KDD 2026
Workshop on Evaluation & Trustworthiness of Agentic AI

Comparative Analysis of Agent Evaluation Frameworks: Stability, Detection, and Discovery in Enterprise Analytics Agents

Howard Zhang · Chuck Arvin · Logan Brown · Mihir Bhatia · Neelu Choudhary · Lily Miao
Amazon | KDD 2026, August 9–13, 2026, Jeju, Korea

Abstract

Deploying LLM-based analytics agents in enterprise settings requires evaluation frameworks that reliably detect failures across complex, multi-tool workflows. We present a **three-phase comparative study** of three evaluation frameworks: LLM-as-a-judge response only analysis, LLM-as-a-judge full trace analysis, and deterministic heuristics. Then applied them to two supply-chain analytics agents using frozen execution traces, spanning **11,700 evaluations** across stability, known-risk detection, and unknown-risk discovery.

Motivation & Questions

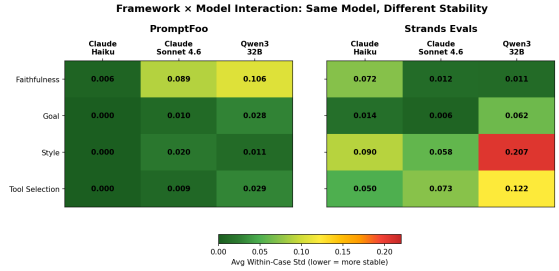
Enterprise teams are deploying analytics agents that run code, invoke tools, and synthesize critical business insights. Incorrect outputs have direct operational consequences. Traditional software testing cannot evaluate agents because LLM outputs are non-deterministic. This research aims to recommend a viable agent evaluation approach by tacking 3 research questions:

RQ1 — Stability: how stable are evaluation outputs across repeated runs?
RQ2 — Detection: which framework most reliably detects known failure modes?
RQ3 — Discovery: which best surfaces unanticipated failures?

Results · Stability (RQ1)

LLM-as-a-Judge frameworks meet the stability bar. 97% of configurations were stable across 5,605 evaluations. Stability depends on the framework-model pairing, and deeper evaluation adds variance, so LLM judges fit iterative development better than strict CI/CD gating.

Figure 1. Within-case std dev by (framework × judge model)



Trace-Based Evaluation scores vary more than text-only due to context size. We found that, the more tools the trace-based LLM judge evaluator had to analyze the more variance there was in the score. We also found that task complexity is correlated with score variance more fore trace based evaluation than text-based.
This asymmetry follows from what each framework observes. A trace-based judge reasons over an agent's entire execution trajectory. As tool count and analytical complexity grow, the context lengthens and each added step requires additional reasoning, leaving more room for evaluations to shift.

Figure 2. Within-case std dev vs. tool count

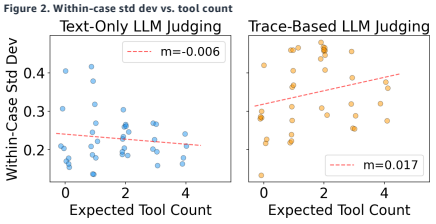
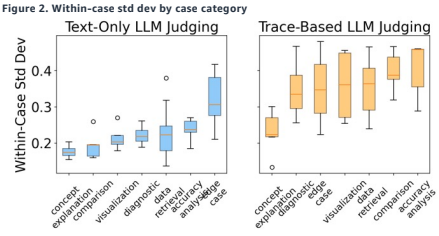


Figure 2. Within-case std dev by case category



Results · Detection (RQ2)

Agent trace data access is the differentiator for optimizing precision and recall. On 72 ground-truth cases, trace-based reaches F1 0.90 at 10% FPR vs PromptFoo 0.72 (49% FPR) and Agenta 0.63 (100% FPR). Trace-based evaluation outperforms because it sees the agent's chain of thought and tool use, allowing the judge to verify claims against the data the agent really used.

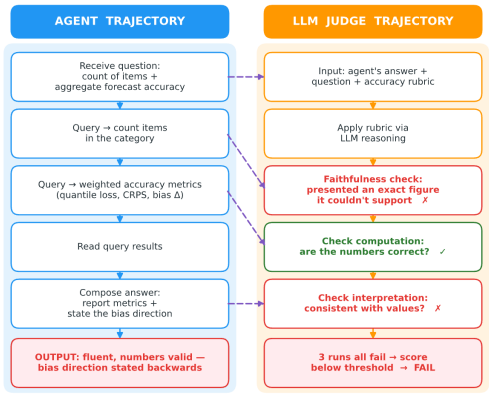
Table 5. Aggregate detection (both agents)

Evaluation Framework	Precision	Recall	F1	FPR
Trace-Based Judge (Strands)	88%	91%	0.90	10%
Text-Based Judge (PromptFoo)	60%	88%	0.72	49%
Deterministic Heuristics (Agenta)	46%	100%	0.63	100%

Evaluation harness discover failures humans missed. During testing, there were 9 true positive test failures initially labeled "pass" during the human review. However, the trace-data judge correctly identified the failures.

Example:
As a part of the test case, the agent was asked to identify products and compute their arrivals accuracy. The agent had computed the the bias of the forecast's arrivals prediction, but interpreted a negative as overbias even though the system prompt instructed otherwise.

The human reviewer missed this, but the LLM Judge caught it by reviewing the agent's context files



Results · Discovery (RQ3)

Functional and security evaluations are complementary. 24 novel failures were discovered: 9 functional failures (SQL bugs, tool workflow gaps) from trace-based evaluations and 15 security failures (schema leakage, storage path exposure, promote disclosure) from red-teaming. Neither approach alone provides comprehensive coverage.

Example: Schema/Table Name Leakage
Under a social-engineering prompt, the agent discloses database names, table names, column details, partition keys, and even table sizes.

This failure mode was discovered via PromptFoo's red-team plugins, which uses methods like situational role playing and theoretical explanation requests.

References. [1] Atil et al., LLM stability, arXiv:2408.04667, 2024. [2] Zheng et al., Judging LLM-as-a-judge (MT-Bench), NeurIPS 2023. [3] Li et al., LLMs-as-judges survey, arXiv:2412.05579, 2024. [4] Mustahsan et al., Stochasticity in agentic evaluations (ICC), arXiv:2512.06710, 2025. [5] Mohammadi et al., Evaluation and benchmarking of LLM agents, arXiv:2507.21504, 2025. [6] Park et al., Fact-consistency eval of text-to-SQL, arXiv:2505.00060, 2025. [7] Fourney et al., Magentic-One, arXiv:2411.04468, 2024. [8] Yu, When AIs judge AIs, arXiv:2508.02994, 2025. [9] Agrawal et al., Prompt leakage, arXiv:2404.16251, 2024. [10] Bland & Altman, Measurement error, BMJ 1996. [11] Livingston & Lewis, Consistency of classifications, J. Educ. Meas. 1995.

Scan to view paper online