

Observability and attribution in large scale multi-agent systems

**Manisha Verma, Executive Director AI/ML,
JP Morgan**

Everyone is talking about agents everywhere

What are they

An agent could be a prompt, a prompt with some tools or a bunch of prompts with bunch of tools.

What do they solve

Ideally each agent solves one task, end to end that satisfies a user need.

Where are they needed

Open ended (or complex) problems like deep research (crunching numbers). Fixed workflows. Branching actions.

What is next?

Run agents like distributed systems: budgeted, traced and tested (not always?).

Agent architecture choices

More autonomy, more value – and harder to observe, attribute and test →



Prompt chain

Fixed sequence of LLM calls; code controls flow

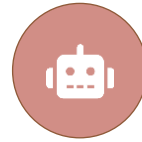
Low autonomy · easy to trace · deterministic topology



Workflow

Router / parallel / evaluator steps defined in code

OTel "invoke workflow" span; predictable paths



Single agent

LLM loops: plan → call tools → observe → repeat

OTel "invoke agent", "plan", "execute tool" spans



Agent + MCP tools

Tools and data exposed via MCP servers

Tool surface owned by teams; stateful/stateless MCP



Multi-agent system

Orchestrator + specialist agents; A2A messaging

Emergent behaviour; failures propagate across agents

Reference: Model Context Protocol specification 2026-07-28 (modelcontextprotocol.io); Agentic AI Foundation, "A2A joins AAIF" (17 Aug 2026, aaif.io); OpenTelemetry GenAI semantic conventions — agent & workflow spans ([open-telemetry/semantic-conventions/genai](https://open-telemetry.com/semantic-conventions/genai), 2026).

Agent implementation and adoption

2026: agents are in production — and observability is now important

57.3%

of organizations have agents in production (LangChain, Jun 2026)

89%

already run some form of agent observability — yet quality remains the #1 barrier (32%)

40%

of large organizations (>\$1B revenue) are scaling agents, up from 27% — McKinsey 2026

17%

of all organizations have deployed agents (Gartner 2026 CIO survey, broader sample); >60% expect to within 2 years

Having traces is not the same as understanding failures. Almost everyone now collects traces, but quality is still the top blocker. The 2026 gap is attribution: which agent, tool or step caused a failure, whether a change helped, and what it cost.

Reference: LangChain, "State of Agent Engineering" (12 Jun 2026, 1,300+ respondents), langchain.com/state-of-agent-engineering; Gartner, Hype Cycle for Agentic AI 2026 (2026 CIO survey); McKinsey, The State of AI 2026.

The demo works. Production is a different claim.

Non-determinism

Same input, different path. You cannot reproduce a failure if you did not record execution in full.

Compounding error

A small early mistake becomes a confident, wrong final answer several agents later.

Silent success

Agents declare completion without checking. Nothing errors, and the output is still wrong.

Cost surprises

Parallel agents multiply tokens. A loop can spend the budget before anyone sees a dashboard.

Reference: Cemri et al., Why do multi-agent LLM systems fail?, NeurIPS 2025 Datasets and Benchmarks; Anthropic Engineering (2025), multi-agent research system.

Six design patterns cover most systems

- **Orchestrator and workers:** one lead plans and delegates, workers execute in parallel.
- **Planner, executor, verifier:** separate the agent that acts from the agent that checks.
- **Router:** send each subtask to the cheapest model and tool that can do it well.
- **Blackboard:** agents read and write a shared, versioned state instead of passing long chats.
- **Pipeline with gates:** fixed stages, each with a schema check before the next begins.
- **Human in the loop:** approval gates for irreversible, high-value or regulated actions.

Why agents are harder to observe than services

Services are deterministic

A service either returns the right rows or it does not.
An agent can be fluent, plausible and wrong.

Agents choose their own path

Control flow is generated at run time, so there is no fixed call graph to monitor in advance.

State lives in language

Context, memory and hand-offs are text. Truncation and summarization lose facts without throwing an error.

The environment talks back

Tools, web pages and other agents can inject instructions, stale data or contradictions mid-run.

Observability means answering a question you did not predict

Traces

The full run as a tree: who decided what, in what order, with which inputs and outputs.

Logs

Structured events at each step, searchable by run, agent, tool, user and cost.

Metrics

Rates, latency, tokens and cost per task, per agent and per tool, over time.

Evaluations

Quality scores attached to the same spans, so a bad answer points to the step that caused it.

Why Trace? Trace What?

Stateless MCP

- No initialize handshake or Mcp-Session-Id header
- Session-based correlation is gone: propagate trace context on every request
- Multi round-trip requests need explicit linking

A2A: agent ↔ agent

- Opaque remote agents: you see messages, not not reasoning
- Attribution crosses organizational boundaries
- Task lifecycle gives natural span boundaries

Implication

- Instrument at the protocol layer, not only inside inside agents
- Log server discovery results and tool-schema versions
- Treat every hop as a trust and audit boundary

- **Root span:** the user task, its owner, budget and deadline.
- **Agent spans:** plan, delegation and synthesis, each with model and prompt version.
- **Tool spans:** arguments, result, latency, retries and errors.
- **Eval spans:** verifier and judge scores on the exact output they judged.

Log for attribution, not for volume

Decide sampling before launch: keep every failed run, sample successes, and set retention by regulation, not by storage cost alone. Redact at capture, not at query time.

5

fields on every span: run id, agent id, step id, model and prompt version, parent span

3

things to capture in full: tool arguments, tool results and the final verifier decision

0

secrets, credentials or unredacted customer data that should ever reach a trace store

Unaided, the best models are poor failure detectives

Reading a whole log and guessing does not work. Attribution needs structure: search over the trace, evidence per step and a method that can say why.

<10%

step-level accuracy of strong reasoning models on Who and When in early attribution studies

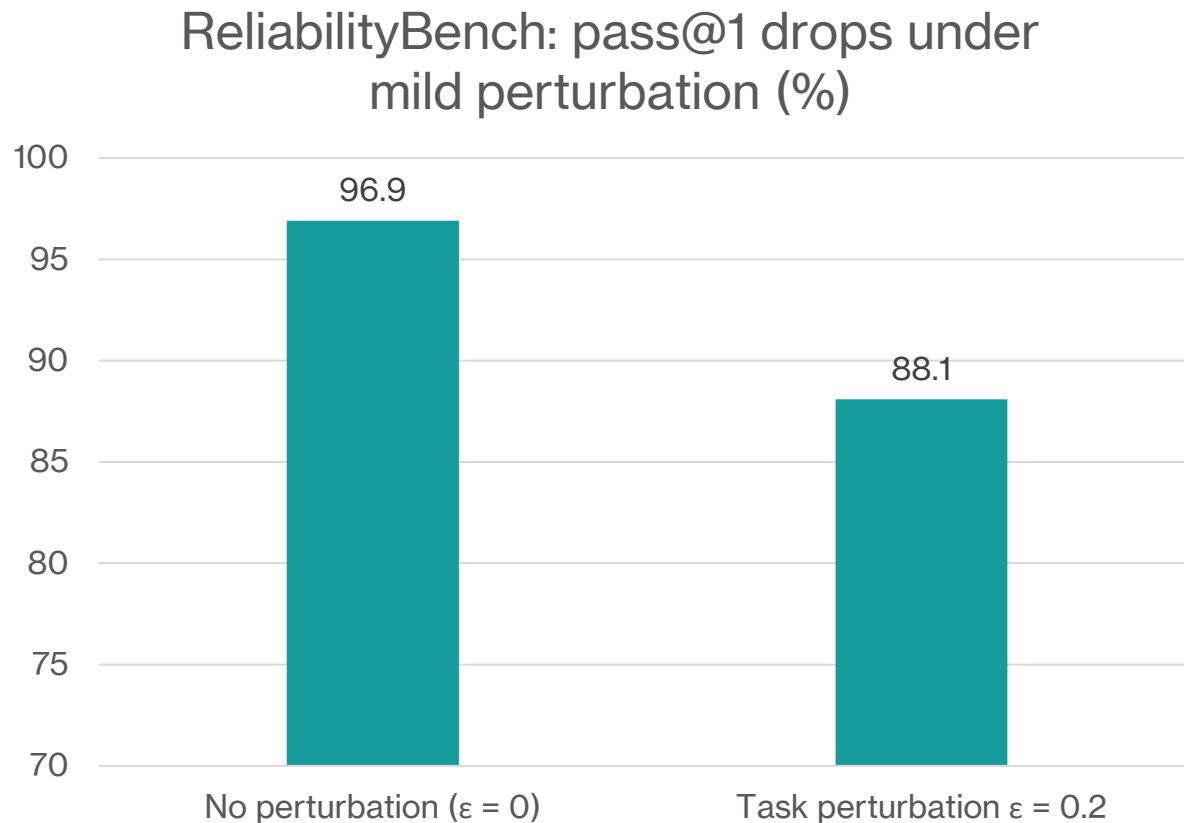
127

expert-labelled failed trajectories in the original Who and When benchmark

3

baseline strategies tested: all at once, step by step and binary search over the trace

Reliability under production stress



A reliability surface, not a single score

- $R(k, \epsilon, \lambda)$: k = repeated runs scored with pass^k (consistency), perturbation, λ = injected fault intensity
- **Rate limiting** was the most damaging fault — an infrastructure failure, not a reasoning one
- **Architecture matters**: ReAct held up better than Reflexion under stress
- **HAL (ICLR 2026)** now adds a reliability dashboard: robustness, safety
- **Lesson**: monitor variance across reruns and under faults; gate releases on pass^k, not best-case accuracy

Reference: "ReliabilityBench: Evaluating LLM Agent Reliability Under Production-Like Stress Conditions", arXiv:2601.06112 (Jan 2026); Holistic Agent Leaderboard (HAL) reliability dashboard, ICLR 2026 (OpenReview vUaY1t64ZZ).

Recommendations: Standardise on OpenTelemetry GenAI conventions

One vocabulary

invoke_agent for a run, chat for a model call, execute_tool for a tool call, in every framework.

Attributes that matter

Model and provider, input and output tokens, tool name, finish reason and error status on every span.

Privacy by design

Prompts and completions are opt-in, not defaults. Capture them as controlled span events, redacted.

Vendor freedom

Instrument once, export anywhere. Avoid a tracing format only one vendor can read.

Recommendations: Differentiate errors, use old classic tools

- 1 Configuration, parsing and serialization defects look like "the model got it wrong" in a dashboard — classify them separately
- 2 Log framework, SDK, model and tool-schema versions on every trace so regressions can be inspected
- 3 Classical contract tests and version pinning catch a large share of incidents before any LLM evaluation runs
- 4 Separate infra/system errors from reasoning errors: different owners, different fixes

Attribution asks who broke it, and at which step

- Agent level: which agent is responsible for the failure?
- Step level: at which exact step did failure become inevitable?
- Cause level: was it a bad prompt, a wrong decomposition, a tool error or lost context?
- Why it is hard: the error surfaces far from where it was introduced, inside long, intertwined logs.

Attribution is moving beyond one blame, one step

Several open datasets exist to benchmark attribution systems.

- Spectrum analysis (FAMAS): rank agents by how often they appear in failing versus passing runs.
- Causal graphs (CHIEF): rebuild the run as a dependency graph and backtrack hierarchically to the root cause.
- Trace benchmarks (TRAIL): 148 OpenTelemetry traces, 841 errors; the best model reached only 11% joint accuracy on category and location.
- Honest limit: some failures have several contributing steps. Report a ranked, evidenced shortlist, not false certainty.

Trained tracers beat giant general models at attribution

Counterfactual replay

Replay the failed run with one agent or step changed. If the outcome flips, you have causal evidence.

Fault injection

Programmatically inject known faults to build labelled training data at scale (TracerTraj).

A small specialist

AgenTracer-8B, trained with multi-granular reinforcement learning, beat Gemini 2.5 Pro and Claude 4 Sonnet by up to 18.18 points on Who and When.

Feedback that fixes

Feeding attribution back to MetaGPT and MaAS improved task performance by 4.8 to 14.2%.

Open Problems

Attribution at scale

Step-level accuracy is still low, traces exceed context windows, and many failures have no single cause.

Evaluation that predicts production

Benchmarks saturate and leak into training. We need live, private, continuously refreshed evaluation.

Standards still maturing

GenAI conventions are at development stability. Agent identity, memory and multi-agent hand-off spans are not settled.

Self-correction that is safe

Agents that fix themselves can also entrench errors. Verification of the fix itself is unsolved.

Thank You