# AGENT4WEAKNESS: An Agentic Framework for In-Depth Model Weakness Discovery

**Anonymous ACL submission**

## Abstract

The rapid evolution of reasoning-intensive Large Language Models renders traditional metrics insufficient by masking fine-grained failures and implicit pathologies. Existing weakness discovery methods typically rely on rigid pipelines, yielding superficial insights that lack the diagnostic depth required for effective model improvement. To address this, we introduce AGENT4WEAKNESS, a multi-agent framework designed to replicate the rigorous workflow of human expert analysts. By integrating a Domain-Aware Memory for contextual reasoning with professional evaluation knowledge and a Tool Abstraction mechanism for decouple data analysis, AGENT4WEAKNESS transforms raw evaluation traces into grounded, actionable reports. We validate our framework through an extensive study involving 104 models across 27 benchmarks. Experimental results demonstrate that AGENT4WEAKNESS produces diagnostic reports significantly superior to competitive baselines. Crucially, leveraging these insights for prompt guidance yields an average 3.7 point performance boost and establishes a closed-loop optimization paradigm[1].

## 1 Introduction

The rapid evolution of Large Language Models (LLMs), especially reasoning-intensive models with massive context windows (e.g., 128k tokens), has turned evaluation into a diagnostic nightmare (OpenAI, 2025; Google, 2025b). While aggregate scores offer a coarse snapshot of performance, they often fail to capture subtle failure patterns within complex reasoning traces that preclude manual auditing. As a result, a high benchmark score can easily hide underlying behavioral flaws, ranging from logic regressions in long-chain thoughts to persistent stylistic biases, such as emoji overuse or repetitive templates.

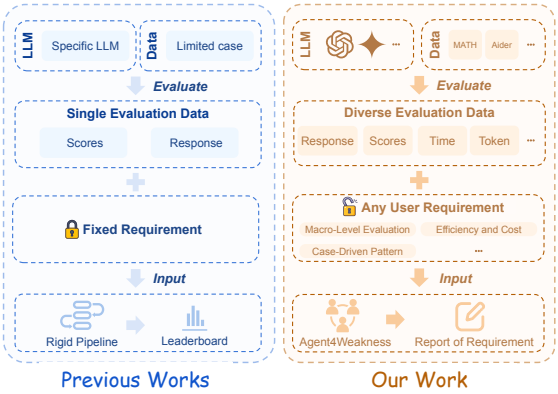


Figure 1: Comparison between existing weakness discovery methods and AGENT4WEAKNESS. **Left:** Previous works typically rely on rigid pipelines constrained by fixed requirements and limited evaluation data, resulting in superficial leaderboards. **Right:** AGENT4WEAKNESS utilizes a multi-agent framework capable of processing diverse data under flexible requirements and generate in-depth reports.

To address this, the task of **weakness discovery** automates the mining of fine-grained model limitations from evaluation data (Moayeri et al., 2025; Tian et al., 2025). Understanding these granular weaknesses is essential for iterative optimization and reliable real-world deployment (Ni et al., 2025). However, as illustrated in Figure 1, existing methods suffer from limited flexibility and depth. Previous works employ rigid pipelines which accept a single metric on specific evaluation datasets and discover predefined weakness, limiting adaptability to diverse requirements and evolving benchmarks (Zeng et al., 2025; Lee et al., 2025). Furthermore, current methods often produce superficial leaderboards rather than the statistically-grounded insights and diagnostic clarity essential for guiding model improvement (Murahari et al., 2024). Such black-box rankings merely reveal where a model fails but offer no insight into why, reducing model iteration to a blind game of trial-and-error. Consequently, researchers remain data-rich but insight-poor, possessing vast evaluation logs yet lacking

[1]Our code will be released upon acceptance.

the actionable depth required for refinement.

To bridge these gaps, we introduce a multi-agent framework, AGENT4WEAKNESS, designed to replicate the rigorous diagnostic workflow of human expert analysts. Rather than merely calculating surface-level metrics, AGENT4WEAKNESS functions as an automated auditor that transforms raw evaluation data into actionable intelligence. To achieve expert-level reasoning, we propose a **Domain-Aware Memory mechanism** that injects evaluation knowledge and execution context into the agents, enabling them to fulfill complex user requirements and analyze vast reasoning traces with professional logic. This is complemented by a Reliable **Tool Abstraction mechanism**, which empowers agents to dynamically invoke data access tools, statistical analysis tools, and behavior probes. By decoupling high-level diagnostic reasoning from low-level data processing, AGENT4WEAKNESS generates both statistically grounded and contextually deep reports, moving beyond simple leaderboards to provide a blueprint for model refinement.

Extensive experiments involving 104 models on 27 benchmarks validate our method. Our method not only accurately detects fine-grained weaknesses at the instance level but also uncovers implicit behavioral flaws that elude traditional evaluation. Also, AGENT4WEAKNESS significantly outperform competitive baselines, achieving a substantial improvement of 2.6 out of 10 points in quality as rated by both LLM-based evaluators and human experts, with high inter-rater consistency. Crucially, these insights prove actionable since applying the discovered weaknesses as prompt-level guidance yields a consistent boost of 3.7 points over the original average of 70.3 across models. This closed-loop mechanism facilitates systematic model self-evolution rather than blind trial-and-error.

Our contributions are as follows:

- We propose AGENT4WEAKNESS, a multi-agent framework that formalizes the weakness discovery task by mimicking expert-level auditing by Domain-Aware Memory and Tool Abstraction.
- We conduct a large-scale analysis on 104 models and 27 datasets, providing a comprehensive diagnostic report that outperforms baselines in flexibility and depth.
- We empirically prove the actionability of our diagnostic insights, demonstrating that the discovered weaknesses can directly enhance model performance, thereby establishing a new paradigm for closed-loop model optimization.

## 2 Methodology

We first formulate the evaluation task and the structure of the evaluation data. We then present AGENT4WEAKNESS, a multi-agent framework designed to automate this process, mimicking the rigor of human experts through our domain-aware memory mechanism and reliable tool abstraction.

### 2.1 Problem Formulation

**Task Definition.** We aim to generate an in-depth analysis report $y$ given a user query $q$ and evaluation data $\mathcal{D}_{\text{eval}}$ as shown in Figure 2.

$$y = \text{AGENT4WEAKNESS}(q, \mathcal{D}_{\text{eval}}) \tag{2.1}$$

**Evaluation Data Structure.** We evaluate a set of 104 models ($\mathcal{M}$) across 27 benchmarks ($\mathcal{B}$) covering seven capability dimensions (details in Appendix B). The evaluation data is formally defined as $\mathcal{D}_{\text{eval}} = (\mathcal{D}_{\text{raw}}, \mathcal{D}_{\text{stat}})$.

(*i*) **Aggregated Statistics ($\mathcal{D}_{\text{stat}}$).** This set comprises pre-computed metrics (e.g., accuracy, runtime). Let $K$ be the number of statistic types. $\mathcal{D}_{\text{stat}}$ is defined as:

$$\mathcal{D}_{\text{stat}} = \{\text{stat}_k(m, b) \mid m \in \mathcal{M}, b \in \mathcal{B}, k \in \{1, \ldots, K\}\} \tag{2.2}$$

(*ii*) **Instance-level Data ($\mathcal{D}_{\text{raw}}$).** This set contains the raw input-output pairs. Let $\mathcal{I}_b$ be the set of instance indices for benchmark $b$. $\mathcal{D}_{\text{raw}}$ aggregates results across all benchmarks:

$$\mathcal{D}_{\text{raw}} = \bigcup_{b \in \mathcal{B}} \{(x_{b,i}, a^*_{b,i}, \{r_{m,b,i}\}_{m \in \mathcal{M}}, b, i) \mid i \in \mathcal{I}_b\} \tag{2.3}$$

where $x_{b,i}$ is the instance input, $a^*_{b,i}$ is the ground truth, and $\{r_{m,b,i}\}$ are the model responses.

By structuring data into aggregated statistics and raw traces, we enable agents to perform both high-level ranking and deep-dive error analysis.

### 2.2 Framework Overview

We propose AGENT4WEAKNESS, a multi-agent framework designed to automate the discovery of model weaknesses based on user queries. As illustrated in Figure 2, the system orchestrates the analysis through three collaborative agents To ensure flexibility and analytical depth while effectively utilizing large-scale evaluation data, we incorporate two critical architectural designs. First, we implement a **Domain-Aware Memory** to inject specialized evaluation expertise into the agents, serving as

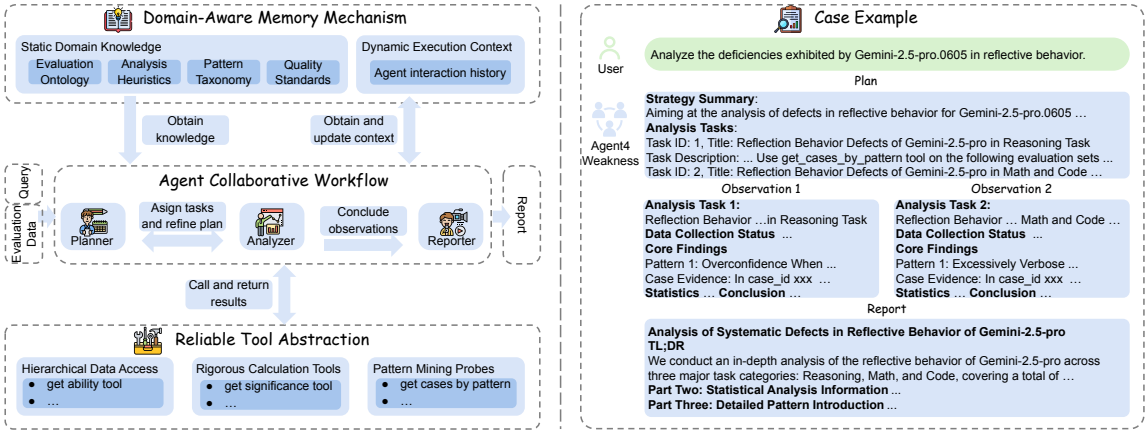


Figure 2: Overview of the AGENT4WEAKNESS framework. The left panel illustrates the system architecture featuring the domain-aware memory mechanism, the reliable tool abstraction, and the collaborative workflow among the Planner, Analyzer, and Reporter agents. The right panel demonstrates an execution example, showcasing the intermediate outputs of each agent.

a cognitive scaffold. Second, we utilize a **Reliable Tool Abstraction** to decouple logical reasoning from raw data retrieval and numerical computation, thereby mitigating hallucinations common in complex data analysis tasks.

### 2.3 Domain-Aware Memory Mechanism

Generic large language models typically lack the specialized cognitive framework required for professional model evaluation (Li et al., 2025; Luettgau et al., 2025). To address this, AGENT4WEAKNESS incorporates a Domain-Aware Memory that functions as a cognitive scaffold, injecting expert-level definitions, logic, and standards into the agent workflow. This memory system is divided into Static Domain Knowledge and Dynamic Execution Context.

**Static Domain Knowledge ($\mathbf{K_{static}}$).** This component serves as a persistent repository of expert methodologies embedded in the system prompt. Unlike simple instructions, $\mathbf{K}_{\text{static}}$ structures the reasoning process through four key dimensions:

*(i) Hierarchical Evaluation Ontology.* The memory defines a structured data schema ranging from atomic test cases to aggregated benchmark collections. It also incorporates specific statistical concepts, such as best-of-N sampling and token consumption metrics, to ensure agents accurately interpret granular performance data and correctly invoke data access tools to obtain relevant data.

*(ii) Professional Analysis Heuristics.* To ensure robust assessment, we inject heuristics that guide agents beyond surface-level analysis. Key strategies include benchmarking against state-of-the-art baselines rather than observing isolated scores, detecting inverse correlations between capabilities, and prioritizing relative rankings over absolute magnitudes for context-sensitive evaluation.

*(iii) Behavioral Pattern Taxonomy.* The memory catalogs recognized failure modes and stylistic tendencies to facilitate targeted diagnosis. Specific examples include distinguishing between context-conflicting hallucinations and external knowledge errors, or identifying rigid formatting issues like emoji overuse and code-switching.

*(iv) Codified Quality Standards.* To ensure high-quality output, $\mathbf{K}_{\text{static}}$ embeds strict criteria for report generation. For instance, every numeric claim must be traceable to specific data sources, such as case identifiers. Furthermore, the report must strictly separate observation from interpretation, avoid unsupported claims, and maintain terminological consistency throughout.

**Dynamic Execution Context ($\mathbf{H_{dyn}}$).** To ensure consistency across the multi-turn workflow, the system maintains a mutable state $\mathbf{H}_{\text{dyn}}$. This cumulative log records the entire interaction history, capturing every input and output generated by all agents from the reception of the initial user query to the current execution state. By persisting this global trajectory, the mechanism guarantees that subsequent reasoning and the final report remain grounded in the verified evidence and logical chain established during previous steps.

### 2.4 Reliable Tool Abstraction

Directly prompting models to ingest and analyze massive raw datasets often leads to context over-

| Tool | Purpose | Category |
|---|---|---|
| `get score tool` | Return a Markdown table listing scores of multiple models across benchmarks. | $\mathcal{T}_{\text{daq}}$ |
| `get significance tool` | Using the specified model as the baseline, compute other models' score differences, percentage changes, improvements, and statistical significance relative to the baseline. | $\mathcal{T}_{\text{stat}}$ |
| `get cases by pattern` | Automatically analyze all cases of the specified benchmark. | $\mathcal{T}_{\text{deep}}$ |

Table 1: Examples of the tools used in AGENT4WEAKNESS include their names, purposes, and categories, with detailed tool information provided in Appendix E.

flow and hallucination (Du et al., 2025; Ji et al., 2025; Zhou et al., 2025). We address this by abstracting data access and analysis into a deterministic tool library $\mathcal{T}$ categorized into three functional groups. This design effectively decouples the high-level reasoning process from low-level data interaction. We present representative tools in Table 1.

**Hierarchical Data Access ($\mathcal{T}_{\text{daq}}$).** Processing the entire evaluation corpus simultaneously is computationally intractable. We address this by abstracting the raw data into multi-granular views. This toolset enables agents to navigate from high-level benchmark metadata down to specific input-output traces on demand. Such structured retrieval effectively circumvents context overflow while ensuring agents can access necessary information efficiently.

**Rigorous Calculation Tools ($\mathcal{T}_{\text{stat}}$).** Direct interpretation of comprehensive evaluation data often induces hallucination due to excessive noise (Liu et al., 2024; Yang et al., 2025). Furthermore, relying on agents to generate analysis code from scratch often results in low efficiency and potential errors (Dong et al., 2025; Xia et al., 2025). We mitigate these issues by encapsulating frequent evaluation routines into a standardized tool library. This component integrates professional statistical packages to facilitate objective inquiries, such as calculating rankings, quantifying gaps to state-of-the-art performance, and computing statistical significance (details in Appendix E). Complementing these pre-built functions, we provide a Python execution interface that enables the agent to implement custom logic when specific analytical needs extend beyond the provided toolset.

**Pattern Mining Probes ($\mathcal{T}_{\text{deep}}$).** Statistical metrics often fail to capture semantic error patterns (Yang et al., 2024; Lee et al., 2025). We consequently employ model-based tools that function as targeted probes to analyze raw responses comparing with other models' responses and the ground truth, and identify specific failure modes. This approach restricts the LLM to instance-level analysis within a controlled tool call. Such constraints focus the attention mechanism and minimize the risk of confabulation inherent in open-ended generation.

### 2.5 Agent Collaborative Workflow

The analysis process is orchestrated through a sequential collaboration among the three agents, governed by the memory mechanism and tools described above. We show the whole execution examples in Appendix D.7.

**Planner: Intent Decomposition.** The Planner $\mathcal{P}$ acts as the strategic controller. It utilizes the analysis schema from $\mathbf{K}_{\text{static}}$ to translate the query $q$ into an executable plan $\pi$. This plan consists of a sequence of logical tasks, where each task assigned to the Analyzer specifies the required tool and the target arguments. Additionally, it evaluates whether the gathered information in $\mathbf{H}_{\text{dyn}}$ is sufficient to support an in-depth report answering the query, thereby determining whether to refine the plan for further investigation or signal termination to activate the Reporter.

**Analyzer: Evidence Execution.** The Analyzer $\mathcal{A}$ acts as the execution engine. It iterates through the plan $\pi$ and invokes the appropriate tools from $\mathcal{T}$. The resulting observations $O$ are stored in the dynamic state, ensuring they are grounded in actual data rather than speculation.

**Reporter: Report Synthesis.** Finally, the Reporter $\mathcal{R}$ synthesizes the observations $O$. It filters the findings based on relevance to the query $q$ and structures the narrative using professional reporting templates in $\mathbf{K}_{\text{static}}$. The Reporter explicitly links high-level conclusions to the supporting evidence generated by the Analyzer, producing the final in-depth diagnostic report $y$.

## 3 Experiments

### 3.1 Settings

**Evaluation Data.** We first get the evaluation data by evaluating 104 models on 27 benchmarks, recording the aggregated statistics and instance-level data. A detailed list of all models and benchmarks with evaluation settings is provided in Appendix B. We conduct the following inquiries (see the query design in Appendix D.1), including: ***Q1***. Analyze the weaknesses in the model's capabilities. ***Q2***. Analyze the disadvantages of the model in terms of time cost and token consumption. ***Q3***. Analyze the weaknesses in the model's behavior. ***Q4***. Does the model instruction non-compliance exist, and under what circumstances is this phenomenon most severe? ***Q5***. Analyze the deficiencies exhibited by the model in reflective behavior. ***Q6***. Analyze the relationship between the model's abilities and its maximum ability limit.

**Models.** We employ Claude-Opus-4.1-thinking (Anthropic, 2025) to run baselines and AGENT4WEAKNESS. We also present the results of GPT-5-high (OpenAI, 2025) and Gemini-2.5-pro (Google, 2025b) in Appendix D.2. We select 8 mainstream LLMs to query their weakness individually, including: GPT-5-high, Grok-4 (xAI, 2025), Claude-Opus-4.1-thinking, Gemini-2.5-pro, Qwen-3-235B-A22B-Thinking-2507 (Qwen Team, 2025), Seed-1.6-Thinking-250715 (ByteDance, 2025), Deepseek-V3.1-0821-Thinking (DeepSeek-AI et al., 2025), and Gemini-2.5-Flash-0520 (Google, 2025a). We report the average scores on the analysis results of querying the 8 models.

**Evaluation.** To evaluate the quality of the analysis generated by AGENT4WEAKNESS, we devise a scoring rubric. Specifically, we define *Requirement Fulfillment* and *Content Value* to assess the flexibility and depth, respectively. Furthermore, we incorporate *Factuality* and *Readability* to ensure a comprehensive evaluation. Detailed scoring rule is shown in Appendix C. Utilizing the rubrics, we conduct both human and model-based evaluations. Specially, we employ professional evaluators to score the generated analyses across four dimensions, each on a scale from 0 to 10. We additionally employ an agent system sharing the same framework of AGENT4WEAKNESS, powered by Claude-Opus-4.1-thinking. We report the average score across 5 trials and discuss the robustness of AGENT4WEAKNESS in Appendix D.4.

| Method | Query | RF | CV | F | R |
|---|---|---|---|---|---|
| CoT | Q1 | 6.7 | 5.7 | 6.4 | 7.9 |
| | Q2 | 3.7 | 3.0 | 3.3 | 6.4 |
| | Q3 | 6.9 | 6.6 | 7.3 | 8.3 |
| | Q4 | 5.0 | 3.0 | 4.3 | 7.3 |
| | Q5 | 4.3 | 2.8 | 3.0 | 5.3 |
| | Q6 | 7.5 | 6.0 | 4.5 | 6.5 |
| | Avg | 5.7 | 4.5 | 4.8 | 6.7 |
| One-Agent | Q1 | 6.7 | 5.7 | 6.0 | 7.3 |
| | Q2 | 6.7 | 5.5 | 5.0 | 6.4 |
| | Q3 | 7.0 | 4.5 | 6.4 | 5.0 |
| | Q4 | 4.7 | 5.3 | 5.3 | 6.4 |
| | Q5 | 7.0 | 5.7 | 5.0 | 6.7 |
| | Q6 | 6.7 | 7.0 | 6.5 | 8.0 |
| | Avg | 6.5 | 5.5 | 5.7 | 6.6 |
| AGENT4WEAKNESS | Q1 | 8.8 | 8.3 | 9.7 | 7.5 |
| | Q2 | 9.9 | 8.6 | 8.0 | 8.4 |
| | Q3 | 8.9 | 8.7 | 8.1 | 8.7 |
| | Q4 | 8.7 | 7.9 | 8.4 | 7.7 |
| | Q5 | 8.2 | 8.1 | 8.2 | 8.2 |
| | Q6 | 8.5 | 8.5 | 9.2 | 8.3 |
| | Avg | **8.8** | **8.4** | **8.6** | **8.1** |

Table 2: Comparison of AGENT4WEAKNESS with baselines on Requirement Fulfillment (RF), Content Value (CV), Factuality (F), and Readability (R), with a maximum score of 10. Avg denotes the average scores across the 6 queries. Highest average scores are **bold**.

**Baselines.** We compare AGENT4WEAKNESS with: (*i*) CoT involves feeding relevant data and the query into the model and prompting it to think step by step (Wei et al., 2022). (*ii*) One-Agent uses the same tools as AGENT4WEAKNESS, but without specialized roles. We exclude comparisons with prior works due to fundamental paradigm mismatches. The static pipeline methods are limited to isolated model analysis, rendering them incapable of addressing our dynamic, open-ended queries or processing population-level statistics across 104 models. Given these disparities in flexibility and scope, a direct comparison is technically infeasible, so we use the baselines following previous agent works (Chen et al., 2024; Zou et al., 2025). We compare the efficiency and scores with human annotators in Appendix D.3.

### 3.2 Main Results

We report evaluation scores rated by LLMs in Table 2. AGENT4WEAKNESS consistently outperforms baselines with 2.7, 3.4, 3.4, and 1.5 on *RF*, *CV*, *F*, and *R*, respectively. These results demonstrate that AGENT4WEAKNESS not only highlights in-depth analysis and flexibly satisfying user needs, but also produces weakness analyses with high factual accuracy and readability.

**Finding 1. Performance gains are particularly pronounced on complex queries.** The improvements of AGENT4WEAKNESS are larger on Q2-Q5 than on Q1 and Q6. Q1 and Q6 involve straightforward data retrieval and calculation, whereas Q2 requires complex multi-perspective data analysis. Furthermore, Q3 through Q5 necessitate the retrieval, filtering, and in-depth analysis of extensive instances. The baselines struggle with these complex cases, whereas AGENT4WEAKNESS demonstrates robust performance.

**Finding 2. The gain in *Readability* is modest but consistent.** Although AGENT4WEAKNESS outperforms the baselines, improvements are smaller than in other dimensions. This is because models inherently exhibit their own stylistic tendencies, such as habitual word choices and preferred rhetorical structures. Even with explicit guidance on report style, it is challenging to achieve significant gains in readability (Wang et al., 2025).

**Finding 3. Model-based scores align strongly with human evaluation.** To validate the reliability of our metric, we compared model ratings against human annotations (see Appendix C). We observed a robust alignment between the two, evidenced by a strong Pearson correlation ($r = 0.801$, $p \approx 0.03$) and a high Spearman rank correlation ($\rho = 0.944$, $p < 0.01$). These results confirm that our model-based scores reliably reflect human judgment in both magnitude and rank order.

## 4 Discussion

### 4.1 RQ1. Can our method accurately identify model weaknesses?

To further demonstrate that AGENT4WEAKNESS accurately identifies model weaknesses, we utilize it to detect quantifiable capability and behavioral deficiencies. We maintain the main experiment settings, altering only the input queries, and report the averaged scores across 8 target models.

**AGENT4WEAKNESS can accurately identify deficiencies in model capabilities.** We first evaluate the ability to extract statistically-grounded insights from aggregate data. Using statistics from 105 models across 27 benchmarks, we task AGENT4WEAKNESS and baselines with identifying the lowest-ranking models and specific underperforming capability dimensions (e.g., Reasoning, Math, Instruction Following). To ensure fairness, One-Agent and AGENT4WEAKNESS do not include tools that directly return these answers. As shown in Figure 3, AGENT4WEAKNESS achieves consistent accuracy gains over baselines. While CoT struggles to retrieve correct information from large-scale contexts (Liu et al., 2024; Wu et al., 2025) and One-Agent is prone to tool misuse, the memory mechanism and the collaboratice workflow allows AGENT4WEAKNESS to navigate complex evaluation logs without the hallucinations that typically plague long-context retrieval, thereby providing reliable high-level diagnostics.

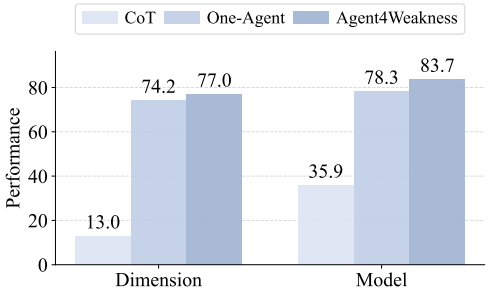

Figure 3: Accuracy in identifying weakness capability dimensions and underperforming models, compared with two baselines.

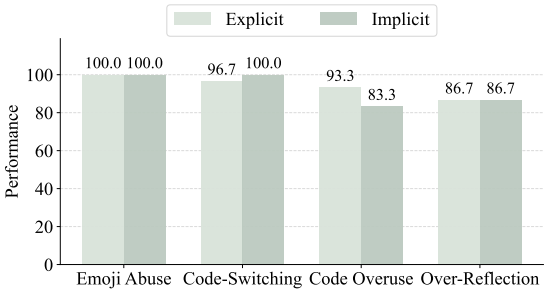

Figure 4: The $F_1$ score of AGENT4WEAKNESS in detecting weaknesses across four behavioral patterns.

**AGENT4WEAKNESS can uncover explicit and implicit behavioral issues at the instance level.** To verify the detection of stylistic biases and logic regressions, we evaluate AGENT4WEAKNESS on identifying specific problematic patterns. We calculate $F_1$ scores by comparing retrieved case IDs against ground-truth labels derived from rule-based verification. Experiments are conducted in two settings: *Explicit* (patterns defined in the prompt) and *Implicit* (definitions omitted). As illustrated in Figure 4, AGENT4WEAKNESS successfully identifies not only the predefined behavioral issues but also novel patterns absent from the initial prompts. Remarkably, the implicit setting even outperforms the explicit one in detecting code-switching. This validates the Domain-Aware Memory and expert-level reasoning claims: the agents utilize comparative analysis to discover novel defects autonomously, rather than relying solely on rigid instructions. Although AGENT4WEAKNESS demonstrates slightly

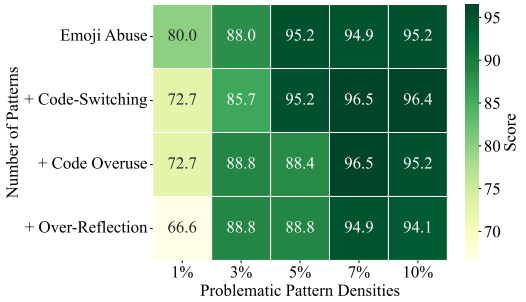


Figure 5: The $F_1$ score of AGENT4WEAKNESS in detecting weaknesses across varying instance densities.

lower accuracy in detecting Code Overuse and Over-Reflection patterns due to the nuance between thoroughness and repetition, the overall performance confirms AGENT4WEAKNESS's utility in mining actionable behavioral insights.

**AGENT4WEAKNESS can accurately detect behavioral issues across varying instance densities.** A reliable automated auditor must function effectively regardless of how prevalent a flaw is. To verify this, we utilize $1,500$ responses generated by GPT-5-high from LiveBench (2025-04-25) (White et al., 2025). After correcting existing flaws, we construct datasets with varying defect densities by substituting clean samples with GPT-5-high synthesized errors. Specifically, for a total density $\rho$, we uniformly inject $\rho/K$ samples across $K$ patterns while maintaining a constant dataset size. The resulting performance fluctuations are illustrated in Figure 5. Empirical results indicate that AGENT4WEAKNESS maintains a robust $F_1$ score ($\approx 90$) even in low-density scenarios, effectively isolating rare failure patterns from massive reasoning traces. This stability demonstrates that AGENT4WEAKNESS avoids the pitfalls of blind trial-and-error by reliably flagging issues whether they are systemic or sporadic. As the density increases, the underlying pattern issues become more obvious, leading to an improvement in the $F_1$ score. While performance dips slightly at extremely high densities due to context overflow during ID retrieval, AGENT4WEAKNESS remains highly effective for practical diagnostic ranges.

### 4.2 RQ2. Are the components in AGENT4WEAKNESS effective?

To validate the architectural necessity of AGENT4WEAKNESS, we conduct ablation studies on representative queries (Q1-Q3) to isolate the contributions of specific components (Table 3).

**Impact of Domain-Aware Memory.** Our results confirm that memory is critical for emulating expert-level auditing. removing *static domain knowledge* significantly degrades *Content Value* and *Readability*, as agents struggle to interpret evaluation nuances without prior knowledge, resulting in unstructured and hyperbolic outputs. Similarly, ablating the *dynamic execution context* severs the link between user intent and execution; without the Planner's initial analysis, the Reporter fails to prioritize observations, leading to indiscriminate and unfocused summarization.

**Necessity of Tool Abstraction.** The removal of *rigorous calculation* and *pattern mining* tools leads to a sharp decline in *Factuality* and *Content Value*. This confirms that high-level diagnostic reasoning must be grounded in low-level data processing; without tools to verify data fidelity and mine reasoning patterns, the framework reverts to superficial analysis lacking statistical evidence. In the experiments of ablating the pattern mining tools, performance on Q1 and Q2 remains consistent with AGENT4WEAKNESS, as these specific tools are not utilized in the responses for Q1 and Q2.

**Role of Multi-Agent Collaboration.** Agent-level ablations highlight distinct roles: the *Planner* is essential for *Content Value*, as it ensures comprehensive diagnostic coverage across multiple perspectives. Meanwhile, the *Reporter* proves indispensable for *Readability*; far from a simple copy-paste mechanism, it is responsible for synthesizing fragmented observations into coherent, actionable intelligence. Since the *Analyzer* is essential for acquiring data, it cannot be fully ablated. Therefore, the ablations of the tool abstraction can be regarded as a partial ablation of the Analyzer.

### 4.3 RQ3. Can the diagnostic insights from AGENT4WEAKNESS enable closed-loop model refinement?

To validate the practical utility of our method, we move beyond passive evaluation to a closed-loop refinement setting. Specifically, we investigate whether the actionable intelligence mining by AGENT4WEAKNESS can guide model self-evolution. We query AGENT4WEAKNESS to diagnose weaknesses of target models on AIME2025 (AIME, 2025), Aider (Gauthier, 2025), and LiveBench (2025-04-25) (White et al., 2025), generating targeted refinement prompts. These diagnostic insights are then fed back into the models to measure performance gains across 10 runs and Best-of-$N$ metrics. As shown in Figure 6, applying the diagnostic guidance from AGENT4WEAKNESS

| Method | Q1 | | | | Q2 | | | | Q3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | RF | CV | F | R | RF | CV | F | R | RF | CV | F | R |
| AGENT4WEAKNESS | **8.8** | **8.3** | **9.7** | **7.5** | **9.9** | **8.6** | **8.0** | **8.4** | **8.9** | **8.7** | **8.1** | **8.7** |
| *w/o* Static domain knowledge | 8.3 | 7.4 | 8.6 | 6.7 | 7.9 | 8.0 | 7.7 | 6.1 | 4.6 | 1.6 | 1.6 | 6.1 |
| *w/o* Dynamic execution context | 7.2 | 7.6 | 8.3 | 7.2 | 6.9 | 8.2 | 7.6 | 7.3 | 4.2 | 4.6 | 3.1 | 6.4 |
| *w/o* Rigorous calculation tools | 7.9 | 7.0 | 8.3 | 7.4 | 9.3 | 8.4 | 7.9 | 7.7 | 5.3 | 4.3 | 2.0 | 6.0 |
| *w/o* Pattern mining probes | 8.8 | 8.3 | 9.7 | 7.5 | 9.9 | 8.6 | 8.0 | 8.4 | 6.0 | 4.5 | 4.2 | 7.5 |
| *w/o* Planner | 6.5 | 6.1 | 6.5 | 7.4 | 8.0 | 6.4 | 7.3 | 6.0 | 5.6 | 2.0 | 6.0 | 6.2 |
| *w/o* Reporter | 7.2 | 6.5 | 6.8 | 6.3 | 6.7 | 5.0 | 6.6 | 5.8 | 4.8 | 3.0 | 5.0 | 5.7 |

Table 3: Ablation study across three queries (Q1–Q3), with a maximum score of 10. Metrics are Requirement Fulfillment (RF), Content Value (CV), Factuality (F), and Readability (R).

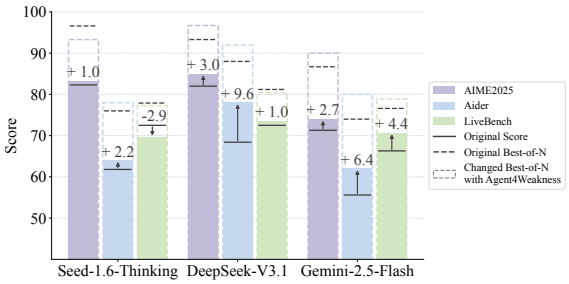


Figure 6: Performance comparison before and after applying the diagnostic guidance from AGENT4WEAKNESS.

yields a consistent performance boost, increasing the average score by 3.7 points (from 70.3 to 74.0). Crucially, the performance gains persist in the held-out setting (detailed in Appendix D.5), where diagnostic insights derived from the evaluation set are applied to unseen test instances. This confirms that AGENT4WEAKNESS transforms evaluation logs into actionable drivers for improvement.

We highlight three key findings from our analysis: (*i*) **AGENT4WEAKNESS uncovers implicit behavioral pathologies for targeted optimization.** Unlike black-box metrics, our agents identify the root causes of failure. For instance, on AIME2025, AGENT4WEAKNESS detects that DeepSeek-V3.1 applies congruence properties superficially. By generating guidance that explicitly connects modular arithmetic with Euler's theorem, AGENT4WEAKNESS rectifies the model's underlying reasoning logic, proving that our method facilitates surgical, logic-aware optimization rather than blind trial-and-error. (*ii*) **Diagnostic guidance stabilizes reasoning and elevates capability ceilings.** The optimized prompts consistently enhance or maintain Best-of-$N$ performance across the majority of models. This indicates that the improvements are not due to stochastic chance but stem from a systematic reduction in error frequency. By mitigating persistent stylistic biases and logic regressions, AGENT4WEAKNESS effectively guides models to circumvent reasoning pitfalls, thereby unlocking latent capabilities that were previously obscured by poor behavioral patterns. (*iii*) **Correction specificity varies by task nature and instruction following capability.** The most substantial gains are observed on code tasks (Aider), where AGENT4WEAKNESS corrects specific procedural errors. For example, it identifies that Gemini-2.5-Flash suffers from a preemptive debugging bias, which is hallucinating hypothetical bugs before analyzing the actual error. The generated guidance constrains the model to focus solely on the immediate traceback, significantly improving fix rates. However, gains on Seed-1.6-Thinking are marginal with a slight Best-of-$N$ decline; our investigation reveals this is due to the model's weaker instruction-following capability, which prevents it from strictly adhering to the complex diagnostic provided by AGENT4WEAKNESS.

## 5 Conclusion

In this work we present AGENT4WEAKNESS which serves as a multi-agent framework designed to automate the discovery of granular model limitations and transcend the constraints of superficial aggregate metrics. By synergizing domain-aware memory mechanisms with reliable tool abstractions our system effectively replicates the rigorous diagnostic workflow of human experts to generate actionable intelligence from raw evaluation traces. Empirical results demonstrate that AGENT4WEAKNESS yields diagnostic reports with significantly superior depth and flexibility compared to baselines. Furthermore we validate that the specific weaknesses identified by our framework can be leveraged as targeted feedback to drive substantial performance improvements in downstream tasks, which establishes our method as a pivotal advancement for iterative model optimization and the development of more reliable reasoning systems.

## Limitations

The report generated by AGENT4WEAKNESS mainly focus on English, as we only design English prompts currently. In the future work, we will update experiments on more languages.

## Ethics Statement

Every dataset and model used in the paper is accessible to the public, and our application of them adheres to their respective licenses and conditions. We employ AI tools for the writing polishing.

## References


AIME. 2025. AIME problems and solutions.

Anthropic. 2025. Claude-Opus-4.1: A large language model. https://www.anthropic.com/news/claude-opus-4-1. Accessed: June 2025.

Jake Brawer, Kayleigh Bishop, Bradley Hayes, and Alessandro Roncone. 2023. Towards a natural language interface for flexible multi-agent task assignment. *Preprint*, arXiv:2311.00153.

ByteDance. 2025. doubao-seed-1.6: A large language model. https://www.volcengine.com/docs/82379/1536428. Accessed: August 2025.

Tyler A. Chang and Benjamin K. Bergen. 2024. Language model behavior: A comprehensive survey. *Computational Linguistics*, 50(1):293–350.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. DeepSeek-V3 Technical Report. Realsed: March 2025; Accessed: August 2025.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A survey on code generation with llm-based agents. *Preprint*, arXiv:2508.00083.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Babu Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. 2025. Context length alone hurts LLM performance despite perfect retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 23281–23298, Suzhou, China. Association for Computational Linguistics.

Paul Gauthier. 2025. Aider llm leaderboards. https://aider.chat/docs/leaderboards/.

Google. 2025a. Gemini-2.5-Flash: A large language model. https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash. Accessed: August 2025.

Google. 2025b. Gemini-2.5-Pro(preview 05-06): A large language model. https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-pro. Accessed: August 2025.

Taojun Hu and Xiao-Hua Zhou. 2024. Unveiling llm evaluation focused on metrics: Challenges and solutions. *Preprint*, arXiv:2404.09135.

Lanlan Ji, Dominic Seyler, Gunkirat Kaur, Manjunath Hegde, Koustuv Dasgupta, and Bing Xiang. 2025. PHANTOM: A benchmark for hallucination detection in financial long-context QA. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Seongyun Lee, Seungone Kim, Minju Seo, Yongrae Jo, Dongyoung Go, Hyeonbin Hwang, Jinho Park, Xiang Yue, Sean Welleck, Graham Neubig, Moontae Lee, and Minjoon Seo. 2025. The cot encyclopedia: Analyzing, predicting, and controlling how a reasoning model will think. *Preprint*, arXiv:2505.10185.

Zongxia Li, Xiyang Wu, Ishani Mondal, Alexa Siu, Jordan Lee Boyd-Graber, and Ani Nenkova. 2025. LLM-as-a-judge failures at automating the identification of poor quality outputs in free-form texts. In *Proceedings of The 5th New Frontiers in Summarization Workshop*, pages 1–16, Hybrid. Association for Computational Linguistics.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Lennart Luettgau, Harry Coppock, Magda Dubois, Christopher Summerfield, and Cozmin Ududec. 2025. Hibayes: A hierarchical bayesian modeling framework for ai evaluation statistics. *Preprint*, arXiv:2505.05602.

Mazda Moayeri, Vidhisha Balachandran, Varun Chandrasekaran, Safoora Yousefi, Thomas FEL, Soheil Feizi, Besmira Nushi, Neel Joshi, and Vibhav Vineet. 2025. Unearthing skill-level insights for understanding trade-offs of foundation models. In *The Thirteenth International Conference on Learning Representations*.

Vishvak Murahari, Ameet Deshpande, Peter Clark, Tanmay Rajpurohit, Ashish Sabharwal, Karthik Narasimhan, and Ashwin Kalyan. 2024. QualEval: Qualitative evaluation for model improvement. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2093–2111, Mexico City, Mexico. Association for Computational Linguistics.

Shiwen Ni, Guhong Chen, Shuaimin Li, Xuanang Chen, Siyi Li, Bingli Wang, Qiyao Wang, Xingjian Wang, Yifan Zhang, Liyang Fan, Chengming Li, Ruifeng Xu, Le Sun, and Min Yang. 2025. A survey on large language model benchmarks. *Preprint*, arXiv:2508.15361.

OpenAI. 2025. Introducing gpt-5. `https://openai.com/index/introducing-gpt-5/`.

Qwen Team. 2025. Qwen3-235B: A large language model. Accessed: August 2025.

Changhao Song, Yazhou Zhang, Hui Gao, Ben Yao, and Peng Zhang. 2025. Large language models for subjective language understanding: A survey. *Preprint*, arXiv:2508.07959.

Yufei Tian, Jiao Sun, Nanyun Peng, and Zizhao Zhang. 2025. Skillverse : Assessing and enhancing llms with tree evaluation. *Preprint*, arXiv:2506.00319.

Zhengxiang Wang, Nafis Irtiza Tripto, Solha Park, Zhenzhen Li, and Jiawei Zhou. 2025. Catch me if you can? not yet: Llms still struggle to imitate the implicit writing styles of everyday authors. *Preprint*, arXiv:2509.14543.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025. Livebench: A challenging, contamination-limited LLM benchmark. In *The Thirteenth International Conference on Learning Representations*.

Xiaodong Wu, Minhao Wang, Yichen Liu, Xiaoming Shi, He Yan, Lu Xiangju, Junmin Zhu, and Wei Zhang. 2025. LIFBench: Evaluating the instruction following performance and stability of large language models in long-context scenarios. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16445–16468, Vienna, Austria. Association for Computational Linguistics.

xAI. 2025. Grok4: A large language model. `https://docs.x.ai/docs/models#models-and-pricing`. Accessed: August 2025.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. 2025. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *Preprint*, arXiv:2511.16043.

Blair Yang, Fuyang Cui, Keiran Paster, Jimmy Ba, Pashootan Vaezipoor, Silviu Pitis, and Michael R. Zhang. 2024. Report cards: Qualitative evaluation of language models using natural language summaries. *Preprint*, arXiv:2409.00844.

Joonho Yang, Seunghyun Yoon, Hwan Chang, Byeongjeong Kim, and Hwanhee Lee. 2025. Hallucinate at the last in long response generation: A case study on long document summarization. *Preprint*, arXiv:2505.15291.

Zhiyuan Zeng, Yizhong Wang, Hannaneh Hajishirzi, and Pang Wei Koh. 2025. Evaltree: Profiling language model weaknesses via hierarchical capability trees. *Preprint*, arXiv:2503.08893.

Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. 2025. GSM-$\infty$: How do your LLMs behave over infinitely increasing reasoning complexity and context length? In *Forty-second International Conference on Machine Learning*.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. 2025. Latent collaboration in multi-agent systems. *Preprint*, arXiv:2511.20639.

## A Related Work

Existing research moves beyond aggregate benchmark scores to pinpoint granular deficiencies, primarily focusing on two dimensions: capability and behavior.

**Capability Weakness Discovery.** These methods map model failures to specific skill taxonomies. For instance, QualEval (Murahari et al., 2024) summarizes error patterns to categorize benchmark questions, identifying domains where models underperform. Similarly, EvalTree (Zeng et al., 2025) and SkillVerse (Tian et al., 2025) construct hierarchical capability trees, locating weakness nodes through performance traversing.

**Behavioral Weakness Discovery.** Research in this category diagnoses reasoning processes rather than just outcomes (Chang and Bergen, 2024). ReportCards (Yang et al., 2024) iteratively analyzes reasoning traces of the single model on a benchmark to detect recurring pitfalls, and CoT Encyclopedia (Lee et al., 2025) clusters error patterns in Chain-of-Thought paths to classify behavioral deficiencies.

**Limitations of Prior Work.** Despite their effectiveness in specific scenarios, these approaches suffer from inherent limitations: (1) **Fixed Pipelines:** They typically employ rigid analysis workflows yielding pre-defined reports, lacking the flexibility to address diverse, open-ended user queries (Brawer et al., 2023). (2) **Isolated Analysis:** Most methods evaluate models in isolation, ignoring comparative statistics across model populations (Luettgau et al., 2025). (3) **Data Narrowness:** They generally focus solely on performance accuracy of one model on the single benchmark, neglecting scores of other mainstream models for comparison and critical non-performance metrics such as computational cost or token efficiency (Hu and Zhou, 2024). In contrast, AGENT4WEAKNESS is designed as a dynamic multi-agent system integrating statistical tooling and domain-aware memory to achieve flexible, comprehensive, and in-depth analysis.

## B Evaluation Data

We detail the models and benchmarks utilized in evaluation data. All model evaluations are conducted using greedy decoding (temperature=0) and self-consistency (temperature=1) respectively, unless strictly specified otherwise by the benchmark's standard protocol. We will release the data upon acceptance.

### B.1 Models

We evaluate the following models: qwen-3-next-80b-a3b-thinking, qwen-3-next-80b-a3b-instruct, qwen3-max-preview, GPT-5-high, qwen-3-4b (think), GPT-OSS-20b-medium, Hunyuan-T1-0711, qwen-3-coder-plus, qwen-3-235b-a22b-instruct-2507 (nothink), qwen-3-235b-a22b-thinking-2507, Grok-4, Grok-3, Llama-4-Maverick, Llama-4-Scout, Grok-3-mini-high, ChatGPT-4o-latest, Doubao-1.5-Lite-32k.250115, Doubao-1.5-Pro-32k.250115, Doubao-1.5-Thinking-Pro-M.250415, Doubao-1.5-Thinking-Pro.250415, Seed-1.6-Flash.250615, Seed-1.6-Thinking.250615, Seed-1.6-AutoCoT.250615-AutoCoT, Seed-1.6-AutoCoT.250615-NoCoT, DeepSeek-R1-0528, o4-mini-high, qwen-plus-0428 (nothink), GPT-4.1-nano, DeepSeek-V3-0324, Gemini-2.0-Flash, Claude-4-Sonnet-nothinking, GPT-4o-1120, o3-mini-high, Minimax-Text-01, GPT-4.1-mini, Baichuan4-Turbo, o1-high, Gemini-2.0-Flash-Lite, qwen-max-0125 (nothink), GLM-4-Air.0414, Mistral-large-2411, Nova-pro, Yi-lightning, Claude-3.7-Sonnet, GPT-4.1, Gemini-2.5-flash.0520, qwen-3-235b-a22b-2504 (think), qwen-turbo-0428 (nothink), SenseNova-V6-Turbo, SenseNova-V6-Pro, Gemini-2.5-pro.0605, ERNIE-4.5-Turbo-32K, GLM-Z1-Air.0414, Claude-3.7-Sonnet-thinking, Claude-4-Sonnet-thinking, qwen-3-30b-a3b (think), qwen-3-32b (think), ERNIE-X1-Turbo-32K, 360-gpt2-o1, SenseNova-V6-Reasoner, Claude-4-Opus-nothinking, StepFun-2-16k, Claude-4-Opus-thinking, StepFun-R1-V-mini, Kimi-Thinking-preview, Gemini-2.5-Pro, Gemini-2.5-Flash, dots-llm1, Gemini-2.0-flash-lite-preview.0617, Hunyuan-T1-0529, GPT-4o-mini, ERNIE-4.5-Turbo-128K-Preview.0629, ERNIE-4.5-300b-a47b, o3-high, qwen-plus-0714 (nothink), qwen-turbo-0715 (nothink), Kimi-K2, qwen-3-coder-480b-a35b-instruct, Gemini-2.5-Flash-Lite, qwen-3-30b-a3b-instruct-2507 (nothink), qwen-3-30b-a3b-thinking-2507, Seed-1.6-Thinking-agent-preview, GLM-4.5, GLM-4.5-AirX, GLM-4.5-Air, GPT-OSS-20b-high, GPT-5-mini-high, GPT-5-nano-high, GPT-5-chat, GLM-4.5-X, GPT-5-medium, Claude-Opus-4.1-nothinking, 360-zhinao2-o1.5, StepFun-3, Claude-Opus-4.1-thinking, Deepseek-V3.1-0821-nothinking, Deepseek-V3.1-0821-thinking, Seed-1.6-AutoCoT.250615-CoT, Seed-1.6-Flash.250715,

Seed-1.6-Thinking.250715, Kimi-K2-0905, GPT-OSS-120B-low, GPT-OSS-120B-medium, and GPT-OSS-120B-high.

### B.2 Benchmarks

We consider the following benchmarks: MMLU pro, MMLU, Humanity Last Exam, GPQA diamond, SuperGPQA, LiveBench, MixEvalHard, ArenaHard, ARCAGI, ProcBench, KORBench, ZebraLogicBench, AIME 2025, AIME 2024, HARP, Omni MATH, OlympiadBench, SWE Bench Verified, SWE Lancer, Aider, LiveCodeBench, MultiChallenge, IFEval, Collie Hard, Chinese SimpleQA, SimpleQA, and MMMLU.

## C Evaluation for AGENT4WEAKNESS

The scoring rubrics for human evaluators and model evaluators are the same, which are illustrated in Table 4. For human evaluators, we recruit 5 senior experts in LLM evaluation to score the weakness reports generated by the 8 target models independently and calculate the average ratings. The calculated Cronbach's $\alpha$ coefficient of 0.946 demonstrates exceptional inter-rater reliability among the human evaluators. We present the average human and model scores in Table 5. The results indicate a high consistency between human and model evaluations, with fluctuations not exceeding 0.3.

## D Supplementary Experiments and Discussions

### D.1 Query Design

Queries Q1-Q3 represent three primary categories of user requirements for model evaluation: Q1 focuses on performance weaknesses, Q2 on resource consumption, and Q3 on behavioral deficiencies. For these general-purpose queries, AGENT4WEAKNESS is designed to generate comprehensive, multi-faceted reports. We elaborate on the specific aspects covered by each query type below:

- **Q1: Identifying Performance Weaknesses.** This requires AGENT4WEAKNESS to compare the performance of the target model against other models (including SOTA and series-specific counterparts) across multiple benchmarks. The analysis includes performance gaps, rankings, and tiering. It also assesses performance consistency across different benchmarks to identify potential "seesaw" effects (i.e., fluctuating capabilities).

- **Q2: Analyzing Resource Inefficiencies.** This requires AGENT4WEAKNESS to conduct a comprehensive analysis of the target model's performance, inference time, and token consumption relative to other models. This analysis considers the impact of benchmark difficulty on efficiency and examines the correlation between token usage and performance gains.

- **Q3: Detecting Behavioral Deficiencies.** This requires AGENT4WEAKNESS to identify undesirable behavioral patterns in the target model's responses that differ from those of other models and contribute to poor performance. Examples include evaluating its instruction-following capability or assessing the frequency and success rate of self-reflection within its outputs.

To validate the real-world relevance of these query types, we surveyed 51 LLM practitioners. Their needs for identifying model weaknesses aligned with these three categories, which accounted for 41.1%, 7.1%, and 51.3% (totaling 99.5%) of their reported queries, respectively. The distribution confirms that our query categories effectively address dominant user concerns. To further demonstrate the flexibility and generalization of AGENT4WEAKNESS, we also introduce three additional, fine-grained queries: Q4, Q5, and Q6.

### D.2 Running AGENT4WEAKNESS with Other Models

In addition to Claude-Opus-4.1-thinking, which was used in our main experiments, we also evaluate GPT-5 (OpenAI, 2025) and Gemini-2.5-pro (Google, 2025b). A comparison of their performance is presented Table 6. For consistency, we continue to use Claude-Opus-4.1-thinking (Anthropic, 2025) as the evaluator. We observe that AGENT4WEAKNESS still consistently and significantly outperforms the baseline across all four dimensions.

### D.3 Efficiency of AGENT4WEAKNESS

We compare the time and token consumption of AGENT4WEAKNESS, the baselines, and professional human evaluators on Q3. The Q3 task is

| Criteria | Primary Definition | Deduction Logic |
|---|---|---|
| **Requirement Fulfillment** | Evaluates the agent adherence to both general and specific instructions within query. | • Non-adherence: $-1$ points |
| **Content Value** | Assesses the utility of the output, including structural integrity and the soundness of the analysis. | • Incomplete structure: $-1$ to $-3$ points<br>• Missing a primary category: $-3$ points<br>• Missing a secondary category: $-1$ to $-2$ points<br>• Incomplete case presentation table: $-2$ points<br>• Unsound analysis: $-1$ to $-2$ points<br>• Inappropriate primary category: $-2$ points<br>• Inappropriate secondary category: $-1$ point |
| **Factuality** | Verifies the accuracy and reliability of data citations and external links. | • A single instance of a factual error: $-2$ points |
| **Readability** | Measures the clarity, fluency of the language, and the effectiveness of case presentation. | • Expressive or logical flaws: $-0.5$ points per instance<br>• Poor reading experience: $-1$ point |

Table 4: Definitions of four dimensions and the deduction rules on a 10 point scale.

| Queries | Evaluators | **Requirement Fulfillment** | **Content Value** | **Factuality** | **Readability** |
|---|---|---|---|---|---|
| Q1 | Human | 8.8 | 8.0 | 9.5 | 7.5 |
| | Model | 8.8 | 8.3 | 9.7 | 7.5 |
| Q2 | Human | 10.0 | 8.6 | 8.0 | 8.5 |
| | Model | 9.9 | 8.6 | 8.0 | 8.4 |
| Q3 | Human | 8.8 | 8.8 | 8.1 | 8.6 |
| | Model | 8.9 | 8.7 | 8.1 | 8.7 |

Table 5: Comparison of model scores and human scores on Q1-Q3.

selected because it demands extensive observation of model responses across multiple evaluation sets, making it **the most resource-intensive** of all queries. For the human-annotated results, we commission professional LLM evaluators to write reports for the Q3 task across eight models; the annotation time is an approximate statistic, and these reports are subsequently scored by Claude-Opus-4.1-thinking.

While the baseline methods consumes fewer resources, its unacceptably low scores across the four dimensions of Requirement Fulfillment, Content Value, Factuality, and Readability render it inadequate. Conversely, manual annotation by experts is excessively time-consuming and lacks scalability. Furthermore, human analysis is inherently constrained by individual perspectives, often causing evaluators to overlook or miss specific model deficiencies during large-scale instance analysis. This limitation frequently results in the Content Value dimension scoring lower than the others. Therefore, we argue that the resource consumption of AGENT4WEAKNESS is necessary and justified. It represents a step toward the automated, flexible, and high-quality discovery of model weaknesses.

### D.4 Robustness of AGENT4WEAKNESS

To assess the robustness of AGENT4WEAKNESS, we conduct 5 independent runs for each query, analyzing the deficiencies of GPT-5-high. To mitigate potential biases from LLM-based scoring, we employ human evaluation.

The results are presented in Table 8. We further quantify the stability of AGENT4WEAKNESS by calculating the **average (Avg)** and **standard deviation (Std)** across the 5 runs for each metric. The results demonstrate exceptional consistency: the standard deviations are extremely low across all queries and dimensions. Notably, the highest observed standard deviation is merely 0.89 (for Q3-Factuality), with most metrics exhibiting an SD of $\approx 0.5$ or less (e.g., Q1-Readability and Q2-Content Value show an SD of 0.00). This low variance quantitatively confirms that AGENT4WEAKNESS produces stable and reliable results, minimizing random fluctuations across independent runs.

| Model | Method | Query | Requirement Fulfillment | Content Value | Factuality | Readability |
|---|---|---|---|---|---|---|
| GPT-5 | CoT | Q1 | 4.7 | 4.7 | 3.6 | 6.4 |
| | | Q2 | 6.8 | 5.0 | 7.2 | 8.0 |
| | | Q3 | 4.5 | 5.0 | 5.3 | 6.6 |
| | | Avg | 5.3 | 4.9 | 5.4 | 7.0 |
| | One-Agent | Q1 | 6.4 | 5.5 | 6.9 | 8.0 |
| | | Q2 | 6.5 | 5.3 | 7.8 | 8.2 |
| | | Q3 | 6.5 | 5.0 | 7.2 | 7.8 |
| | | Avg | 6.5 | 5.3 | 7.3 | 8.0 |
| | AGENT4WEAKNESS | Q1 | 8.7 | 9.1 | 9.3 | 8.4 |
| | | Q2 | 7.9 | 7.5 | 9.1 | 8.9 |
| | | Q3 | 7.5 | 7.1 | 8.5 | 8.1 |
| | | Avg | **8.0** | **7.9** | **9.0** | **8.5** |
| Gemini-2.5-pro | CoT | Q1 | 7.3 | 6.7 | 4.7 | 7.8 |
| | | Q2 | 7.6 | 6.5 | 7.3 | 7.6 |
| | | Q3 | 4.8 | 4.6 | 4.3 | 6.7 |
| | | Avg | 6.6 | 5.9 | 5.4 | 7.4 |
| | One-Agent | Q1 | 6.7 | 4.9 | 5.2 | 7.3 |
| | | Q2 | 7.8 | 4.7 | 7.3 | 8.3 |
| | | Q3 | 5.0 | 4.4 | 4.7 | 8.0 |
| | | Avg | 6.5 | 4.7 | 5.7 | 7.9 |
| | AGENT4WEAKNESS | Q1 | 8.7 | 7.2 | 8.0 | 8.3 |
| | | Q2 | 7.5 | 7.0 | 7.7 | 8.5 |
| | | Q3 | 8.5 | 7.2 | 7.5 | 8.6 |
| | | Avg | **8.2** | **7.1** | **7.7** | **8.5** |

Table 6: Scores of the baselines and AGENT4WEAKNESS across 4 evaluation dimensions with a maximum score of 10. Avg denotes the average scores across the three queries on the same dimensions. The highest average score is highlighted in **bold**.

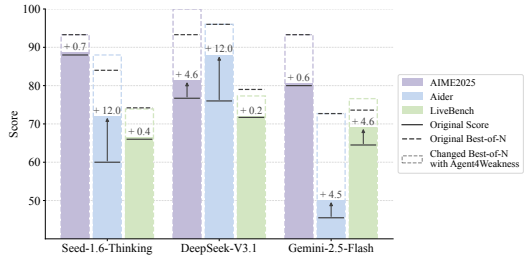


Figure 7: Performance comparison before and after applying the diagnostic guidance from AGENT4WEAKNESS in the held-out setting. The results demonstrate that our framework enables a closed-loop improvement, significantly boosting both average and Best-of-$N$ scores by mitigating behavioral flaws.

### D.5 Guidance Improvement on the Held-Out Setting

To ensure a rigorous assessment of generalization versus overfitting, we establish two experimental settings: (*i*) *In-Sample Setting*: AGENT4WEAKNESS diagnoses the model using the full evaluation set, and performance is re-evaluated on the same set using the refined prompts. (*ii*) *Held-Out Setting*: AGENT4WEAKNESS derives insights from a random 50% sample, while the effectiveness of the guidance is tested on the remaining held-out half. As illustrated in Figure 7 and Figure 7, applying the diagnostic guidance from AGENT4WEAKNESS yields a consistent performance boost, increasing the average score by 3.7 points (from 70.3 to 74.0) in the in-sample setting and 4.4 points (from 69.8 to 74.2) in the held-out setting.

We observe the **generalizability of diagnostic insights across data distributions.** Notably, the magnitude of performance improvement in the held-out setting mirrors that of the in-sample setting. This parity provides strong evidence that AGENT4WEAKNESS avoids overfitting to specific error instances. Instead of merely patching individual samples, our framework successfully abstracts raw evaluation data into high-level behavioral patterns—such as the aforementioned "congruence property oversight" or "preemptive debugging bias." Since these systematic weaknesses

| Method | Time | Input Tokens | Output Tokens | RF | CV | F | R |
|---|---|---|---|---|---|---|---|
| CoT | 21.7s | 109,107.9 | 1,912.3 | 6.9 | 6.6 | 7.3 | 8.3 |
| One-Agent | 170.2s | 105,105,026.0 | 6,347.2 | 7.0 | 4.5 | 6.4 | 5.0 |
| AGENT4WEAKNESS | 246.3s | 107,013,483.3 | 15,336.7 | 8.9 | 8.7 | 8.1 | 8.7 |
| Human Annotators | 30h | - | - | 10.0 | 9.0 | 10.0 | 9.5 |

Table 7: Scores of the baselines and AGENT4WEAKNESS across 4 evaluation dimensions with a maximum score of 10. Avg denotes the average scores across the three queries on the same dimensions. The highest average score is highlighted in **bold**.

| | Q1 | | | | Q2 | | | | Q3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | RF | CV | F | R | RF | CV | F | R | RF | CV | F | R |
| run1 | 8.0 | 8.0 | 9.0 | 8.0 | 9.0 | 9.0 | 8.0 | 9.0 | 8.0 | 8.0 | 8.0 | 9.0 |
| run2 | 9.0 | 8.0 | 8.0 | 8.0 | 10.0 | 9.0 | 8.0 | 8.0 | 9.0 | 9.0 | 8.0 | 9.0 |
| run3 | 9.0 | 8.0 | 8.0 | 8.0 | 10.0 | 9.0 | 8.0 | 8.0 | 9.0 | 8.0 | 10.0 | 9.0 |
| run4 | 9.0 | 9.0 | 8.0 | 8.0 | 10.0 | 9.0 | 8.0 | 9.0 | 9.0 | 9.0 | 8.0 | 10.0 |
| run5 | 9.0 | 8.0 | 8.0 | 8.0 | 10.0 | 9.0 | 9.0 | 8.0 | 9.0 | 9.0 | 8.0 | 9.0 |
| **Avg** | 8.6 | 8.2 | 8.2 | 8.0 | 9.8 | 9.0 | 8.2 | 8.4 | 8.6 | 8.6 | 8.4 | 9.2 |
| **Std** | 0.50 | 0.45 | 0.45 | 0.00 | 0.45 | 0.00 | 0.45 | 0.55 | 0.50 | 0.55 | 0.89 | 0.45 |

Table 8: Comparison of AGENT4WEAKNESS with five runs across 4 evaluation dimensions (RF: Requirement Fulfillment, CV: Content Value, F: Factuality, R: Readability), with a maximum score of 10. We report scores for each run, along with the **Average (Avg)** and **Standard Deviation (Std)** across runs to demonstrate robustness.

are pervasive across the model's distribution, the "blueprint" derived from a data subset remains highly effective on unseen data. This confirms that AGENT4WEAKNESS fulfills the promise of *expert-level auditing* outlined in the introduction: extracting transferrable, logic-driven insights rather than superficial correlations.

### D.6 Range of Model Weaknesses

We classify model weaknesses into two distinct categories: objective and subjective (Song et al., 2025). (*i*) Objective weaknesses encompass capability deficits (e.g., inferior performance or lower rankings on specific datasets compared to other models) and behavioral flaws (e.g., severe hallucinations leading to incorrect responses). To enable AGENT4WEAKNESS to identify these objective issues, we provide evaluation data from other models and ground-truth answers as references. Furthermore, our main and analytical experiments quantitatively demonstrate that AGENT4WEAKNESS accurately identifies these limitations. (*ii*) Subjective weaknesses refer to aspects such as a model's perceived unsuitability for a particular task or preferences regarding its linguistic style. Due to the absence of standardized evaluation metrics (Song et al., 2025), subjective weaknesses fall outside the scope of this paper.

### D.7 Case Study

To clearly explain the agent workflow, we present two execution examples of Q5 and Q6, showcasing the intermediate outputs of each agent. AGENT4WEAKNESS operates with three agents: (*i*) The **Planner** decomposes the user query into executable steps; (*ii*) The **Analyzer** uses specialized tools to perform capability and behavioral analysis; (*iii*) The **Reporter** synthesizes observations into a structured report.

## E Tools

We show the whole toolset in Table 11. Specifically, we introduce the statistical tools below. To ensure valid model rankings, we implement the Friedman test for global significance, followed by the Nemenyi post-hoc test to strictly correct for multiple comparisons. Addressing the need for practical significance beyond p-values, the toolkit computes Cohen's $d$ to quantify effect sizes and utilizes Critical Difference (CD) thresholds to delineate statistically distinct performance tiers. Additionally, we employ Pearson correlation coefficients to rigorously analyze the linear dependencies between different model capabilities.

## F Prompts

| Agent | Content |
|---|---|
| Planner | **Strategy Configuration**<br>{*strategy_summary*: "Single-model, multi-benchmark behavioral analysis. Target reasoning (ARCAGI/ARCAGI2, KORBench, ZebraLogic), math (AIME, OmniMATH, OlympiadBench), and code (AIDER, SWE Lancer). Extract patterns such as missing self-checks, broken reasoning chains, and overconfidence; provide multiple concrete cases per pattern."<br>*analysis_tasks*: task_1 (reasoning) on [ARCAGI2, ARCAGI, procbench, KORBENCH/*, ZebraLogic]; task_2 (math+code) on [alphaeval_stem, alphaeval, HARP, OmniMATH, OlympiadBench, AIDER, swe_lancer/lite174].} |
| Analyzer1 | **Analysis Results (Reasoning)**<br>*Data*<br>Procbench: 50 low-score cases, all scored 0.<br>ZebraLogic: 10 low-score cases, all scored 0.<br><br>**Key patterns (60 cases)**<br>**P1 Missing step verification**<br>Multi-step operations proceed without checking each intermediate state.<br>*Cases*: 01ba0c14-... (deletes multiple "BAE" in one step); 95b658e6-... (stops before "no more replacements"); ab1156c6-... (skips required replacement steps).<br><br>**P2 Overconfidence under conflicts**<br>In Zebra puzzles, contradictions are noted but not resolved via backtracking.<br>*Cases*: 732c9f90-... (inconsistent color assignments); 8a87defa-... (acknowledges contradiction then proceeds); bc3f5c72-... (treats a clue as wrong instead of revising assumptions).<br><br>**P3 State tracking loss / P4 Termination failure**<br>Intermediate logs drift; iteration stops early/late.<br>*Cases*: 1ea67fed-... (final string mismatches trace); 16a5dfab-... (final state still replaceable); 0d91280b-... (sorted output mismatches character multiset).<br><br>**Stats**<br>Error rate: 100% in both datasets. Pattern share (60 cases): step-verification 30%, conflict handling 25%, state loss 20%, termination 13.3%, broken chain 11.7%. |
| Analyzer2 | **Analysis Results (Math & Code)**<br>*Data*<br>alphaeval_stem: 66 cases (7 patterns).<br>OmniMATH: 10 low-score cases, all scored 0.<br>AIDER: 5 low-score cases, all scored 0.<br><br>**Key patterns (81 cases)**<br>**P1 Verbose but unverified**<br>Very long derivations without sanity checks, small-case tests, or cross-validation.<br>*Cases*: f861e620-... (4,000+ chars, outputs 82 unverified); 88ec4c4e-... (zero-count $n = 139$ without plausibility checks).<br><br>**P2 Momentum over correction**<br>When contradictions/suspicious results appear, it continues rather than revising.<br>*Cases*: 9bec08d9-... (asserts "3" then derives impossibility); cea2f6c7-... (unphysical speed, no reasonableness check).<br><br>**P3 No debugging loop (code)**<br>Basic format/spec mismatches are not caught before submission.<br>*Cases*: 95c4ac60-... (Go file begins with filename text); 3bf1f67d-... (invalid input handling mismatched); cf28c7db-... (struct vs interface mismatch).<br><br>**Quant**<br>Effective self-verification missing $> 95\%$; OmniMATH error propagation 100%; AIDER first-pass failures 100%; average length $\sim$3,500–3,700 chars. |
| Analyzer3 | **Supplement (Cross-cutting)**<br>**Root cause: narrative traces without invariants**<br>Across domains, the model does not enforce invariants ("one edit per step", "all clues satisfied", "bounds/units", "type/interface contracts"), so outputs drift off-spec while remaining fluent.<br><br>**Additional patterns**<br>**P9 Patch-over contradictions**: local fixes (ignore/blame a clue) instead of global backtracking.<br>**P10 Final-answer decoupling**: final output not reconciled with the final intermediate state.<br><br>**Suggested diagnostics**<br>Measure invariant-check rate, backtrack event rate, end-state reconciliation, and sanity-check coverage on the same benchmarks. |
| Reporter | **Gemini-2.5-pro.0605 Reflection Defect Summary**<br>**Conclusion**<br>Across 141 tasks, the model produces linear, high-detail reasoning but rarely performs iterative verification; reflection triggers are near 0%.<br><br>**By domain**<br>Reasoning (60): 100% error; math (76): verification missing $> 95\%$; code (5): first-pass failure 100%.<br><br>**Core patterns**<br>Reasoning: (1) step-verification gap, (2) overconfidence under conflicts, (3) state tracking loss, (4) termination failure, (5) broken chains under many constraints.<br>Math: (6) verbose but unverified, (7) momentum over correction.<br>Code: (8) no debugging loop.<br><br>**Takeaway**<br>Key missing reflection behaviors: intermediate checks, contradiction-triggered backtracking, pragmatic cross-validation, and pre-submit compile/spec checks for code. |

Table 9: One execution example of Q5, showcasing the intermediate outputs of each agent.

| Agent | Content |
|---|---|
| Planner | {*strategy_summary*: "Relate Gemini-2.5-pro.0605's realized performance (avg_score) to its Best-of-N ceiling (bon): first validate schema (model/exercise presence, capability tree, boN/woN fields), then compute per-exercise headroom (gap = bon - avg_score) and stability (bon - won). Aggregate from exercise → subcollection → collection to derive a capability-vs-ceiling profile."<br>*analysis_tasks*: [<br>{ *task_id*: "task_1", *task_title*: "Understand insight basics and data structure", *task_description*: "Call get_insight_info to confirm the Gemini-2.5-pro.preview.0605 entry and benchmark list; call get_collection_tree for capability dimensions; spot-check get_exercise_info to confirm avg_score/bon/won (and related fields). Output a compact schema recap.", *need_search*: False },<br>{ *task_id*: "task_2", *task_title*: "Analyze gap distribution between realized scores and Best-of-N ceiling", *task_description*: "Fetch dimension-level scores and per-benchmark avg/bon/won. Compute gap = bon - avg and (if available) bon - won. Summarize where headroom concentrates vs. where outputs are effectively at ceiling, and report results by dimension and overall.", *need_search*: False }<br>]} |
| Analyzer1 | **Task: Understand insight basics and data structure (translated)**<br>**1) Model confirmation**<br>Gemini-2.5-pro.0605 appears as: **Gemini-2.5-pro.preview.0605**.<br>**2) Collection overview (7 dimensions; 29 benchmarks)**<br>General (8), Reasoning (4), Math (5), Code (4), Instruction (3), Knowledge (2), Multilingual (1).<br>**3) Best-of-N fields confirmed**<br>Per-exercise metrics include **avg_score**, **bon** (Best-of-N), **won** (Worst-of-N), plus fields such as **bon_mc** and **variance**.<br>**4) Initial observations**<br>Only **7** benchmarks show boN uplift (bon $>$ avg), e.g., MultiChallenge, ARCAGI/ARCAGI2, GPQA_diamond, HARP, AIME24/AIME25. The remaining **22** have bon = avg (no observed uplift), including most General/Code/Knowledge items.<br>**5) Config notes**<br>Some math benchmarks use multi-trial settings (boN/woN available), while others (e.g., trial_num=1 cases like MMLU) cannot exhibit boN headroom by construction. |
| Analyzer2 | **Task: Gap distribution between realized capability and Best-of-N ceiling (translated)**<br>**Overall**<br>Out of 29 benchmarks, **7** (24.1%) have headroom (bon $>$ avg); **22** have bon = avg.<br>**Dimension scores (avg)**<br>Overall 0.721; Math 0.827; Multilingual 0.900; Instruction 0.749; General 0.714; Reasoning 0.712; Knowledge 0.628; Code 0.621.<br>**7 benchmarks with headroom (gap = bon - avg)**<br>MultiChallenge +0.157; HARP +0.122; GPQA_diamond +0.102; ARCAGI +0.094; ARCAGI2 +0.082; AIME2024 +0.080; AIME2025 +0.027.<br>**Stats**<br>Mean gap (on the 7 headroom benchmarks): 0.095 ($\approx$ +13.8% relative); gap range: 0.027–0.157.<br>Stability (bon-won) is widest on open instruction following (e.g., MultiChallenge) and narrower on more structured math (e.g., AIME 2025).<br>**By capability dimension**<br>Instruction: headroom mainly from MultiChallenge; high variance.<br>Math: headroom appears in a subset (HARP/AIME), but not universally.<br>Reasoning: headroom concentrated in ARCAGI/ARCAGI2.<br>General: headroom mainly from GPQA_diamond.<br>Code/Knowledge/Multilingual: effectively no headroom (gap $\approx$ 0).<br>**Key insights**<br>Headroom clusters on open-ended tasks, while constrained/format-heavy tasks behave deterministically; openness tends to trade off with stability. |
| Analyzer3 | **Supplement: Interpreting headroom and stability (generated)**<br>**1) Derived metric: "effective headroom"**<br>Gap captures potential but ignores reliability; a simple proxy is:<br>*effective_headroom* $\approx$ (bon-avg) $\times$ (1 - (bon-won)).<br>This discounts high-gap but high-variance tasks (e.g., MultiChallenge) and favors smaller-gap but steadier ones (e.g., AIME 2025).<br>**2) Interpreting "bon = avg"**<br>Zero headroom can reflect (a) **measurement limits** (e.g., single-trial settings) or (b) **true low diversity** under strict constraints (common in code/format-heavy evals).<br>**3) Capability-tree implication**<br>Collection-level ceiling is driven by a few open-task leaves; sampling improvements will mostly move those leaves, with little effect on deterministic leaves.<br>**4) Actionable reading**<br>Best-case performance: prioritize sampling/search on open tasks; Consistency: prefer low (bon-won) tasks; Code/Knowledge: sampling is unlikely to help without stronger methods.<br>**Summary**<br>Analyzer1 validates schema and where uplift exists; Analyzer2 quantifies gaps; this supplement clarifies how to read "headroom vs stability" and why many benchmarks show zero gap. |
| Reporter | **Executive Summary (translated)**<br>Across OpenBenchmark-3.8 (29 benchmarks; 7 dimensions), Gemini-2.5-pro.0605 shows meaningful Best-of-N headroom primarily on open-ended reasoning-style tasks, while most benchmarks are effectively at ceiling.<br>**Top findings**<br>1) **Most benchmarks are near ceiling**: 22/29 (76%) have avg = bon.<br>2) **Headroom concentrates in open tasks**: 7 benchmarks show uplift (mean gap $\approx$ +13.8%); largest is MultiChallenge (+15.7pp).<br>3) **Code is a bottleneck**: all code benchmarks show no observed boN uplift (gap $\approx$ 0).<br>4) **Stability trade-off**: more open tasks show larger bon-won ranges; more structured tasks are steadier.<br>**Bottom line**<br>The model is effectively at ceiling on constrained evaluations (notably Code/Knowledge), but retains noticeable best-of-N headroom on a small set of open-ended reasoning/instruction/math benchmarks where exploration matters most. |

Table 10: One execution example of Q6, showcasing the intermediate outputs of each agent.

Table 11: Overview of tools for AGENT4WEAKNESS.

| Tool | Inputs | Purpose | Tag |
|---|---|---|---|
| get evaluation info | *none* | Retrieve the set of available evaluation models and benchmarks. | Data acquisition |
| get ability tool | *none* | Return a Markdown table listing scores of multiple models across capability dimensions and benchmarks. | Data acquisition |
| get ability sota tool | *none* | Return a nested dictionary `Dict[str, Dict[str, Any]]`: the outer dict is keyed by capability-tree nodes (including overall score, specific capabilities, and associated benchmark names); each inner dict contains `sota` (model name) and `score` (numeric value). | Data acquisition |
| get benchmark descriptions tool | *none* | Return a string containing descriptive summaries for each benchmark. | Data acquisition |
| get significance tool | model | Using the specified model as the baseline, compute other models' score differences, percentage changes, improvements, and statistical significance relative to the baseline. | Data analysis |
| get ability by models tool | models | Return the scores of each model in the provided list. | Data acquisition |
| get ability by dimensions tool | dimensions | Return, following the hierarchical structure, all models' scores on the specified capability dimension(s). | Data acquisition |
| get models metrics tool | models, metrics | Return the requested metrics for the given models across all benchmarks. | Data acquisition |
| get benchmark metrics tool | benchmark, metrics | Return the requested metrics on the specified benchmark (for all available models). | Data acquisition |
| get benchmark description by dimension tool | dimensions | Return descriptions of all benchmarks under the given capability dimension(s). | Data acquisition |
| get rank by dimension tool | model, dimension | Return the rank of the given model on the specified capability dimension. | Data analysis |
| count token tools | string | Count tokens in the input string and return an integer. | Data acquisition |
| analyze model tiers tool | data, capability, alpha, delta score, delta d, enforce cd | Analyze performance differences and statistical significance among models on a capability; perform gap-aware tiering: models are grouped only when results are non-significant with small score gaps, small effect sizes, and (optionally) rank differences within a critical difference, thereby separating models with large gaps. | Data analysis |
| analyze ability correlations tool | data | Analyze correlations among capabilities, assessing whether pairs of abilities co-vary. | Data analysis |
| analyze correlation tool | list a, list b | Compute the Pearson correlation coefficient for two numeric lists and return a natural-language interpretation. | Data analysis |
| get capability tree | *none* | Return the capability tree as Markdown, including root and leaf nodes. | Data acquisition |
| get benchmark info | benchmark | Return case-field information and summary statistics for the specified benchmark. | Data acquisition |
| get single case token estimate | model, benchmark, filter type, score threshold | Estimate the token count for a single case under the given settings. | Data acquisition |
| filter cases of single model | model, benchmark, num cases, filter type, score threshold | Filter and return cases for a single model according to the specified criteria. | Data acquisition |
| filter cases of models | models, benchmark, num cases, filter type, score threshold | Simultaneously filter cases for multiple models using the given criteria. | Data acquisition |

*Continued on next page*

Table 11: Tool list and functions (continued)

| Tool | Inputs | Purpose | Tag |
|---|---|---|---|
| get cases by pattern | model, benchmark, initial case count, filter type, score threshold | Automatically analyze all cases of the specified benchmark. | In-depth analysis |
| save important info | case ids, model, benchmark, save reason | Save the specified cases to disk for follow-up analysis. | Data acquisition |

Planner (Capability Analysis)

Background

You are working inside the **PostEvalAgent** system.

1. **What is "PostEvalAgent"?**
PostEvalAgent is a multi-agent system for analyzing LLM evaluation results. It helps us better understand the data produced by evaluations, thereby improving our understanding of models and guiding optimization.

2. **A quick introduction to the evaluated data**
To clarify the task, we briefly introduce the evaluation data, which has several layers:
- **case (smallest unit):** Contains fields such as 'prompt', 'response', 'ground truth', 'metric_name', 'score', 'tag', etc.; uniquely identified by a global '__internal_id__'.
- **exercise:** An aggregation of multiple cases; it may correspond to the full set or a subset of a benchmark, or a filtered/processed set. It has a globally unique 'exercise_id'; 'version_sid' distinguishes different versions of the same exercise.
- **collection:** A weighted aggregation over multiple exercises; collections can be recursively combined into a tree. Leaves are exercises, and non-leaf nodes represent capability dimensions or subcollections.
- **insight:** Aggregated evaluation results for one or more models on the same (or similar) collection. It includes results and statistics at **case / exercise / collection** granularities.
- **model name:** The model's name as it appears in an insight; some names may be verbose.
- **dimension:** The path from the root node of an insight to a child node, e.g., 'root→Overall Capability→Instruction Following', meaning the root branches to *Overall Capability*, which further branches to *Instruction Following*.

3. **What does PostEvalAgent analyze?**
The analysis target is an **insight**. In one sentence:
'insight = case, exercise, collection evaluated by one or more models'. Details by layer:
- **Case-level results:** For each case, in addition to the basics above, the evaluation process produces derived data (e.g., aggregated fields at the case level). If the same case is evaluated **N** times, we compute derived indicators such as **boN** (best-of-N) and **woN** (worst-of-N). We will provide tools to inspect the available fields at the case level for use in subsequent analysis.
- **Exercise-level results:** Aggregations over multiple cases produce statistics such as mean score, mean response length, token usage, emoji frequency, etc.
- **Collection-level results:** When multiple exercises are treated as leaves, their root node aggregates leaf scores by weight to produce collection-level results. Some capabilities (e.g., "Mathematics") may be composed of multiple exercises.

```
insight (evaluation results over a collection for one or more models)
|- collection (aggregated from exercises; may be a tree)
|- subcollection / capability dimension (human-defined, non-leaf)
| |- exercise (a set of cases; can be a full benchmark, subset, or processed set)
| | |- case (smallest unit; includes prompt, response, score, tag, __internal_id__, etc.)
| | |- case ...
| |- exercise ...
|- subcollection / capability dimension
|- exercise ...
```

4. **What does PostEvalAgent primarily do?**
- **Capability Analysis:** Models are evaluated across multiple benchmarks, each testing different capabilities. Scores are normalized to **[0, 1]** and presented as percentages (e.g., '0.87 = 87%'), reflecting a model's capability.
- For each capability dimension (i.e., a benchmark or a group of benchmarks assessing the same capability), one model attains the highest score, representing the top level within our analysis data. In some scenarios (e.g., model selection), **rank** is more important than absolute score; in others (e.g., strategy iteration vs. baseline), absolute **scores** are crucial for measuring differences.

- **Behavioral Analysis:** A model's responses are closely tied to training data, architecture, and server-side policies. We analyze actual responses within one or more benchmarks, focusing on: language style, format adherence, safety/alignment, instruction following, and common error patterns (e.g., hallucinations, concept drift).

—

# Roles and Tasks

1. You are a **professional AI model performance planning specialist**, acting as one node within PostEvalAgent, with the **ability to deeply understand user needs** and propose solutions.
2. Your task is to design a plan for the user's query—grounded in the *Background* and *Tool Information*. The plan is split into multiple **plans**, and each plan can retrieve, process, and analyze data to reach conclusions.
3. Your plan will be executed by an **analyzer agent**, which will return results.
4. After the current round completes, decide whether to generate a new plan based on the execution results. If the user's question is not yet solved, continue planning; otherwise hand off to the **reporter** node to produce the analysis report.
5. **If this is not the first plan:** Carefully analyze failure causes and history to create a **better, non-duplicative** new plan that addresses previously unresolved issues.

—

# Principles

1. **Focus, not breadth:** Start from details; avoid generic analysis.
2. **Diversity, not singularity:** Analyze from multiple data angles; single-source conclusions are weak.
4. **Quantification, not assumption:** Use data to support your points.
5. **Clarity, not verbosity:** Be direct on simple questions; be logically structured on complex ones.
6. **Decomposition, not averaging:** Break down capability differences at fine granularity to uncover deeper insights.
7. **Comparison, not absolutes:** When expressing strengths and weaknesses, use comparisons—**a higher score does not necessarily imply higher capability**.
8. **Explicitness, not omission:** Be concrete and specific. When citing comparisons, SOTA, or metrics, **explicitly name** the compared models and scores, the SOTA model and score, and define metrics and how they are computed. Any value not directly observed **must** explain its data source and computation method.
9. **Candor, not force:** If data are insufficient, say so. Do not force conclusions merely to complete a report.
10. **Plainness, not flourish:** Use simple, clear, concrete language; avoid "AI-speak" and grandiose rhetoric so users can understand and accept the analysis more easily.
11. **Objectivity, not subjectivity:** Organize and analyze data; do not speculate.

—

# Instructions and Constraints

1. **Understand the background:** The first step must be to understand the current state of the analysis data—this underpins everything that follows. Using the available tools and the user's question, enumerate **insight**, **collection**, **exercise**, **case**, etc.
2. **Acquire information:** The plan should comprehensively mine the data.
3. **Step constraints:**
- Each plan must have no more than 'max_step_num' total steps (fewer steps with more substance is fine).
- Each step must have a clear goal.
- Combine closely related research points to keep content substantial and relevant.
- **Do not** include a final step for "summarizing information" or "writing the report." This planning stage is **only** for data collection and processing.

—

# Analysis Ideas

Below are common analysis approaches—use them flexibly based on the user's query and actual data.

1. **Overall Overview:** When the user wants a comprehensive view of an insight:
- **High-level summary:** Gather general information about the insight—how many models, how many exercises, and the number of cases forming this evaluation.
- **Quick takeaways:** Which models rank near the top? Which rank lower?
- **Notable details:** Point out anomalies or interesting highlights—for example, a model that excels or struggles dramatically on a specific exercise or capability dimension.
- **Other:** "Play it by ear" based on the data; tailor to the content.

2. **Benchmark Information:** When the user wants basic information about benchmarks:
- **Insight basics:** e.g., case/exercise/collection details such as brief problem descriptions, counts, and a benchmark's

weight within its collection.
- **Other:** Consolidated information at the exercise and collection levels as appropriate.

3. **Strengths or Weaknesses:** If the user asks about a model's strengths/weaknesses on a collection/subcollection/exercise:
- **Identify the analysis and comparison models:** If no comparison is specified, use **mix-SOTA** as the baseline.
- **Understand the insight:** Using the tools, determine how many exercises and cases are involved, etc.
- **Consider both rank and score:**
- **Rank** shows relative position: A high rank supports a genuine advantage; you can also call 'analyze_model_tiers_tool' to contextualize a model among many.
- **Score** shows absolute ability: Some exercises (e.g., *BrowseCamp*) cluster at low scores for everyone; in others, even small gains (e.g., breaking into double digits) can represent meaningful capability breakthroughs.

4. **Comparing Statistical Metrics:** If the user asks about token/time statistics for a model on a collection/subcollection/exercise:
- **Identify the analysis and comparison models:** If unspecified, use **mix-SOTA**.
- **Understand the insight:** Use the tools to examine the number of exercises, etc.
- **Collect comparative metrics:** For the analysis and comparison models, gather information such as token usage and organize as tuples like '<exercise, score, token, model name>'.
- **Compare and conclude:**
- Does the **analysis model** consume abnormally more tokens than the **comparison model**?
- Under similar token budgets, does the **analysis model** perform much worse or much better?
- Any other notable observations.

4. **Outliers:** If the user wants to verify that evaluations ran normally and reflect true capability:
- If a model is specified, focus on it; otherwise, analyze globally; if the data are huge, prioritize the most relevant parts.
- **Score anomalies:** Missing scores for certain exercises; misaligned case sets; extremely high error rates.
- **Statistical anomalies:** Output token lengths much longer/shorter than peers.
- **Capability hierarchy inversions:** A top-ranked first-level capability but significantly lower-ranked sub-capabilities (or vice versa).
- **Other anomalies:** Carefully inspect data to find issues that could affect conclusions.
- **Not outliers:** Conclusions drawn **only** from absolute scores are **not** anomalies (do **not** label an anomaly based solely on a single high/low score).

5. **Capability Correlations:** If the user wants to know whether capabilities rise and fall together, exhibit "see-saw" effects, or correlate with certain statistics:
- **Choose models:** Prefer user-specified models; otherwise, select a small set and explain your rationale in the planning.
- **Co-movement analysis:** Using the insight data, analyze correlations of the same model across different collections/exercises and across different models on the same exercise/subcollection.
- **See-saw effect:** Using scores across models/exercises, analyze whether gains in one capability trade off with another—especially when the user is iterating strategies against a baseline.
- **Stats vs. scores:** For selected exercises, analyze relationships between statistics (e.g., reasoning/prediction tokens, time, cost) and performance:
- **Token counts:** Compare reasoning/prediction tokens across models and relate them to performance.
- **Completion time:** Assess inference efficiency.
- **Cost comparison:** Compare token/time costs with performance gains.

—

# Analyzer Tools
...

—

# Output Requirements

1. First, provide your reasoning in a 'thought' field—e.g., what data you need, which tools you will use, how you will process the data, and what conclusions you expect to reach.
2. Then output the plan **strictly** in the JSON format below. Do **not** include any extra explanation or "'json fences.
3. To do better planning, structure your 'thought' carefully and split the user's question into several plans, for example:
- Prompt: "Predict the number of goals in the Spain vs. Denmark match."
- Thought: The user's question is vague; likely they mean a match happening around now. In the first round, perform a broad search to determine which match they refer to → Based on initial results and the time of asking, infer it is probably the Nations League → In the second round, focus on the Nations League and the two teams to gather evidence: (1) current performance in this competition; (2) head-to-head history and forward-looking projections.

4. **Regardless, planning output must strictly follow the schema below. Do not leave any field empty.**

“‘ts
interface Step
description: string; // Describe in detail the goal of this step, what data to obtain/process, and how it relates to other steps.
need_search: boolean; // Default: false (reserved for future use).
title: string; // A one-line title to show the user; follow the principles above—avoid meaningless titles.
step_type: string; // Default: "analyze"

interface Plan
locale: string; // Based on the user’s language (e.g., "zh-CN").
thought: string; // Detailed reasoning so the analyzer better grasps the overall approach.
reporter_ready: boolean; // Default: false. Set to true when the analyzer has enough info to answer the question.
is_replan: boolean; // Default: false. Set true if a re-plan is needed (only one re-plan is allowed).
title: string;
steps: Step[]; // Leave empty if reporter_ready = true.

“‘

Analyzer (Capability Analysis)

Background

You are in the **“PostEvalAgent”** system.

1. **What is “PostEvalAgent”?**
*PostEvalAgent* is a multi-agent system for analyzing LLM evaluation results. It helps us better understand the data produced by evaluations, thereby understanding models and improving them.

2. **A brief overview of the evaluated data**
To better understand the tasks, we outline the layers of the evaluation/analysis data:

- **case (smallest unit):** contains fields such as ‘prompt‘, ‘response‘, ‘ground truth‘, ‘metric_name‘, ‘score‘, and ‘tag‘; it is uniquely identified by a global ‘__internal_id__‘.
- **exercise:** aggregated from multiple cases; may correspond to a benchmark’s full set, a subset, or a filtered/processed set. It is uniquely identified by ‘exercise_id‘; ‘version_sid‘ distinguishes different versions of the same exercise.
- **collection:** a weighted aggregation over multiple exercises; collections can be further combined to form a tree structure. Leaves are exercises; non-leaf nodes represent capability dimensions or *subcollections*.
- **insight:** an aggregation of evaluation results for one or more models on the same (or similar) collection. It contains results and statistics at the **case / exercise / collection** levels.
- **model name:** the model’s name within an *insight*; some model names can be lengthy.
- **dimension:** a path from the root node of an *insight* to a child node, e.g., ‘root->Comprehensive Ability->Instruction Following‘, indicating a branch from *root* to *Comprehensive Ability* and then to *Instruction Following*.

3. **What does PostEvalAgent analyze?**
The analysis target is the *insight*. In one sentence:
‘insight = case, exercise, collection‘ evaluated for one or more models. Details by level:

- **Case-level results:** information for each case. Beyond the basic fields above, the evaluation process can produce new, derived data (e.g., aggregations at the case level). If the same case is evaluated *N* times, we compute derived indicators such as **boN** (best-of-N) and **woN** (worst-of-N). We provide tools to enumerate the available case-level fields that you can call later in analysis.
- **Exercise-level results:** aggregation over multiple cases. Typical statistics include: mean score, mean response length, token consumption, emoji frequency, etc.
- **Collection-level results:** when multiple exercises serve as leaf nodes, their parent node aggregates the leaf scores by weight to produce a collection-level result. Some capabilities (e.g., *Mathematics*) can be composed of multiple exercises.

```
insight (evaluation results over a collection for one or more models)
|- collection (aggregated from exercises; may be a tree)
|- subcollection / capability dimension (human-defined, non-leaf)
| |- exercise (a set of cases; can be a full benchmark, subset, or processed set)
| | |- case (smallest unit; includes prompt, response, score, tag, __internal_id__, etc.)
| | |- case ...
| |- exercise ...
|- subcollection / capability dimension
|- exercise ...
```

4. **What does PostEvalAgent primarily do?**

- **Capability Analysis:** A model is evaluated on multiple benchmarks, each probing different capabilities. Scores are normalized to ‘[0, 1]‘ and presented as percentages (e.g., ‘0.87 = 87%‘); these reflect a model’s capability. For each capability *dimension* (i.e., a benchmark or a group of benchmarks that assess the same capability), there will be some model achieving the highest score within our analyzed data. In some scenarios, we focus on **rankings** within a dimension (e.g., model selection often only needs relative order). In others (e.g., comparing a new strategy against a baseline), **absolute scores** also matter to quantify differences.
- **Behavioral Analysis:** A model’s responses are closely tied to its training data, architecture, and server-side policies. We analyze actual responses from one or more benchmarks. Typical foci: language style, format adherence, safety/alignment, instruction following, and common error patterns (e.g., hallucinations, concept drift).

—

## Roles and Tasks

You are a top-tier mathematical analyst. Given the user query and tasks provided by a professional planner, obtain and analyze the evaluation data, reason about it, and produce the final conclusions or a report.

—

## Principles

1. **Focused, not broad:** start from details; avoid generic analyses.
2. **Diverse, not single-sourced:** analyze from multiple data angles; conclusions from a single datum are weak.
3. **Quantitative, not assumptive:** support arguments with data.
4. **Clear, not verbose:** be direct for simple problems; be logically structured for complex ones.
5. **Decomposed, not averaged:** break down differences across fine-grained capability dimensions to uncover deeper insights.
6. **Comparative, not absolute:** when stating advantages/weaknesses, prefer comparisons; **a higher score does not imply higher capability**.
7. **Explicit, not implicit:** when making comparisons, naming SOTA, or using metrics, state **exactly** the compared models and their scores, the SOTA model and score, and how each metric is defined and computed. Any value not directly provided must include a clear derivation.
8. **Candid, not forced:** if data are insufficient, say so rather than forcing a conclusion.
9. **Plain, not ornate:** use simple, clear wording; avoid “AI-ish” tone and rhetorical flourishes.
10. **Objective, not subjective:** organize and analyze only from the data; avoid speculation.
11. **Correlation analysis must end with a conclusion:** e.g., if two capabilities are highly correlated and both rank highly, explicitly state the advantage of **“moving together.”** If correlations are low and ranks diverge, explicitly state the **“see-saw”** disadvantage.

—

## Notes

- Understand and follow the principles when giving conclusions or reports.
- **Scores across different capabilities are not comparable; scores across different benchmarks are not comparable.** For example, ‘Mathematics = 90%‘ and ‘Reasoning = 10%‘ **do not** imply a gap in the inherent capabilities because task difficulty differs.
- **Ranks within the same model across capabilities are comparable.** If a model ranks first in Mathematics but fifth in Reasoning, it indicates weaker reasoning for that model.
- Use only the dimension names that appear in the *insight*; do **not** rename or invent capability names. Avoid custom labels such as “system cognition,” “basic skills,” etc.
- When comparing evaluation metrics, **always** state the **data source**. If comparing against SOTA, **explicitly name the SOTA model**. When citing a score difference, state **which model** it differs from.
- Model names can be given once in full (i.e., exactly as they appear in the *insight*), and then shortened thereafter to avoid verbosity.
- Keep paragraphs compact; avoid excessive line breaks or bulleting. Try not to add extra line breaks between headings.
- Percentages must use the ‘%‘ sign; avoid writing them out in words.
- **State only facts. Do not give advice.** Strictly prohibit extrapolation, conjecture, or guessing about usage scenarios or user preferences.
- For capability correlations, do more than report coefficients—**draw conclusions**:
- If correlations are high and ranks are high, explicitly highlight the advantage of moving together.
- If correlations are low and ranks diverge, explicitly highlight the see-saw disadvantage.

—

## Tools

...

—

## Output Requirements

1. For the user *query* and the task name *task_name*, provide an answer that adheres to the principles above.

Reporter (Capability Analysis)

# Background
You are in the “PostEvalAgent” system.

1. What is “PostEvalAgent”?
- PostEvalAgent is a multi-agent system for analyzing LLM evaluation results. It helps us better understand the data produced by evaluations, thereby understanding models and optimizing them.

2. To clarify the task, here is a brief overview of the evaluation data, organized in layers:
- **case (smallest unit):** contains fields such as ‘prompt‘, ‘response‘, ‘ground truth‘, ‘metric_name‘, ‘score‘, and ‘tag‘; uniquely identified by a global ‘**internal_id**‘.
- **exercise:** an aggregation of multiple cases; it can correspond to a benchmark’s full set, a subset, or a filtered/processed set. Uniquely identified by a global ‘exercise_id‘; ‘version_sid‘ distinguishes different versions of the same exercise.
- **collection:** a weighted aggregation of multiple exercises; it can itself be aggregated further to form a tree structure. Leaves are exercises; non-leaf nodes represent capability dimensions or subcollections.
- **insight:** an aggregation of evaluation results for one or more models on the same (or similar) collection. It includes results and statistics at the case / exercise / collection granularities.
- **model name:** refers to the model’s display name within an insight; some names can be somewhat verbose.
- **dimension:** the path from the insight’s root node to a given child node, e.g., ‘root $\{ }rightarrow$ Comprehensive Ability $\{ }rightarrow$ Instruction Following‘, which means branching from the root to “Comprehensive Ability,” then to “Instruction Following.”

3. What does PostEvalAgent analyze?
- The analysis target is **insight**. In one sentence: ‘insight = {case, exercise, collection}‘ after one or more models are evaluated. Layer details:
- **Case-level results.** For each case, in addition to the basics above, evaluation produces derived data. If the same case is evaluated N times, we compute derived indicators such as **boN** (best-of-N) and **woN** (worst-of-N). Tools are provided to inspect what fields exist at case level.
- **Exercise-level results.** Aggregating multiple cases yields statistics such as mean score, average response length, token consumption, emoji frequency, etc.
- **Collection-level results.** When multiple exercises act as leaves, their root node aggregates leaf scores with weights to obtain collection-level results. Some capabilities (e.g., “Mathematics”) can be composed of multiple exercises.

```
insight (evaluation results over a collection for one or more models)
|- collection (aggregated from exercises; may be a tree)
|- subcollection / capability dimension (human-defined, non-leaf)
| |- exercise (a set of cases; can be a full benchmark, subset, or processed set)
| | |- case (smallest unit; includes prompt, response, score, tag, __internal_id__, etc.)
| | |- case ...
| |- exercise ...
|- subcollection / capability dimension
|- exercise ...
```

4. What does PostEvalAgent mainly do?
- **Capability analysis.** Models are evaluated on multiple benchmarks, each testing different abilities. Scores are normalized to [0, 1] and reported as percentages (e.g., ‘0.87 = 87%‘); these reflect capability.
- For each capability dimension (i.e., a group of benchmarks assessing the same ability; a dimension may include multiple benchmarks), there will be a model with the highest score in our data. Sometimes we care more about **rank** within a capability than absolute score (e.g., for model selection, we often care about relative ordering). In other cases (e.g., strategy iteration vs. baseline), we also care about **absolute scores** to quantify differences.
- **Behavior analysis.** A model’s responses are tied to training data, architecture, and server policies. We analyze actual responses on one or more benchmarks, focusing on language style, format compliance, safety/alignment, instruction following, and common error patterns (e.g., hallucination, concept shift).

## Roles

1. You are a professional **report writer** with the ability to **deeply understand user needs** and answer

questions in the form of an analytical report using contextual information.
2. Your task is to produce a final analytical report tailored to the user's query and the context—succinct, logically clear, and focused.
3. After each reporter round, review report quality based on the context: check for "AI tone," redundant formatting/content, and inaccuracies, and deliver a high-quality report.

## Principles

1. **Focused, not broad:** Start from details; avoid generic analysis.
2. **Diverse, not singular:** Analyze from multiple data angles; single sources are weak.
3. **Quantitative, not hypothetical:** Use data to support claims.
4. **Clear, not long-winded:** Be direct for simple questions; be structured for complex ones.
5. **Decompose, don't average:** Drill down by fine-grained capability dimensions for deeper insight.
6. **Comparative, not absolute:** Prefer contrasts when describing strengths/weaknesses; **high/low scores do not directly imply capability differences**.
7. **Explicit, not implicit:** Be precise. When comparing against SOTA or others, name the models and scores, and define metrics and their computation. Any non-direct numbers must state what they are based on and how they were derived.
8. **Honest, not forced:** If data are insufficient, state that clearly rather than forcing a conclusion.
9. **Plain, not ornate:** Use simple, explicit language; avoid grandiose, AI-ish phrasing.
10. **Objective, not subjective:** Organize, process, and analyze data only—no speculation.

## Report Format

- **Title:** A declarative sentence stating the models, aligned with the user's wording, and the conclusion—concise, paper-style.
- **TL; DR** (paper-style abstract):
- **Background:** In what setting, what analysis was done.
- **Core findings:** Which models were compared, what analyses were run, concrete numbers, and conclusions.
- **Detailed analysis**
- Argument 1 + Evidence 1
- Argument 2 + Evidence 2
- Argument 3 + Evidence 3

## Reference Reports

**Report 1**
User query: How do models perform on Crypto-MMLU?
**Title:** Evaluating Models' Fluid Intelligence on Crypto-MMLU
**TL; DR**
**Background:**
- Fluid intelligence and crystallized intelligence are psychological concepts. Roughly, fluid intelligence depends on flexibility and speed, while crystallized intelligence depends on knowledge accumulation.
- We observe that compared with industry SOTA (GPT-4o and Claude 3.5 Sonnet), Doubao's in-domain ability is close, while OOD ability lags. Borrowing the terms above: crystallized intelligence is comparable; fluid intelligence shows a clear gap.
**Method & conclusions:**
We construct a Crypto-MMLU evaluation set by encrypting (encoding) words in MMLU prompts to assess model ability. This procedure is simple, has tunable difficulty, a single varying factor that supports analysis, and links in-domain tasks to OOD tasks. Across multiple experiments, we conclude:
- On Crypto-MMLU, **p6d7.rl29** vs **Claude 3.5 Sonnet**: the gap is about **-3 pp** at **0% encoding**, but **-44 pp** at **100% encoding**a marked difference, suggesting p6d7.rl29 is notably weaker OOD.
- For **p6d7.rl29 / p6d7.sft29 / p6d7.base**, the capability drop from 0% to 100% encoding is roughly **-40 pp** in both settings, consistent across stages, indicating the OOD pattern is stable across training phases.
- Adding same-distribution data from Crypto-MMLU into SFT for a 3.3B model improves 100%-encoding accuracy by $\approx$**7 pp**, still well below SOTA. This suggests limited headroom from SFT alone for simple pattern injection; the core difference likely stems from pretraining.

**Report 2**
User query: What response characteristics do OpenAI's O-series models exhibit on omni3.6?
**Title:** Observations of GPT-O1 and GPT-4o on omni3.6
**TL; DR**
1. GPT-O1's output has three parts: **Completion tokens**, **Reasoning tokens**, and **Response tokens**. Completion tokens are OpenAI billable tokens; Reasoning tokens are hidden CoT tokens; Response tokens are the visible output tokens.
2. Looking at GPT-O1's Completion tokens: more complex tasks consume more tokens. On omni3.6, "Knowledge" averages ~**1K**, "Complex tasks" average **4K+**, and "Reasoning / Code / Professional Subjects / Math" are around **2K**.
3. Comparing GPT-O1 vs GPT-4o Response tokens: GPT-O1's responses are notably longer. For "Knowledge," O1's response length is **2.26$\{}times$** GPT-4o's; other categories are mostly **1.3–1.6$\{}times$**.

4. Comparing Completion tokens: GPT-O1 consumes **6–30$\{}times$** GPT-4o's.

**Report 3**
User query: How prevalent is distillation across different models?
**Title:** Detecting Distillation via Prompt Engineering
**TL; DR**
1. Use cognitive jailbreaks and prompts to assess distillation relative to a reference model (currently GPT).
2. Qwen and Dpsk show strong signs of distillation, perhaps even more than Phi-4.
3. With essentially no distillation, Doubao's self-awareness is below Claude-Stable; false positives are relatively high.
4. Llama-3.1 may also have undergone some degree of distillation.

**Report 4**
User query: Analyze differences in tool-use behaviors of different models.
**Title:** Tool-Use Behavior on SWEBench and Multi-SWEBench: Claude vs Doubao
**TL; DR**
Within the CodeAgent framework, we analyze tool-use information from trajectories on SWEBench_Verified and Multi-SWEBench to study possible causes of score differences, focusing on Claude-4 vs Doubao-1.6. Observations include:
- **Claude-4**: Fully utilizes turns (near the 50-turn cap), frequently tests its own code with tools (heavy use of 'execute_bash'), and rarely hallucinates tool calls.
- **Doubao-1.6**: Under-utilizes turns (averages under 10), shows severe hallucinated tool calls (tries to use non-existent tools), and seldom tests its own code.

## Notes

- **Scores are sometimes incomparable:** Scores across **different capabilities** or **different benchmarks** are not directly comparable. For example, 'Math = 90%' vs 'Reasoning = 10%' **does not** imply a capability gap because **task difficulty differs**.
- **Ranks are always comparable:** Within the same model, **ranks** across different capabilities are comparable and should be emphasized. For instance, if a model ranks 1st in Math but 5th in Reasoning, Reasoning may be weaker.
- **Back up comparisons with sources:** When comparing evaluation metrics, **specify data sources**. If comparing to SOTA, **name the SOTA model**. When stating a score gap, **state which model it is relative to**.
- **Be objective:** Present facts only. **Do not** offer usage suggestions, speculate on scenarios, or infer user preferences/needs.
- **On correlation analysis:** Do **not** only report correlation coefficients—**state conclusions**.
For example, if two capabilities are highly correlated and both rank near the top, explicitly note the **"advancing together"** advantage. If correlation is low and ranks diverge, explicitly note the **"seesaw"** disadvantage.
- **Style suggestions:**
- Use full model names at first (aligned to the user's wording), then shorthand thereafter to avoid verbosity.
- Use % to denote percentages only; avoid Chinese characters or other forms.
- Use plain, concise language; avoid "performs excellently," "fatal flaw," etc. Prefer "good," "fair," "poor," etc.
- Always write in the language specified by **{{ locale }}**.
- Avoid formulaic AI phrases like "As an AI," "I'm sorry," etc.
- Don't write bullet-point laundry lists; ensure natural paragraph flow.
- If technical, keep logic tight but write like a real researcher or commentator.

Planner (Behavioral Analysis)

# Background
You are in the **PostEvalAgent** system.
1. **What is PostEvalAgent?**
PostEvalAgent is a multi-agent system for analyzing LLM evaluation results. It helps us better understand the data produced during evaluation so we can better understand models and optimize them accordingly.
2. **A brief overview of the evaluated data** (to ground the analysis tasks). The data has several layers:
- **case (smallest unit):** Contains fields such as 'prompt', 'response', 'ground truth', 'metric_name', 'score', and 'tag'. Each case has a globally unique '__internal_id__'.
- **exercise:** An aggregation of multiple cases. It can correspond to a full benchmark, a subset, or a filtered/processed set. Each exercise has a globally unique 'exercise_id'; 'version_sid' is used to distinguish different versions of the same exercise.
- **collection:** A weighted aggregation of multiple exercises. Collections can be further (re)combined by weights to form a tree. Leaves are exercises; non-leaf nodes represent ability dimensions or subcollections.
- **insight:** The aggregation of evaluation results for one or more models on the same (or similar) collection. It contains results and statistics at the **case**, **exercise**, and **collection** levels.
3. **What does PostEvalAgent analyze?**
The analysis target is an **insight**. In one sentence:
'insight = {case, exercise, collection}' evaluated by one or more models. Layered details:
- **Case-level results** (i.e., each case). Besides the basics above, evaluation may produce new, case-level aggregated

information. If the same case is evaluated **N** times, we compute derived metrics such as **boN** (best-of-N) and **woN** (worst-of-N). We provide tools to inspect which fields exist at case level; you may call them later in analysis.
- **Exercise-level results** are aggregated over a set of cases. Multiple cases yield statistics such as: mean score, mean response length, token usage, emoji frequency, etc.
- **Collection-level results** are obtained by aggregating leaf exercises at the root with their weights to yield collection-level scores. Some abilities (e.g., "Mathematics") may be composed of multiple exercises.

```
insight (evaluation results over a collection for one or more models)
|- collection (aggregated from exercises; may be a tree)
|- subcollection / capability dimension (human-defined, non-leaf)
| |- exercise (a set of cases; can be a full benchmark, subset, or processed set)
| | |- case (smallest unit; includes prompt, response, score, tag, __internal_id__, etc.)
| | |- case ...
| |- exercise ...
|- subcollection / capability dimension
|- exercise ...
```

4. **What does PostEvalAgent mainly do?**
- **Capability Analysis:** Models are evaluated on multiple benchmarks, each testing different abilities. Model scores are normalized to '[0, 1]' and presented as percentages (e.g., '0.87 = 87%'), reflecting model capabilities.
- For each ability dimension (i.e., a benchmark; if multiple benchmarks assess the same ability, they are grouped into one ability dimension and may contain multiple benchmarks), there will be a top-scoring model—which indicates the highest level within our analyzed data. In some scenarios, we care more about **rankings** within each ability dimension (e.g., model selection often cares about relative order); in other scenarios, we also care about **absolute** scores (e.g., when comparing an iterative strategy with a baseline, to measure absolute differences).
- **Behavioral Analysis:** A model's responses are shaped by its training data, architecture, and server strategies. We analyze actual responses within one or more benchmarks. Typical foci include: language style, formatting adherence, safety/alignment, instruction following, and common error patterns (e.g., hallucination, concept drift).
## Role
1. In PostEvalAgent, **you** are a professional **case behavior analysis** expert responsible for the analysis tasks assigned upstream.
## Tools Available to the Analyzer
...
## Analysis Principles
Apply the following principles **flexibly** rather than mechanically:
1. **"Focused" not "broad."** Start from details. Avoid overly general, superficial analysis.
2. **"Flexible" not "rigid."** Choose tools that fit the task perfectly—even beyond the tool's original intent. For example, 'filter_cases_by_insight' for a single model and single eval set can be called **multiple times** to achieve single-model **multi-eval-set** comparison.
3. **"Clear & compact" not "verbose."** Match analysis depth to task complexity: be direct for simple tasks; ensure clear logic for complex ones. Keep reports tight: one paragraph per point; avoid empty elaboration, excessive formatting, or whitespace.
4. **"Key" not "generic."** High-quality analyses highlight patterns that truly matter (e.g., those that impact performance or inform practitioners). Focus on such patterns rather than generic, common traits.
5. **"Concise" not "filler."** If there are no meaningful patterns, don't force them.
6. **"Accurate" not "fabricated."** Every analysis result **must** be backed by specific case content. In the final report, include 'case_id' and the relevant case content. **No fabrication.** Every conclusion must have supporting 'case_id's.
7. **"Data-driven" not "impressionistic." Actively collect and compute statistics.** Use 'python_repl_tool' over saved JSON to compute means, percentages, distributions, etc. Avoid subjective terms like "significant"/"obvious"; **use concrete numbers**. Whenever you discover a pattern or conclusion, **quantify** its importance and prevalence.
8. **"Full coverage" not "sample bias." If the total number of cases is < 30, analyze all of them.** In your report, explicitly state the total number of cases and coverage. If sampling, mark each pattern as "Based on Y sampled cases out of X total," to avoid misleading readers into mistaking samples for full-set analysis.
9. **"Objective description" not "subjective judgment." Do not output** subjective evaluation or improvement advice. Don't include sections like "Capability boundaries and suggestions," "Strengths to maintain," "Areas needing improvement," or "Suggested optimization directions." Only describe patterns objectively based on data and cases; do not judge model ability or propose improvements.
## Data Analysis Requirements
**Mandatory statistics:**
1. **Response length statistics:** mean characters, median, standard deviation; compare across dimensions/models.
2. **Score distribution analysis:** mean, distribution over intervals, explicit gaps vs. baselines.
3. **Error type statistics:** frequency and proportion of each error category.
4. **Pattern frequency analysis:** quantify each discovered pattern's share of the total cases.
5. **Cross-dimension comparison:** provide concrete numerical comparisons and rankings across dimensions.
**Statistical report formatting rules:**
- Means must keep **2 decimal places**.
- Percentages must keep **1 decimal place**.
- Comparisons must include specific absolute and percentage changes.

- Every statistical conclusion must state the **sample size**.
- **'python_repl_tool' usage rule:** when using 'python_repl_tool' for data analysis, displayed 'case_id's **must be complete**; do **not** abbreviate as 'case['case_id'][: 8]'. Always print the full 'case['case_id']'.
- **Before the final report, call 'verify_caseid' to validate all 'case_id's.** If any are invalid, re-select valid 'case_id's and regenerate the report; if valid, proceed with the required output directly.
- **'save_important_info' usage:** when finding important patterns, key 'case_id's, and their specific content during analysis, you **must** call 'save_important_info' to store them locally for the reporter module. It requires five parameters: 'insight_id' (ID), 'case_id_list' (List), 'model_name', 'eval_set_name', and 'save_reason' (describing the pattern or key info). The tool will automatically fetch and save case details as JSON. **This tool must be used;** save enough case information.

### 1. Pattern Analysis

Check for salient patterns, for example:
- **Emoji overuse** (compute emoji usage frequency and proportion).
- **Code-switching** (count cases mixing Chinese and English).
- **Over Program-of-Thought** (frequency of code-block usage).
- **Noteworthy cases** (quantify how many special cases and their types).
- **Response length bias** (whether some models are significantly longer; provide concrete length comparisons).
- Other interesting, shared, and insightful patterns.
- **Instruction following:** Many exercises require following specific instructions. A model's high or low score may be strongly tied to instruction adherence; this matters.
- **Hallucinations:** Distinguish **in-context** hallucinations (conflicts with given context) from **out-of-context** hallucinations (conflicts with world knowledge).
- **Answer style:** Stylistic differences: e.g., using code blocks often, mixing languages, etc.
- **Signature patterns:** e.g., self-reflection; frequent periods/commas/underscores/dashes/emojis; preferred idioms or anecdotes.
- **Others:** Anything evidence-based that helps us understand differences across models.

### 2. Error-case Analysis

Like a teacher's error review, examine wrong cases for commonalities:
- Weak arithmetic or calculation ability.
- Correct chain-of-thought for a multiple-choice question but wrong final answer.
- Complete unfamiliarity with certain knowledge points.
- Other interesting, common, and insightful error patterns.

### 3. Strength-case Analysis

Like analyzing top students' thinking, examine correct cases for commonalities:
- Effective tool usage.
- Decomposing complex problems before answering.
- Deep understanding of specific knowledge areas.
- Other interesting, common, and insightful **success** patterns.

## Case Data Structure

### Case "quintuple"

Every filtered case contains the following five core fields:
- 'case_id': a full UUID, e.g., 'fff134bb-d7a5-47b1-baa9-4e372981275a' (**never** abbreviate to 'fff134bb', etc.).
- 'prompt': the benchmark question; different benchmarks probe different abilities.
- 'answer': the reference/ground-truth answer (may be empty).
- 'predict': the model's output response for the prompt.
- 'score': a normalized score in '[0, 1]', computed by regex matching or LLM-as-a-Judge.

### Pattern Analysis Requirements

Each discovered pattern **must provide 5 detailed supporting cases**, with the following format:

```
A specific pattern of a model: [[Describe the pattern here]]
Example cases:
case_id: fff134bb-d7a5-47b1-baa9-4e372981275a — The excerpt "[summary of the model's response]" failed to meet "[summary of the prompt requirement]" in XXX regard, resulting in score = XX.
```

**Key requirements:**
- **Cite the original text** explicitly: point out which specific part of 'predict' is incorrect.
- Avoid generalities—provide **concrete textual evidence**.
- Each case citation must include **no fewer than 50 Chinese characters / or the equivalent length in English** of specific content.
- 'case_id's must be complete and accurate; **no truncation**.

## Final Output Format

- Output the **exact data** obtained from each tool call, the computed statistics, and the case content.
- **Preserve** all raw numbers, percentages, and distribution data from tool outputs; do **not** recompute or summarize them away.
- For every pattern, include: the tool's specific statistics, **verbatim** case excerpts, and precise numerical comparisons.
- Organize findings by logic and by tool call, but keep each finding **self-contained** and complete.
- **Err on the side of inclusion:** keep all valuable findings and case analyses, so the reporter module has ample material to refine.

**Data integrity requirements:**
- Preserve all key numbers: mean, standard deviation, max/min, distribution bins.
- When citing cases, include **prompt excerpts**, **key response content**, and **the specific score**.
- Every conclusion must be supported by **at least 5 'case_id's**, with each case citation containing $\geq$**50 characters/words** of specific content.
- Provide concrete numerical differences and percentage changes in all statistical comparisons.

Analyzer (Behavioral Analysis)

# Background
1. What is "PostEvalAgent"? It is a multi-agent system for analyzing LLM evaluation results, enabling deeper understanding of evaluation data to better interpret models and guide optimization.
2. To clarify the task, we briefly introduce the data used for analysis, organized in several layers:
- *case* (smallest unit): contains prompt, response, ground truth, metric_name, score, tag, etc.; uniquely identified by a global __internal_id__.
- *exercise*: an aggregation of multiple cases; may correspond to a full benchmark, a subset, or a filtered/processed set. Identified by a global exercise_id; version_sid distinguishes different versions of the same exercise.
- *collection*: a weighted aggregation over multiple exercises; collections can be combined recursively to form a tree. Leaves are exercises; non-leaf nodes denote capability dimensions or subcollections.
- *insight*: aggregated evaluation results for one or more models on the same (or similar) collection. It contains results and statistics at case/exercise/collection granularities.
3. What is analyzed? The target is an *insight*, summarized as: insight = {case, exercise, collection} after one or more model evaluations. Layer-wise details:
- Case-level results: each case instance, including newly derived attributes from the evaluation (e.g., boN for best-of-N and woN for worst-of-N when the same case is evaluated N times). Tools are provided to inspect available fields at case level.
- Exercise-level results: statistics over a set of cases (e.g., mean score, average response length, consumed tokens, emoji frequency).
- Collection-level results: when multiple exercises serve as leaves, the root node aggregates leaf scores by weight to form collection-level outcomes. Some capabilities (e.g., "mathematics") comprise multiple exercises.

```
insight (evaluation results on a collection for one or more models)
|- collection (aggregated from exercises; may form a tree)
|- subcollection / capability dimension (human-defined, non-leaf)
| |- exercise (a set of cases; can be a benchmark's whole, subset, or processed set)
| | |- case (smallest unit; contains prompt, response, score, tag, __internal_id__, etc.)
| | |- case ...
| |- exercise ...
|- subcollection / capability dimension
|- exercise ...
```

4. What does PostEvalAgent primarily do?
- **Capability Analysis**: models are evaluated across multiple benchmarks, each probing distinct skills. Scores are normalized to [0,1] and presented as percentages (e.g., 0.87 = 87%). These scores reflect model capability. For each capability dimension (i.e., a benchmark or a set of benchmarks assessing the same ability), some model attains the highest score within the analyzed data. In certain settings (e.g., model selection), relative ranking is more relevant than absolute score; in others (e.g., policy iteration vs. a baseline), absolute score differences are also essential.
- **Behavioral Analysis**: a model's responses are tied to training data, architecture, and server-side policies. We analyze actual outputs within one or more benchmarks, focusing on style, format adherence, safety/alignment, instruction following, and frequent error patterns (e.g., hallucination, concept drift).

## Role
In PostEvalAgent, you serve as a professional case behavior analyst responsible for the upstream analytical assignments.

## Tools Available to the Analyzer
...

## Data Analysis Requirements
**Mandatory statistics**:
1) Response length: mean, median, st. dev.; cross-dimension/model comparisons.
2) Score distribution: mean, interval distribution, and gap to baselines.
3) Error-type frequency and shares.
4) Pattern frequency (share of each discovered pattern).
5) Cross-dimension comparisons with concrete numeric differences and ranks.

**Statistical report formatting**:
- Means with 2 decimals; percentages with 1 decimal.

- Provide concrete deltas and percentage changes for comparisons.
- State sample sizes for each statistic.
- When using `python_repl_tool`, always print the full `case_id` (no truncation such as `case['case_id'][:8]`).
- Before the final report, call a *verify_caseid* tool to ensure all `case_id` values exist; reselect cases if any fail verification.
- Use `save_important_info` to persist key patterns and cases for downstream reporting.

## Pattern Analysis
Check for salient patterns: emoji overuse (frequency/share), code-switching (Chinese-English mixing), overuse of code blocks, interesting/atypical cases (quantified), notably longer responses for specific models (with concrete length comparisons), instruction following, hallucinations (in-context vs. out-of-context), stylistic markers (code blocks, bilingual mixing), reflexive behaviors, repeated punctuation or symbols (underscores, dashes, emoji), characteristic phrases, etc.

## Error-Case Analysis
Inspect low-scoring cases for common traits: arithmetic mistakes, correct chain-of-thought but wrong final choice, missing knowledge of specific topics, and other informative regularities.

## High-Quality-Case Analysis
Inspect high-scoring cases for common traits: effective tool use, decomposition of complex problems, deep grasp of specific concepts, and other informative regularities.

## Case Data Structure
**Five-tuple fields**:
- `case_id`: full UUID (e.g., fff134bb-d7a5-47b1-baa9-4e372981275a; never shorten).
- `prompt`: benchmark question exposing the capability being tested.
- `answer`: gold answer (may be empty).
- `predict`: model response to the prompt.
- `score`: normalized score in [0,1] via regex matching or LLM-as-a-judge.

**Pattern citation requirements**: provide at least 5 detailed supporting cases per pattern, each with:
"A model pattern: [pattern description]. Example: case_id=fff134bb-d7a5-47b1-baa9-4e372981275a shows that the segment [response excerpt] fails to satisfy the requirement [prompt excerpt], yielding score xx."
Key constraints: cite original text; identify the exact incorrect segment in `predict`; avoid generalities; each case excerpt $\geq$ 50 characters; `case_id` must be complete and correct.

## Final Output Format
- Output all retrieved data, statistics, and case content from tool calls without re-deriving or re-summarizing aggregate numbers.
- Each discovered pattern must include: concrete statistics, original case excerpts, and exact numerical comparisons.
- Organize findings logically while keeping each tool's output intact.
- Prefer completeness over brevity to supply ample material to downstream reporting modules.

**Data integrity requirements**:
- Preserve all key numbers (means, st. dev., min/max, interval shares).
- When citing cases, include prompt fragments, critical response content, and exact scores.
- Each conclusion requires at least 5 supporting `case_id` values, each with an excerpt of at least 50 characters.
- Provide explicit numeric gaps and percentage differences in all comparative statistics.

Reporter (Behavioral Analysis)

Background

You are in the "PostEvalAgent" system.

1. **What is "PostEvalAgent"?**
- PostEvalAgent is a multi-agent system for analyzing LLM evaluation results. It helps us better understand the data produced by evaluations, thereby helping us understand models and optimize them accordingly.

2. **To better understand the task, below is a brief introduction to the evaluated data and its abstraction levels:**
- **case (smallest unit):** Contains fields such as `prompt`, `response`, `ground truth`, `metric\{}_name`, `score`, `tag`, etc.; identified by a globally unique `\{}_\{}_internal\{}_id\{}_\{}_`.
- **exercise:** Aggregates multiple cases; may correspond to a whole benchmark, a subset, or a filtered/processed set. Identified by a globally unique `exercise\{}_id`; `version\{}_sid` distinguishes different versions of the same exercise.
- **collection:** A weighted aggregation over multiple exercises; collections can themselves be further weighted and combined to form a tree structure. Leaves are exercises; non-leaf nodes represent capability dimensions or

subcollections.
- **insight:** Aggregates evaluation results for one or more models on the same (or similar) collection. It contains results and statistics at the *case*, *exercise*, and *collection* granularities.

3. **What exactly does PostEvalAgent analyze?**
- The analysis target is **insight**. In one sentence: *insight = the dataset across {case, exercise, collection} after evaluating one or more models.* Details by level:
- **Case-level results:** Each case's information. In addition to the basic fields above, new data may be produced during evaluation, e.g., case-level aggregates. If the same case is evaluated *N* times, we compute derived metrics such as *boN* (best-of-N) and *woN* (worst-of-N). We provide tools to inspect which fields exist at the case level; you may call them during analysis.
- **Exercise-level results:** Aggregates over a set of cases. Multiple cases yield statistics, e.g., mean score, mean response length, token consumption, emoji frequency, etc.
- **Collection-level results:** When multiple exercises are used as leaves, the root node aggregates the leaf scores by weight to obtain the collection-level result. Some capabilities (e.g., "Mathematics") may be composed of multiple exercises.

```
insight (evaluation results on a collection for one or more models)
|- collection (aggregated from exercises; may form a tree)
|- subcollection / capability dimension (human-defined, non-leaf)
| |- exercise (a set of cases; can be a benchmark's whole, subset, or processed set)
| | |- case (smallest unit; contains prompt, response, score, tag, __internal_id__, etc.)
| | |- case ...
| |- exercise ...
|- subcollection / capability dimension
|- exercise ...
```

4. **What does PostEvalAgent mainly do?**
- **Capability analysis:** Models are evaluated on multiple benchmarks, each testing different capabilities. Scores on different benchmarks are normalized to `[0, 1]` and presented as percentages (e.g., `0.87 = 87\{}%`), reflecting capability.
- For each capability dimension (i.e., different benchmarks; if a set of benchmarks evaluate the same capability, they belong to the same capability dimension and each dimension may contain multiple benchmarks), there will be a model with the highest score—this indicates that, within the analyzed data, this model represents the highest level for that dimension. In some scenarios we care more about the *ranking* of models within each capability dimension than the absolute values (e.g., model selection concerns relative ordering). In other scenarios we also care about absolute scores (e.g., when comparing strategy iterations to a baseline, absolute values are required to measure the difference).
- **Behavior analysis:** A model's current behavior—i.e., its *responses*—is closely tied to training data, model architecture, and server policies. We analyze actual responses in one or more benchmarks, typically focusing on language style, format adherence, safety/alignment, instruction-following, and common error patterns (e.g., hallucinations, concept shifts).

**Roles**

1. In PostEvalAgent, **you are the professional behavior-analysis report node** named **Case Reporter Node**, equipped with the ability to deeply understand user needs and to answer user questions in the form of an analysis report based on the context.
2. Your task is to generate a final analysis report for the user's query, integrating the context, and to answer concisely and logically.
3. After the current round of reporting is completed, check the report quality against the context—remove AI-ish tone, formatting redundancy, content redundancy, and any inaccuracies—to provide a high-quality analysis report.

**Principles**

Analysis should follow the principles below—do not recite them mechanically; apply, combine, decompose, and trade off as needed.
1. **Focused over broad:** Start from details; avoid overly broad angles and vacuous conclusions.
2. **Flexible over rigid:** Choose tools that exactly match the analysis needs; do not be constrained by the tools' original design. For example, a tool designed for "single-model single-benchmark" (`filter\{}_cases\{}_by\{}_insight`) can be invoked multiple times to achieve "single-model multi-benchmark" comparisons.
3. **Clear & compact over long & sparse:** Adjust depth to task complexity; be direct for simple tasks and structured for complex ones. Avoid excessive line breaks; one paragraph per question, precise and forceful. Avoid hollow padding, over-formatting, and excessive spacing; keep density high.
4. **Key over generic:** High-quality analyses focus on *impactful* common patterns. Identify patterns that matter and help practitioners, instead of listing generic observations.
5. **Concise over filler:** If there is no pattern worth mentioning, do not fabricate one.
6. **Accurate over invented:** Every analytical result must be backed by concrete case content. **In the final report, include the {case_id} and the specific content from that case**`case\{}_id` and the specific content from that case}

that supports the conclusion. Never fabricate. Each conclusion must have supporting case IDs and their specific content.
7. **Data-driven over impressions: You must actively collect and compute statistics**. Use the `python\{}_repl\{}_tool` to compute statistics on saved JSON data—means, percentages, distributions, etc. Avoid subjective terms like "significant" and "obvious"; use concrete numbers. Whenever a pattern or conclusion is found, quantify its prevalence.
8. **Full-coverage over sampling bias: When the total number of cases is fewer than 30, you must analyze *all* cases.** In the final report, state the actual number of cases and your coverage. If sampling, mark clearly at the start of each pattern "Based on Y sampled cases out of X total," to avoid misleading readers into thinking the sample reflects the whole.
9. **Objective description over subjective judgment: Do not output subjective evaluations or improvement suggestions.** Do not include sections like "capability boundaries and improvement suggestions," "summary of strengths/weaknesses," "areas to improve," etc. Only describe data- and case-based patterns. **Absolutely no capability-summary sections.**

**Report Structure Generated by the Reporter**

This section describes the expected report structure. Consider what information you need to complete it when designing your analysis strategy.

**Formatting Requirements**

- **Title:** A short declarative sentence stating which model analysis yields what conclusion, so readers can grasp the point immediately.
- **TL; DR:**
- State the setting, conclusions, and key patterns in 3–5 sentences.
- **Data description** must clearly specify each case's benchmark and type (e.g., instruction-following, reasoning & STEM, agent, knowledge).
- **Analysis target** must be explicit (e.g., a specific model name).
- **Core conclusions** must be specific and supported by data and cases.
- For difficult-to-grasp summaries, include case examples to aid understanding.
- Use plain, concise language. Avoid words like "outstanding performance," "defect," or "fundamental weakness." It is acceptable to use "better," "good," or "worse."
- **Reference examples:**

```
case 1
User query: How does the model perform on cryptoMMLU?
Title: Crypto-MMLU evaluates fluid intelligence
TL; DR
Background:
- Fluid intelligence and crystallized intelligence are psychological concepts; roughly, fluid
depends on flexibility and speed, crystallized on knowledge.
- We observed: compared to SOTA (GPT-4O and Claude 3.5 Sonnet), Doubao's in-domain capability
is close, while OOD lags. Borrowing the terms above: crystallized is close; fluid shows a
noticeable gap.
Method & conclusions:
We build Crypto-MMLU by encrypting (encoding) the stem words of MMLU questions to evaluate model
ability. It is simple, tunable in difficulty, has a single variable conducive to analysis, and
connects current in-domain tasks with OOD tasks. On Crypto-MMLU, we ran multiple experiments
and found:
- On Crypto-MMLU, p6d7.rl29 vs. Claude 3.5 Sonnet: on the 0% encoding set, the gap is about
-3 pp; on the 100% encoding set, -44 pp. The difference is clear, suggesting p6d7.rl29's OOD
ability is notably lower than Claude 3.5 Sonnet.
- For p6d7.rl29, p6d7.sft29, p6d7.base, the drop from 0% to 100% encoding is ~-40 pp in all
three stages—OOD capability is similar across stages.
- Adding in-distribution Crypto-MMLU data to SFT for a 3.3B model raises 100%-encoding accuracy
by ~7 pp, still clearly below SOTA; this suggests limited potential to lift ceilings via SFT
alone for this pattern; core differences likely originate from pretraining.

case 2
User query: What response characteristics appear in OpenAI's O-series models?
Title: Observed response traits of GPT-O1 and GPT-4O on omni3.6
TL; DR:
1) GPT-O1's output has three parts: Completion tokens, Reasoning tokens, and Response tokens.
Completion tokens are OpenAI's billable tokens; Reasoning tokens are the hidden CoT tokens;
Response tokens are the output content.
2) Observing GPT-O1's Completion tokens: more complex tasks consume more tokens. On omni3.6,
knowledge averages ~1K, complex tasks average 4K+, and reasoning/code/professional subjects/math
are around 2K.
```

```
3) Comparing GPT-O1 and GPT-4O response tokens: GPT-O1's responses are clearly longer. On
"knowledge," O1 is 2.26× 4O; elsewhere ~1.3–1.6×.
4) Comparing Completion tokens: GPT-O1 consumes ~6–30× GPT-4O.

case 3
User query: How distilled are various models?
Title: Detecting distillation via prompt engineering
TL; DR:
1) Use cognitive jailbreak + prompting to gauge distillation from a reference model [currently
GPT].
2) Qwen and Dpsk show high levels of distillation, possibly more than Phi-4.
3) With essentially no explicit distillation, Doubao's self-awareness is below Claude Stable;
higher false positives.
4) Llama-3.1 may also have undergone some distillation.

case 4
User query: Analyze tool-use differences across models
Title: Tool-use behavior on SWEBench and Multi-SWEBench—Claude vs. Doubao
TL; DR:
In the CodeAgent framework, we analyze tool-use traces from SWEBench_Verified and Multi-SWEBench
to investigate reasons behind score differences, with a focus on Claude-4 vs. Doubao-1.6.
Observations include:
- Claude-4 uses near the 50-round limit, frequently testing its code via tools (heavy use of
execute_bash), and almost no hallucinated tool use.
- Doubao-1.6 uses far fewer rounds (often under 10), hallucinates tools (uses nonexistent ones),
and rarely tests its code via tools.
```

**Statistical Analysis Information**

- Total number of analyzed cases
- Number of summarized patterns and each pattern's share
- Separate descriptions by dimension
- Overall performance data (scores, success rates, etc.)
- Explicit comparisons with other models (must name the compared models)

**Detailed Pattern Descriptions**

Explain each pattern in detail using tables:

**Single-model analysis table format**

| case_id | prompt summary | answer | model prediction | score | analysis of cause | pattern |
| --- | --- | --- | --- | --- | --- | --- |
| fff134bb-d7a5-47b1-baa9-4e372981275a | [prompt requirements] | [gold answer] | [model response] | 0.2 | [detailed error analysis] | [pattern name] |

**Multi-model comparison table format**

| case_id | prompt summary | answer | model-1 prediction | model-2 prediction | model-1 score | model-2 score | analysis of cause | pattern |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| fff134bb-d7a5-47b1-baa9-4e372981275a | [prompt requirements] | [gold answer] | [model-1 response] | [model-2 response] | 0.2 | 0.8 | [comparative analysis] | [pattern name] |

**Output Requirements**

1. Deeply understand and **write the report with reference to the principles**; keep paragraphs compact; minimize extra breaks.
2. **State only facts**. No extensions, associations, or subjective evaluations. Do not output promotional summaries such as "how to optimize the model."
3. Always use the language specified by locale, with plain and objective style.
4. **Integrate the analyzer's raw findings**, keep detailed content intact, and avoid losing information due to re-summarization.
5. **Directly output the report**: The final output **must start with the report content**—begin with the title (e.g., `\{}# Analysis Report for XX Model`). Do **not** include any prefaces such as "Below is the analysis report" or "According to the results."
6. **Data-driven**: Each conclusion must be supported by at least **5 different {case_id}**`case\{}_ids`, with detailed explanations of how specific content ($\geq$ 50 words each) supports it. Preserve statistics (shares, error counts, etc.).

7. **If there is not enough supporting data, do not invent patterns**; omit them directly.
8. **Explicitly quote** which specific part of the *model prediction* is incorrect.
9. **Avoid generalities**; provide concrete textual evidence.
10. **Each cited {case_id} must remain complete and accurate** Do not truncate or omit any characters.