
| Tool_select_Error | 266 | 6.60 |
| Toolcall_Missing | 307 | 7.61 |
| Toolcall_Redundancy | 296 | 7.34 |
| Dependence_Error | 278 | 6.89 |
| Dependence_Null | 252 | 6.25 |
| Serial_Error | 407 | 10.09 |
| Parallelism_Error | 268 | 6.65 |
| Null_Error | 210 | 5.20 |
| Timeout_Error | 219 | 5.43 |
| Total | 4033 | 100.00 |

Table 10: Error type statistics in LEVEL-1

| Error Type | Round1 | Ratio(%) | Round2 | Ratio(%) | Round3 | Ratio(%) |
|---|---|---|---|---|---|---|
| Parameter_Missing | 429 | 6.51 | 443 | 6.72 | 198 | 7.20 |
| Parameter_Invalid | 528 | 8.01 | 544 | 8.25 | 204 | 7.42 |
| Parameter_Error | 428 | 6.49 | 416 | 6.31 | 187 | 6.80 |
| Type_Error | 598 | 9.07 | 553 | 8.39 | 197 | 7.16 |
| Tool_name_Error | 554 | 8.40 | 485 | 7.36 | 181 | 6.58 |
| Tool_select_Error | 435 | 6.60 | 370 | 5.61 | 177 | 6.44 |
| Toolcall_Missing | 558 | 8.46 | 554 | 8.40 | 203 | 7.38 |
| Toolcall_Redundancy | 540 | 8.19 | 530 | 8.04 | 206 | 7.49 |
| Dependence_Error | 449 | 6.81 | 483 | 7.33 | 208 | 7.56 |
| Dependence_Null | 395 | 5.99 | 417 | 6.32 | 203 | 7.38 |
| Serial_Error | 506 | 7.67 | 494 | 7.49 | 191 | 6.95 |
| Parallelism_Error | 353 | 5.35 | 397 | 6.02 | 214 | 7.78 |
| Null_Error | 394 | 5.98 | 456 | 6.92 | 189 | 6.87 |
| Timeout_Error | 426 | 6.46 | 451 | 6.84 | 191 | 6.95 |
| **Total** | 6593 | 100.00 | 6593 | 100.00 | 2750 | 100.00 |

Table 11: Error type statistics in LEVEL-2

## H Maximum Number of Execution Rounds

In our experiments, an agent is considered to have successfully completed a task if it reaches the response node within the maximum number of execution rounds, $r$. We set $r = 40$ because preliminary experiments show that trajectories exceeding 40 rounds typically indicate either an inherent bias in the agent's task comprehension or an infinite loop. Consequently, such tasks are highly unlikely to be completed even if additional rounds are allowed.

| Error Type | Description |
|---|---|
| Dependence_Error | Dependency Relation Error: In the unfinished tasks of the updated DAG generated by the agent, some dependencies between nodes are incorrectly specified. For example, node 3 should depend on node 2, but is mistakenly represented as depending on node 1. |
| Dependence_Null | Missing Dependency Error: In the unfinished tasks of the updated DAG generated by the agent, some nodes have missing dependencies. |
| Parallelism_Error | Incorrect Parallelization: In the unfinished tasks of the updated DAG generated by the agent, some nodes are incorrectly arranged in parallel. Specifically, two nodes that should be executed sequentially are treated as parallel. For example, node 2 requires the execution result of node 1 as its input parameter, but the agent schedules them to run in parallel. |
| Serial_Error | Incorrect Serialization: In the unfinished tasks of the updated DAG generated by the agent, some nodes are incorrectly arranged sequentially. Specifically, two nodes have no dependency on each other and can be executed in parallel, but they are mistakenly represented as having a sequential dependency in the updated DAG. |
| Parameter_Error | Parameter Name Error: In the unfinished tasks of the updated DAG generated by the agent, some tool-call nodes contain invalid parameters that are not defined in the current tool's parameter list. For example, the supported parameter in the parameter list is $begin_date$, but the tool-call node incorrectly provides $begin - date$. |
| Parameter_Invalid | Invalid Parameter Value: In the unfinished tasks of the updated DAG generated by the agent, some tool-call nodes assign parameter values with the correct type but invalid content. For example, the tool requires a $date$ parameter, but the call provides an invalid value such as "2026/02/32". |
| Parameter_Missing | Missing Required Parameter: Each tool is associated with a parameter list and a set of required parameters. In the unfinished tasks of the updated DAG generated by the agent, some tool-call nodes fail to provide required parameters. |
| Type_Error | Parameter Type Error: Each parameter in a tool's parameter list has a predefined value type, such as "int" or "string". In the unfinished tasks of the updated DAG generated by the agent, some tool-call nodes assign values with incorrect types. For example, a parameter requires the integer value 16, but the node provides the string value "16". |
| Null_Error | Empty Tool Response Error: During tool execution, the tool returns an empty result for some reason. |
| Timeout_Error | Timeout Error: During tool execution, the tool call fails with a timeout error due to issues such as network instability. |
| Tool_name_Error | Invalid Tool Name Error: In the unfinished tasks of the updated DAG generated by the agent, some tool-call nodes invoke an invalid tool name that is not included in the current tool list. |
| Tool_select_Error | Incorrect Tool Selection: Some tools in the tool list have similar or easily confused functions. In the unfinished tasks of the updated DAG generated by the agent, some tool-call nodes select an inappropriate tool for the intended operation. |
| Toolcall_Missing | Missing Required Tool Call: In the updated plan DAG generated by the agent, some tool calls required to complete the task are missing. |
| Toolcall_Redundancy | Tool call redundancy: In the unfinished tasks of the updated DAG generated by the agent, there are some unnecessary tool calls. |

Table 12: Error Types and Descriptions