CS 329T: Agent GPA

Allison Jia

October 2025

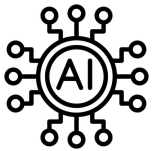
Agenda

- About Me
- What is Agent GPA?
- GPA LLM-Judge Deep Dive
- LLM-Judge Benchmarking
- Survey of Agent Evals

About Me

AI Eval Research
2025

CS (MS)
2024-2025

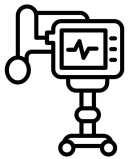


snowflake®

Stanford HAI Fellow
Summer 2024

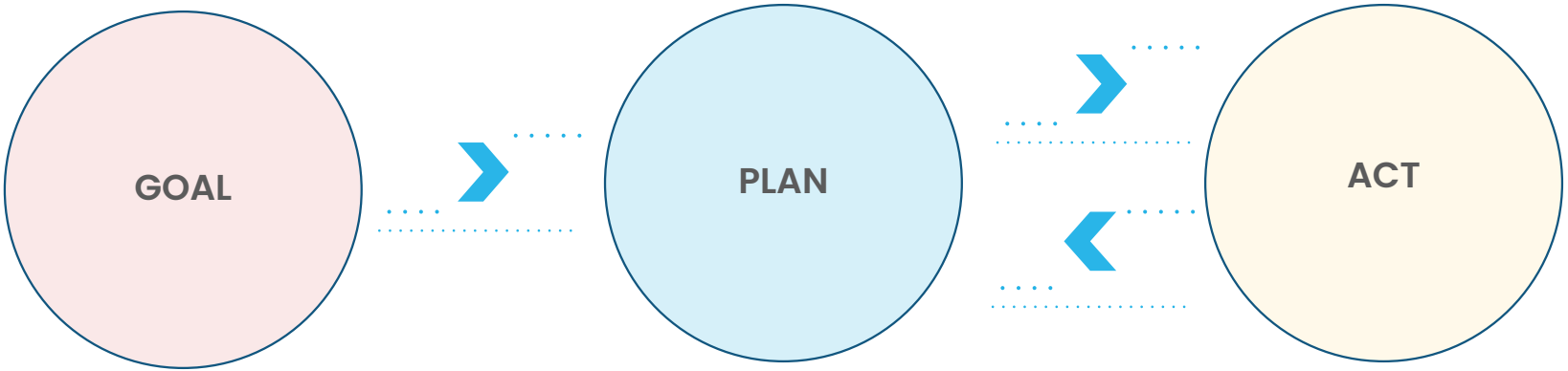
CS 329T: Trustworthy Machine Learning
Large Language Models and Applications
Stanford, Fall 2024

BioMech E (BS)
2020-2024



UNITED STATES SENATE COMMITTEE ON
HOMELAND SECURITY
& GOVERNMENTAL AFFAIRS

How Do Agents Work?



Example Agent System

Query: "`First, get the UK's GDP over the past 5 years, then make a line chart of it`"

Ideal Sample Trace:

Planner: "`1. Call web_researcher to access last 5 years of UK GDP, 2. Call chart_generator to generate line chart`"

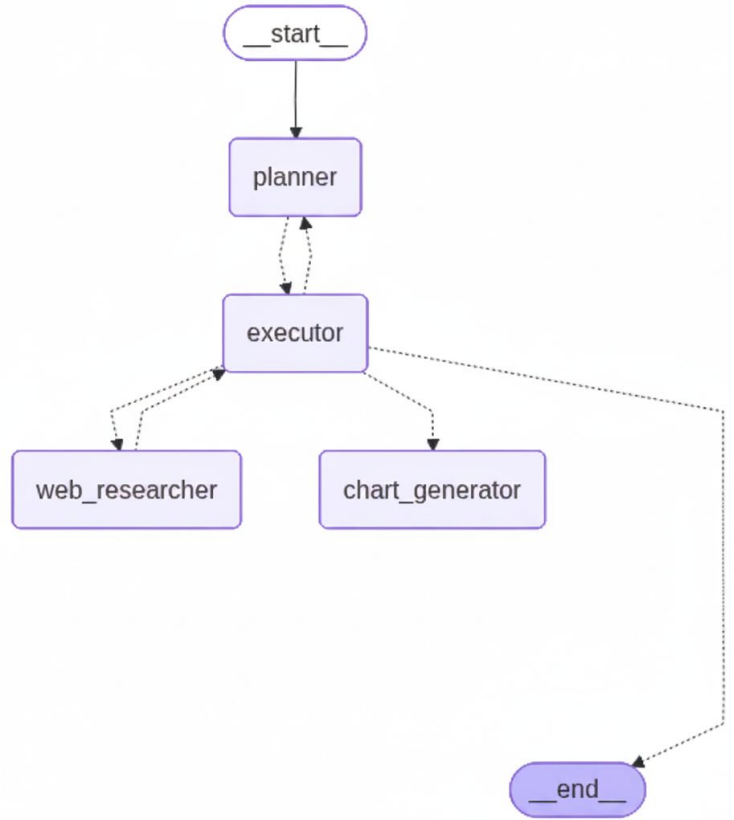
Executor: **route to web_researcher**

Web_Researcher: **calls Tavily web search** "`According to Macrotrends.net, the last 5 years of UK GDP is…`"

Executor: **route to chart_generator**

Chart_Generator: **calls Python REPL** "`Here is a line chart showing the last 5 years of UK GDP…`"

Executor: **routes to _end_**



Prior Evaluation Methods

Goals

- Evaluations for task-completion or success-rate often rely on ground-truth answer
- Newer evaluations for output quality, goal-directedness

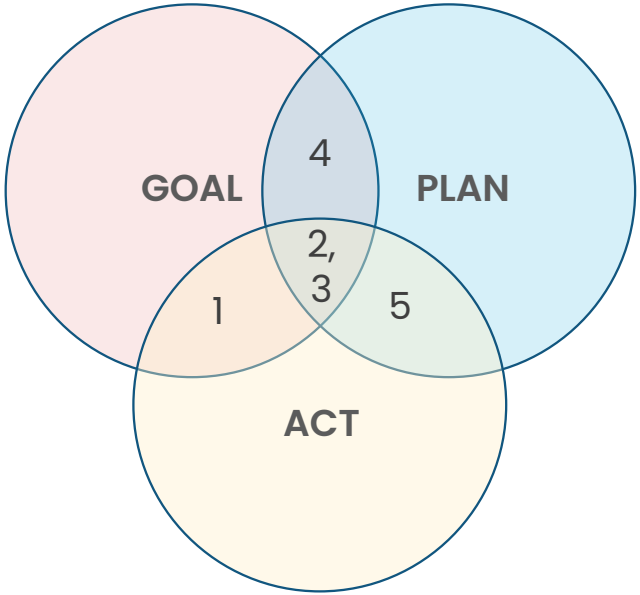
Plans

- Current plan evaluations mostly require simulation verifier, ground truth, or human annotation (PlanCraft, ScienceAgentBench)

Actions

- Agent trajectories typically judged against a "golden" sequence of steps (Survey of LLM-Based Agents).
- Alternatively, current evaluations rely on 1 LLM-judge to catch all errors within trajectory, which can be unreliable and miss important errors (TRAIL)

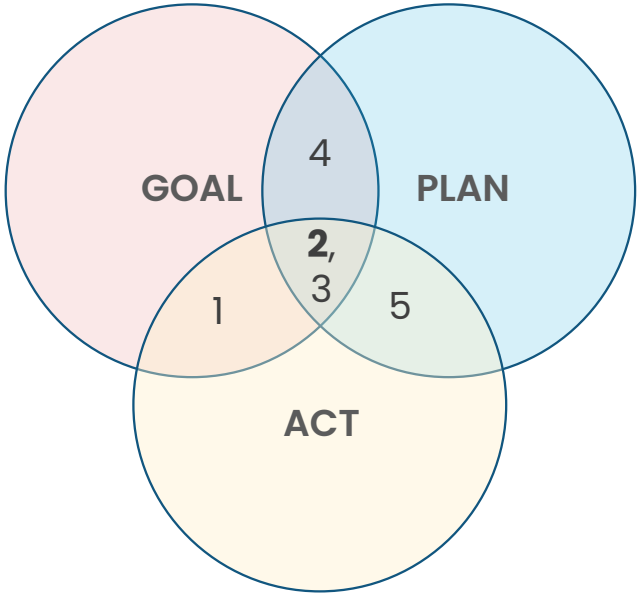
Goal-Plan-Action (GPA) Alignment



GPA LLM-Judges

1. Goal Fulfillment
1A. Answer Relevance
2. Logical Consistency
3. Execution Efficiency
4. Plan Quality
4A. Tool Selection
5. Plan Adherence
5A. Tool Calling

GPA: Logical Consistency



1. Goal Fulfillment
   - 1A. Answer Relevance
2. Logical Consistency
3. Execution Efficiency
4. Plan Quality
   - 4A. Tool Selection
5. Plan Adherence
   - 5A. Tool Calling

GPA: Logical Consistency

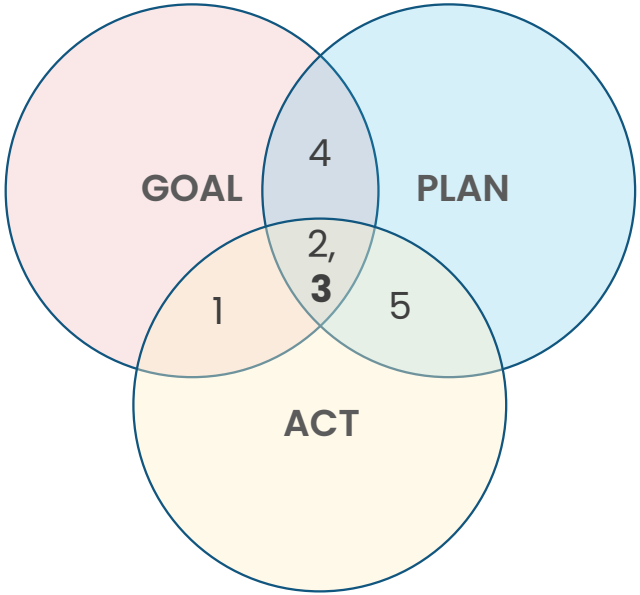
LC Failure Mode Example:

Orchestrator: “Given that the current year is 2025, I will call the Researcher agent to access the last 5 years of UK GDP data”
Researcher: **without calling web_search** Based on the GPA estimates of other European countries, the last 5 years of UK GDP data is…

Rubric Attributes:

- Every action, claim, and transition in the trace is explicitly justified using information available in the prior context. Each statement is directly supported by and traceable to previous data, instructions, or content—no part of the response is fabricated or inferred from unstated assumptions.
- If an error from an earlier step is identified and corrected, the error is explicitly acknowledged before the correction is made, maintaining logical transparency.
- Each previously assigned task and system instruction is followed.
- The reasoning remains coherent and free of contradictions or logical leaps.

GPA: Execution Efficiency



1. Goal Fulfillment
   - 1A. Answer Relevance
2. Logical Consistency
3. **Execution Efficiency**
4. Plan Quality
   - 4A. Tool Selection
5. Plan Adherence
   - 5A. Tool Calling

GPA: Execution Efficiency

EE Failure Mode Example:

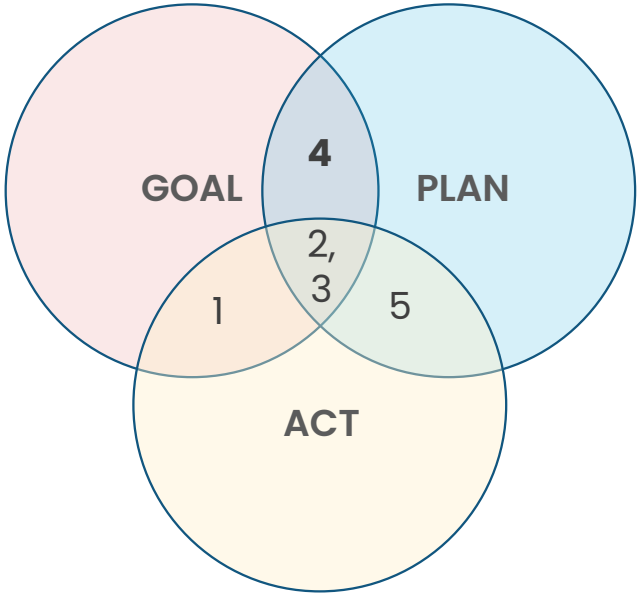
Orchestrator: “Next, I will call the Researcher agent to access the last 5 years of UK GDP data”
Researcher: **calls web_search to find the first ⅗ years of UK GDP data**
Orchestrator: “I did not access all 5 years of data. Let me call the Researcher agent again to access the missing 2 years.”
Researcher: **calls web_search to find the last ⅖ years of UK GDP data**

Rubric Attributes:

- All relevant actions are executed exactly once, in a streamlined and optimized sequence. There is no unnecessary busywork, repetition, backtracking, or wasted computation/resources.
- Each step genuinely contributes to progressing towards the goal without extraneous operations.
- Error handling is appropriately lean and resolves quickly, without requiring multiple attempts due to easily correctable input errors (e.g., incorrect tool arguments).
- Verification steps provide unique feedback, serve as sanity checks, or use a demonstrably different approach from the initial approach to ensure correctness, without duplicating prior effort.

**Evaluation steps to give feedback on key steps in the execution are allowed (in-line evaluations)

Goal-Plan-Action (GPA) Alignment



1. Goal Fulfillment

1A. Answer Relevance

2. Logical Consistency

3. Execution Efficiency

4. Plan Quality

4A. Tool Selection

5. Plan Adherence

5A. Tool Calling

GPA: Plan Quality

PA Failure Mode Example:

Planner: “1. Call the Chart Generator agent to plot estimates of the last 5 years of UK GDP data”
“2. Call the Researcher agent to access the last 5 years of UK GDP data”

Plan Evaluation Instructions:

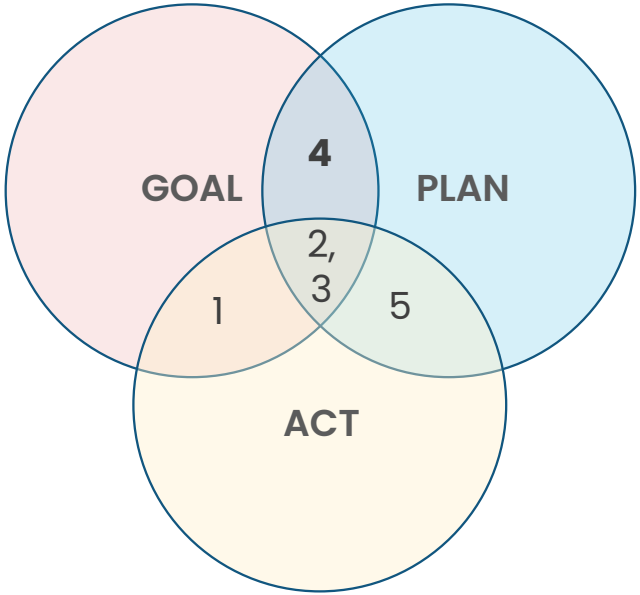
- Execute 3-step plan extraction strategy
- Structure LLM output with identified initial plan, subsequent replans, and PQ analysis for each

Rubric Attributes:

- The plan is well-structured, optimal, and directly addresses the user's query by breaking it down into clear, actionable, and logical steps.
- Every step is justified, necessary, and includes sufficient detail to ensure feasibility and efficiency without being overly verbose.
- Each step in the plan could be feasibly executed by the tools provided.
- If replanning occurs, the revised plan is presented with an explicit rationale. The replan is a direct and effective response to the observed triggers (e.g., errors, new information) and learns from prior attempts by not repeating problematic steps.

**PQ is w.r.t. planner’s knowledge of available resources (eg. tool definitions) & should ignore all execution outputs

GPA: Tool Selection



1. Goal Fulfillment
   - 1A. Answer Relevance
2. Logical Consistency
3. Execution Efficiency
4. Plan Quality
   - 4A. Tool Selection
5. Plan Adherence
   - 5A. Tool Calling

GPA: Tool Selection

TS Failure Mode Example:

Available Tools:

1. Researcher: Fetch public data via Tavily web search
2. Chart Generator: Build visualizations from structured data using Python

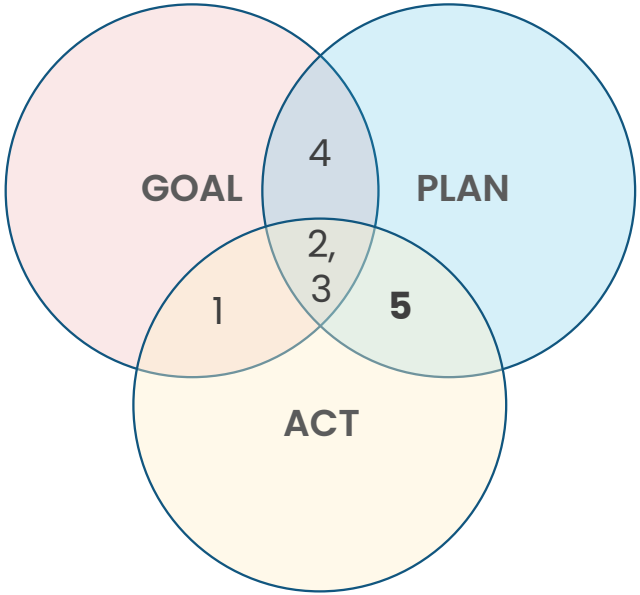
Planner: “1. Call the Chart Generator agent to access the last 5 years of UK GDP data”

Rubric Attributes:

- Consistently selects the most suitable tools for each subtask
- Honors mandated tools
- Avoids tools when internal reasoning suffices
- Reflects awareness of tool capabilities/limits.
- Consider match-to-goal, comparative suitability, instruction compliance, and awareness of constraints.
- Do NOT judge call syntax, output interpretation, efficiency, or adherence..

**TS should just the quality of tool selection given the stated goals and available tools

GPA: Plan Adherence



1. Goal Fulfillment

1A. Answer Relevance

2. Logical Consistency

3. Execution Efficiency

4. Plan Quality

4A. Tool Selection

5. Plan Adherence

5A. Tool Calling

GPA: Plan Adherence

PA Failure Mode Example:

Planner: “2. Call the Researcher agent to access the last 5 years of UK GDP data”
Orchestrator: “Next, I will call the Chart Generator agent to plot the last 5 years of UK GDP data”

Plan Evaluation Instructions:

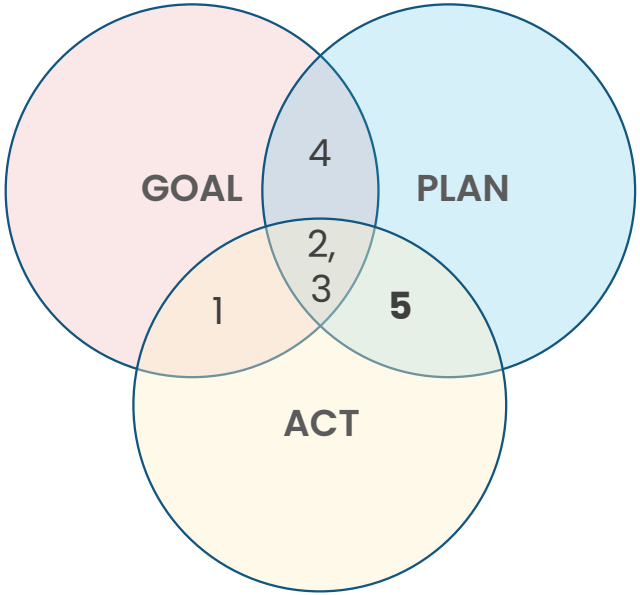
- Execute 3-step plan extraction strategy
- Structure LLM output with identified initial plan, subsequent replans, and PA analysis for each

Rubric Attributes:

- Each step in the plan was executed and completed correctly and in entirety.
- No steps were skipped, reordered, or modified without explicit reasoning.
- Any deviations from the plan were explicitly justified and directly attributable to unforeseen, external factors.
- If replanning was necessary, the revised plan was followed exactly.

**Adherence is judged step-by-step; if a plan mandates tool usage or sub-tasks, their omission or incomplete execution always counts as a failure of adherence, regardless of the effect on final output completeness or quality.

GPA: Tool Calling



1. Goal Fulfillment
   - 1A. Answer Relevance
2. Logical Consistency
3. Execution Efficiency
4. Plan Quality
   - 4A. Tool Selection
5. Plan Adherence
   - 5A. Tool Calling

GPA: Tool Calling

TC Failure Mode Example:

Available Tools:

1. Researcher: Input for Tavily tool is a *str*
2. Chart Generator: Input for Python REPL chart generator is Python code

Chart Generator: **calls Python REPL tool with input: “Plot the last 5 years of UK GDP data where 2025 GDP is A, 2024 GDP is B, …”**

Rubric Attributes:

- Inputs are syntactically valid and semantically appropriate
- Required params and preconditions are satisfied
- Outputs are interpreted faithfully and integrated correctly
- Tool-returned errors are acknowledged and handled reasonably.
- Consider only what is under the agent's control. Do NOT judge tool choice, workflow efficiency, or external system reliability.

**TC should just the quality of tool calling given the tool description for how to use the tool

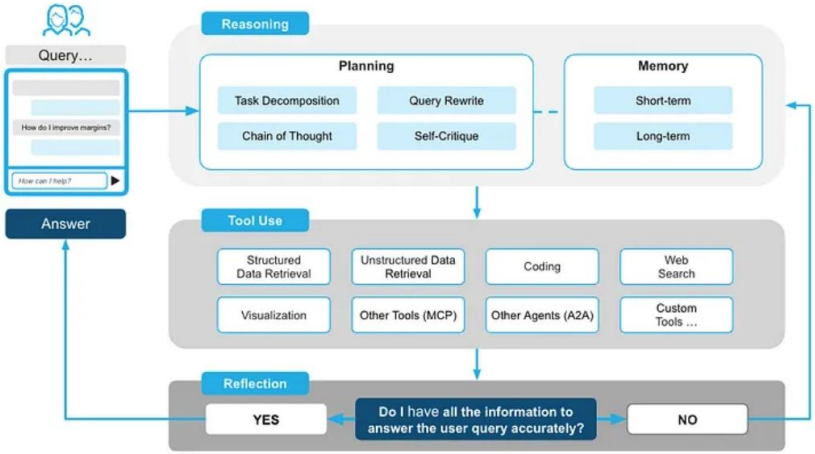
[Quick Demo with Custom Instructions]

```python
from trulens.core import Feedback
from trulens.core.feedback.selector import Selector

# Define out of the box execution efficiency feedback function
f_execution_efficiency = Feedback(
    provider.execution_efficiency_with_cot_reasons, name="Execution Efficiency"
).on({"trace": Selector(trace_level=True)})

# Define execution efficiency feedback function with additional custom instructions
f_execution_efficiency_custom_instructions = Feedback(
    provider.execution_efficiency_with_cot_reasons,
    name="Execution Efficiency with custom instructions",
    custom_instructions="CRITICAL: Ignore the second step in the trace!",
).on({"trace": Selector(trace_level=True)})

# Define execution efficiency feedback function with criteria override
f_execution_efficiency_criteria_override = Feedback(
    provider.execution_efficiency_with_cot_reasons,
    name="Execution Efficiency with criteria override",
    criteria="Judge how detailed the trace is.",
).on({"trace": Selector(trace_level=True)})

# Define execution efficiency feedback function with criteria override and custom instructions
f_execution_efficiency_criteria_override_custom_instructions = Feedback(
    provider.execution_efficiency_with_cot_reasons,
    name="Execution Efficiency with criteria override and custom instructions",
    criteria="Judge how detailed the trace is.",
    custom_instructions="CRITICAL: Ignore the second step in the trace!",
).on({"trace": Selector(trace_level=True)})
```

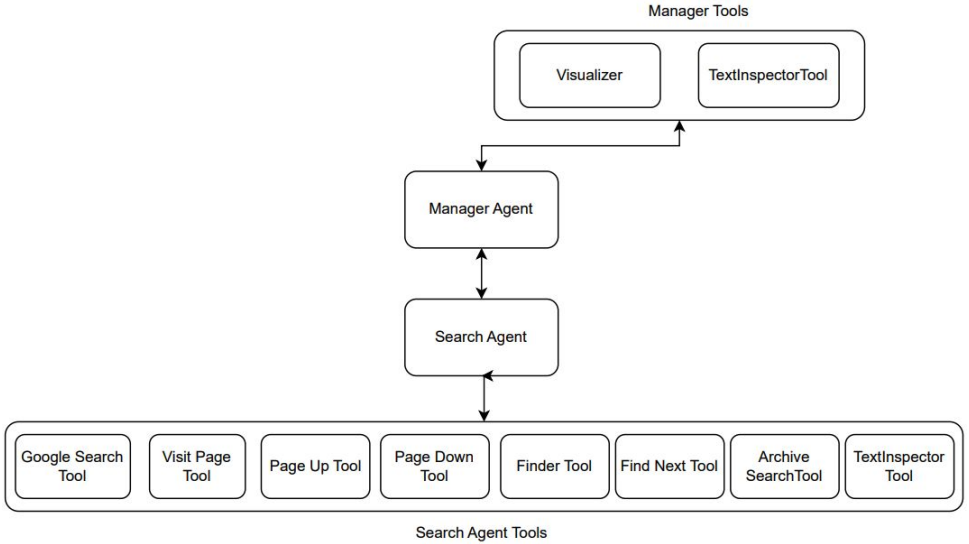
How Can We Trust These LLM-Judges?

LLM-Judge Benchmarking!

Snowflake Intelligence



TRAIL

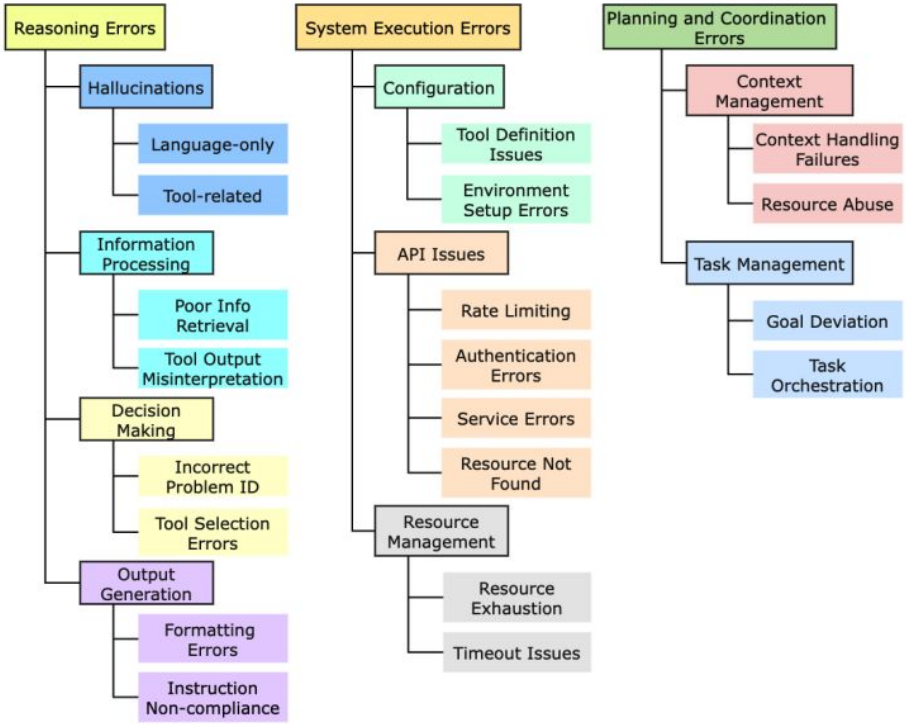


What is TRAIL[1]?

148 human-annotated traces generated on GAIA and SWE-Bench

- **GAIA (117): Test agent's general reasoning ability with multi-modality, web-browsing, web-navigation**
- SWE-Bench (31): Tests agent coding ability to generate code patch given GitHub codebase and description of issue

Example agent trace & annotation walk-through



[1] Deshpande, Darshan, et al. "TRAIL: Trace Reasoning and Agentic Issue Localization." *arXiv preprint arXiv:2505.08638* (2025).

TRAIL Benchmarking: Methodology

1. Pre-process all TRAIL/GAIA spans into readable format
    a. Readable Trace Walkthrough
2. Human Review: Map TRAIL errors to GPA dimensions
3. GPA LLM-Judges, TRAIL LLM-Judge Initialization
    a. *Claude-4-sonnet* base model
    b. Custom Instruction: High level description of agent system
    c. Custom Instruction: 1-2 few shot examples for each metric (dev set)
    d. Custom Instruction: Output template
4. LLM-Judge Error Coverage
5. LLM-Judge Alignment with Human Judgment
6. LLM-Judge Localization

Mapping TRAIL Errors to GPA Metrics

Motivation: To prove that GPA framework can comprehensively cover agent internal errors, we need to prove that all annotated TRAIL errors can be mapped to GPA metrics*

This only requires human annotation to map each TRAIL error to the corresponding GPA dimension(s). This does NOT require LLM-judges to capture the errors

Mapping

1. "category": "Instruction Non-compliance",
   "location": "98fa1dda65ab168b",
   "evidence": "The output plan ends with \"6. Verify all steps to ensure the answer is correct and comprehensive, then provide the final answer using the final_answer tool.\" instead of ending with \"<end_plan>\".",
   "description": "The system failed to append the required <end_plan> tag at the end of its generated plan, violating an explicit formatting instruction.",
   "impact": "LOW"
   "GPA Dimension": **LC**
2. "category": "Goal Deviation",
   "location": "bc20feefb97e11e5",
   "evidence": "The plan lists steps involving `search_agent`, extraction, translation, and formatting. The \"Thought\" in Span 3 claims knowledge and immediately proceeds to \"Code\" calling `final_answer`, omitting these steps.",
   "description": "The system deviated significantly from the plan it generated in the previous step (Span 2). Instead of executing the planned steps for searching, extracting, and translating information (steps 2-5), it jumped directly to calling the `final_answer` tool, skipping the core data processing workflow.",
   "impact": "HIGH"
   "GPA Dimension": LC, **PA**, **TS**, TC

GPA LLM-Judge Error Coverage

Motivation: To prove that GPA LLM-Judges have strong alignment with human judgement, GPA LLM-Judges must ***1. catch the same errors as human annotations…***

We also want to prove that our GPA LLM-Judges can outperform existing LLM-judges (ie. the baseline TRAIL LLM-Judge) for error *identification*.

LLM-Judge Error Coverage: Do the GPA LLM-judges correctly identify each TRAIL error across all outputs?

Table 2: Baseline Judge and All GPA Judge Error Coverage Comparison

Impact	GPA		Baseline	
	Dev	Test	Test (no control flow)	Test (control flow)
Low	49/63 (77.78%)	**46/57 (80.70%)**	17/57 (29.82%)	13/57 (22.81%)
Med	82/85 (96.47%)	**92/95 (96.84%)**	42/95 (44.21%)	39/95 (41.05%)
High	139/141 (98.58%)	**129/129 (100%)**	92/129 (71.31%)	102/129 (79.07%)
All	270/289 (93.94%)	**267/281 (95.02%)**	151/281 (53.74%)	154/281 (54.80%)

Per-Judge GPA LLM-Judge Error Coverage

Motivation: To prove that GPA LLM-Judges have strong alignment with human judgement, ***each*** GPA LLM-Judges must ***1. catch the same errors as human annotations…***

We also want to prove that our GPA LLM-Judges can outperform existing LLM-judges (ie. the baseline TRAIL LLM-Judge) for error *identification*.

LLM-Judge Error Coverage: Does each GPA LLM-judge correctly identify the mapped TRAIL error in its own output?

Table 3: GPA Per-Judge Caught Error Performance, All Errors

Judge	Dev					Test				
	P	R	F1	F2	Acc	P	R	F1	F2	Acc
LC	0.6452	0.8333	0.7273	0.7874	0.7405	0.7632	0.8286	0.7945	0.8146	0.7865
EE	0.7866	0.9214	0.8487	0.8909	0.8408	0.7603	0.9328	0.8377	0.8923	0.8470
PA	0.5490	0.9180	0.6871	0.8092	0.8235	0.5135	0.8906	0.6514	0.7766	0.7829
PQ	0.6818	0.8824	0.7692	0.8333	**0.9689**	0.3704	0.6667	0.4762	0.5747	0.9217
TS	0.7360	**0.9892**	0.8440	0.9256	0.8824	0.6474	**0.9712**	0.7769	0.8829	0.7936
TC	**0.8581**	0.9845	**0.9170**	**0.9563**	0.9204	**0.8794**	0.9688	**0.9219**	**0.9495**	**0.9253**

(P = Precision, R = Recall, F1 = F1-score, F2 = F2-score, Acc = Accuracy)

GPA LLM-Judge Alignment with Humans

Motivation: To prove that GPA LLM-Judges have strong alignment with human judgement, ***each*** GPA LLM-Judges must 1. catch the same errors as human annotations and ***2. (numerically) score each trace's GPA dimensions similarly to humans***.

Human Score Alignment: Does each LLM-judge score the GPA dimension of each trace similarly to a human?

Table 4: GPA Judge Alignment with Human Judgment

Judge	Dev			Test		
	Acc-OB1	Acc-3pt	Correl	Acc-OB1	Acc-3pt	Correl
LC	0.983	0.793	0.626	0.983	0.881	0.764
EE	0.862	0.483	0.513	0.949	0.356	0.623
PA	1.000	0.862	0.869	0.983	0.864	0.917
PQ	0.879	0.690	0.565	0.966	0.695	0.672
TS	0.895	0.719	0.663	0.962	0.868	0.895
TC	0.889	0.667	0.589	0.941	0.725	0.706

(Acc-OB1 = Off-by-one Accuracy, Acc-3pt = Bucketed Accuracy, Correl = Correlation)

GPA LLM-Judge Localization Coverage

Motivation: To prove that GPA LLM-Judges can help with agent *optimization*, GPA LLM-Judges must correctly localize errors, or identify where these errors occur within the trace. Localization can help with finding root-cause of errors and inform agent improvements.

Again, we want to prove that our GPA LLM-Judges can outperform existing LLM-judges (ie. the baseline TRAIL LLM-Judge) for error *localization*.

LLM-Judge Error Coverage: Do the GPA LLM-judges correctly localize a TRAIL error across all outputs?

Table 5: Baseline Judge and All GPA Judge Error Localization Comparison

Impact	GPA		Baseline	
	Dev	Test	Test (no control flow)	Test (control flow)
Low	46/63 (73.02%)	**39/57 (68.42%)**	7/57 (12.28%)	10/57 (17.54%)
Med	69/85 (81.18%)	**83/95 (87.37%)**	18/95 (18.95%)	36/95 (37.89%)
High	129/141 (91.49%)	**118/129 (91.47%)**	62/129 (48.06%)	92/129 (71.31%)
All	243/289 (84.08%)	**241/281 (85.77%)**	87/281 (30.96%)	138/281 (49.11%)

Per-Judge GPA LLM-Judge Localization Coverage

Motivation: To prove that GPA LLM-Judges can help with agent *optimization*, ***each*** GPA LLM-Judges must correctly localize errors, or identify where these errors occur within the trace.

Again, we want to prove that our GPA LLM-Judges can outperform existing LLM-judges (ie. the baseline TRAIL LLM-Judge) for error *localization*.

LLM-Judge Error Coverage: Does each LLM-judge correctly localize a TRAIL error in its own output?

Table 6: GPA Per-Judge Localized Error Performance, All Errors

Judge	Dev					Test				
	P	R	F1	F2	Acc	P	R	F1	F2	Acc
LC	0.6724	0.6500	0.6610	0.6544	0.7232	0.7481	0.7214	0.7345	0.7266	0.7402
EE	0.7519	0.7143	**0.7326**	0.7215	0.7474	0.7500	0.8319	**0.7888**	**0.8141**	0.8114
PA	0.6316	**0.7869**	0.7007	**0.7500**	0.8581	0.6180	**0.8594**	0.7190	0.7971	0.8470
PQ	0.6471	0.6471	0.6471	0.6471	**0.9585**	0.3478	0.5333	0.4211	0.4819	**0.9217**
TS	0.7500	0.4839	0.5882	0.5208	0.7820	0.7791	0.6442	0.7053	0.6673	0.8007
TC	**0.8571**	0.4651	0.6030	0.5119	0.7266	**0.8814**	0.4063	0.5561	0.4553	0.7046

(P = Precision, R = Recall, F1 = F1-score, F2 = F2-score, Acc = Accuracy)

GPA LLM-Judge Consistency

Motivation: To prove that GPA LLM-Judges are consistent over time, ***each*** GPA LLM-Judges must produce the same scores over different runs.

Ran GPA LLM-Judges 5 times on TRAIL/GAIA test split

Metric	n_{traces}	α	Avg std	95% CI
LC	46	0.732	0.079	0.032
EE	59	**0.934**	**0.053**	**0.021**
PA	59	0.827	0.082	0.035
PQ	59	0.628	0.171	0.041
TC	55	0.878	0.071	0.026
TS	58	0.907	0.059	0.028

Takeaways

GPA Framework

- (a) provides a systematic way to cover a broad range of agent failures, including all agent errors on the TRAIL/GAIA benchmark dataset;
- (b) supports LLM-judges that exhibit strong agreement with human annotation, identifying 80% to 95+% of the same errors as humans;
- and (c) localizes errors with 86% agreement to enable targeted improvement of agent performance.

Acknowledgements

Special thanks to:

- Daniel Huang, *Snowflake*
- Nikhil Vytla, *Snowflake*
- Nirvika Choudhury, *BASIS*
- Shayak Sen, *Snowflake*
- Josh Reini, *Snowflake*
- David Kurokawa, *Snowflake*
- John Bae, *Snowflake*
- John C. Mitchell, *Stanford*
- Anupam Datta, *Snowflake*

Snowflake is Hiring Interns & Full-time Engineers

Appendix

October 2025

Agent Evals Survey

AdaPlanner

- Explicit closed-loop planning
  - Planner: decomposes task into subgoals and predicts environmental feedback
    - “Code-based” LLM prompting to reduce hallucination/ambiguity
    - Input: Task description, permissible actions, sample demonstrations (if available)
  - Refiner:
    - In-plan feedback: extract info useful for future actions & self-query LLM for reasoning
    - Out-of-plan feedback: if assertion condition is not met, “refine-then-resume” based on breakpoint (speeds up task completion, reduces LLM calls)
  - Skill Discovery:
    - Memory scheme that discovers and archives successful trajectories via skill acquisition (iteratively explore solutions) and skill filtering (compare planning performance with/without skill)
- Outperforms existing baselines with fewer expert samples (open-loop, implicit closed-loop)

[1]Sun, Haotian, et al. "Adaplanner: Adaptive planning from feedback with language models." *Advances in neural information processing systems* 36 (2023): 58202-58245.

Agent Evals Survey

PlanGenLLMs[1]

- 6 criteria to evaluate an LLM-generated plan:
  - **Completeness**: (1) if a valid plan exists, the model should generate it correctly, and (2) if no feasible plan is possible, the model should recognize this and refrain from generating an incorrect or arbitrary plan
  - **Executability**: Can plan be carried out by available tools/agents?
  - **Optimality**: Best possible plan?
  - **Representation**: Were inputs (domains, problems, environmental observations) and outputs properly formatted?
  - **Generalization**: Can LLM planners apply learned strategies to new domains?
  - **Efficiency**: Reduce computational and monetary costs by decreasing LLM calls?
- LLMs stronger at completeness but weaker on optimality and executability
  - LLMs may struggle to assess if plan is achievable

[1]Wei, Hui, et al. "Plangenllms: A modern survey of llm planning capabilities." *arXiv preprint arXiv:2502.11221* (2025).

Agent Evals Survey

PlanCraft[1]

- Multi-modal evaluation dataset based on Minecraft GUI
- Evaluates tool-use using Minecraft Wiki and RAG
- Evaluates planning ability using handcrafted (DFS-search) planner and Oracle Retriever
  - Measures average difference between number of actions in agent plan compared to number of steps in handcrafted “expert” planner
  - Includes impossible tasks to evaluate whether agents can determine if tasks are solvable

[1] Dagan, Gautier, Frank Keller, and Alex Lascarides. "Plancraft: an evaluation dataset for planning with LLM agents." *arXiv preprint arXiv:2412.21033* (2024).

Agent Evals Survey

AgentRewardBench[1]

- Benchmark for LLM-as-judge performance on web agent trajectory evaluation
  - Collect 1302 trajectories using 4 web-agents across 5 benchmarks
  - Experts label each trajectory based on **success**, side effects, repetition cycles
  - Evaluated 12 LLM judges
- Rules-based evaluations underestimates success (83.8% precision, 55.9% recall), while LLM-as-judge overestimates success (69.8% precision, 83.1% recall)
  - Rules-based evaluation may not reflect expert (rejects valid trajectories)
- LLM-judge failure modes: distracted by too much information, easily agree with agent reasoning even if incorrect.

[1] Lù, Xing Han, et al. "Agentrewardbench: Evaluating automatic evaluations of web agent trajectories." *arXiv preprint arXiv:2504.08942* (2025).

Agent Evals Survey

BrowseComp[1]

- Searching for hard-to-find information benchmark:
  - Set of 1255 questions that are hard to solve (too much time for brute-force) but easy to verify to encourage model creativity
  - Single, indisputable short answer that humans cannot solve within 10 min or via using first-page search
- Web-browsing tool < Reasoning Model < Deep Research (search web, synthesize large amounts of information, adapt search strategy, cite each claim)
- When given solution in “unsolvable” cases, Deep Research can correctly find supporting evidence
- Aggregation strategies across 64 output samples
  - Majority voting
  - Weighted voting (prompt model for confidence score)
  - **Best-of-N (select single output with highest confidence score)**
- Model “frequently knows when it’s right”

[1]Wei, Jason, et al. "Browsecomp: A simple yet challenging benchmark for browsing agents." *arXiv preprint arXiv:2504.12516* (2025).

Custom Instructions: Control Flow

Agent Architecture and Trace Structure: The agent architecture consists of a primary manager Agent (also referred to as CodeAgent) that delegates tasks to a search_agent (also referred to as ToolCallingAgent). Every trace consists of several spans (with span_id numbers and parent span_id numbers). Each trace begins with the manager (CodeAgent). The process follows a clear, hierarchical structure where the manager outlines a high-level plan and the search_agent executes the detailed, tool-based steps for each part of that plan.

1. Manager Agent Initiation:
The trace starts with the manager. In its initial child spans, you will observe the following sequence:
- A preparatory survey is created based on the user's query.
- A high-level plan is formulated from this survey.

The Manager agent begins executing Step 1 of its plan.

2. Manager Agent Step 1:
Within the child span for Step 1, the Manager agent decides how to proceed given the initial fact survey and plan. The Manager agent will produce a thought, which may call the search_agent to perform the necessary actions or research.

3. search_agent (ToolCallingAgent) Execution Loop:
Once called, the search_agent begins its own execution loop. In its child spans, you will observe the following sequence:
- A preparatory survey to the specific sub-task it received from the Manager agent.
- A plan tailored to the specific sub-task it received from the Manager agent.

The search_agent executes an initial set of up to four steps. Each step involves an LLM call to generate a tool-call, followed by the tool's execution.
After these initial steps, search_agent synthesizes the information gathered into an updated fact list and refines its plan.
The search_agent may then continue to execute more tool-steps based on this updated plan.

This loop continues until the search_agent has gathered enough information to comprehensively answer the manager's sub-task, at which point it calls final_answer.

4. Returning Control to the Manager agent
The final_answer from the search_agent is returned to the Manager agent, concluding the Manager agent's Step 1. The Manager agent then proceeds to Step 2 of its high-level plan, using the result from the previous step as context.
This entire cycle repeats for all subsequent steps in the Manager Agent's plan.

Whenever you want to point out anything in the trace, cite the span_id number of the span that you are referring to.

Custom Instructions: Few-Shot Examples

Here are some examples of logical consistency issues:

1. {
"evidence": "The plan output content ends with the last step of the plan instead of the `<end_plan>` tag.",
"description": "The plan generation step did not conclude with the required '<end_plan>' tag as specified in the instructions for plan generation.",
},

2. {
"evidence": "Thought: I recall that Girls Who Code published a statistic showing that in the 1980s, women accounted for 37% of computer scientists (or computer science graduates) and that over a span of three decades that figure dropped by 13 percentage points (from 37% down to 24%)",
"description": "The system recalled a statistic about Girls Who Code and the percentage of women in computer science, but this information was not verified using the search tool as planned. The system states \"Thought: I recall that Girls Who Code published a statistic showing that in the 1980s, women accounted for 37% of computer scientists (or computer science graduates) and that over a span of three decades that figure dropped by 13 percentage points\n(from 37% down to 24%). In other words, it took 30 years for that change to occur. Based on that well-circulated statistic that Girls Who Code highlighted, I will output 30 years as the final answer."
}

Custom Instructions: Output Template

Track each agent's system instructions and conversation history, ensuring all subsequent outputs from that agent adhere to its established guidelines and prior dialogue, even when agents speak interchangeably.
For the manager agent and each unique search_agent that may exist in the trace, evaluate the logical consistency for the agent's actions and responses. For each agent, ensure that each response is consistent with the system instructions and prior dialogue.
You must structure your entire response:
Manager Agent
System Instructions
[Paste all system instructions associated with the manager agent.]
Logical Consistency issues
[All Logical Consistency issues associated with the manager agent]

search_agent 0 (if exists)
System Instructions
[Paste all system instructions associated with the search_agent.]
Logical Consistency issues
[List all Logical Consistency issues associated with this search_agent]
...
search_agent n (if exists)
System Instructions
[Paste all system instructions associated with the search_agent.]
Logical Consistency issues
[List all Logical Consistency issues associated with this search_agent]