WSO2CON NORTH AMERICA

May 20 - 22, 2026 | Austin, Texas, USA

Managing the Agent Lifecycle at Scale



Malith Jayasinghe

VP of AI

Overview

Understanding Agentic Systems

Characteristics of agentic systems and how they differ from traditional software systems.
Critical for scaling AI agents.

Agent Development Life Cycle (ADLC)

Emerging lifecycle for building and operating agentic systems.
Focus on key stages, challenges, and operations.

Scaling AI Agents

Evolving ecosystem to address operational complexity.
Role of centralized control planes in managing scale.

Understanding Agentic Systems

Agentic systems introduce fundamentally different characteristics compared to traditional software

- **Behavior**
- **Execution**
- **Context Usage**
- **Failure Traits**
- **Evolution**

These characteristics will change the way we build, operate, and govern agentic systems

Behaviour Model

Traditional Systems

- Deterministic behavior
- Same input produces the same output
- Behavior is usually **predictable** and easier to test

Agentic Systems

- Nondeterministic behavior
- Same input may produce different outputs
- Behavior **adapts** based on context memory, models and runtime conditions

Agent behavior becomes less predictable and harder to validate using traditional testing approaches

Execution Model

Traditional Systems

- Static execution paths
- Workflows are predefined during development
- Limited branching complexity

Agentic Systems

- Dynamic runtime orchestration
- Agents decide what actions to take during execution
- Execution paths may emerge during execution

Autonomous decision making introduces new operational and governance challenges

Context Model

Traditional Systems

- Limited runtime context
- Execution (mainly) depends on the **application state and predefined inputs**
- External information is usually explicitly integrated

Agentic Systems

- Context-driven execution
- Decisions **depend on prompts, memory, retrieved knowledge, and tool outputs**
- Runtime context heavily influences behavior and outputs

Context management becomes a critical part of system behavior and reliability

Failure Model

Traditional Systems

- Failures are usually **binary and easier to identify**
- Issues are often reproducible and deterministic
- **Near-zero failure** is commonly expected

Agentic Systems

- Failures become **probabilistic**
- Issues may appear intermittently and be harder to reproduce
- **Acceptable levels of failure** (depend on the use case and risk profile)

Reliability must be measured and managed differently in agentic systems

Evolution Model

Traditional Systems

- Behavior changes mainly through new deployments/releases
- Logic remains fixed between release
- Stable release cycles

Agentic Systems

- Behavior can change without new deployments
- Models, Context, tools, evolve over time
- Continuously evolving behavior



Continuous monitoring, evaluation and governance become essential over time

Agent Development Life Cycle (ADLC)

Define	Design	Develop	Evaluate	Deploy	Operate & Govern
• Goals • Responsibilities	• Agents • Orchestration • Tools / Memory • Guardrails	• Workflows • Integrations • Context Frameworks	• Safety • Reliability Accuracy	• Environments Configuration • Rollouts	• Observability • Policies • Drift • Optimization

Defining Agent Scope

Problem Definition

- What problem is the agent solving?
- What business outcome is expected?

Autonomy & Approvals

- Level: Read/suggestions-only, Semi-autonomous, Fully-autonomous

Data & Tool Access

- Access to APIs, MCP servers, databases, and enterprise systems?

Allowed Actions

- Read, generate, update, or trigger workflows?

Why Scope Matters

- Scope impacts **risk, complexity, and governance** requirements
- Influences: Guardrails, Authorization, Observability, and Evaluations
- Poorly defined scope creates operational instability and gaps

Design

Single vs. Multi-agent Architecture

(Deterministic) work-flows vs. agentic integrations

Tools, API and MCP Integration

Short-term vs. Long-term Memory

Context Engineering and Retrieval

Identity, Auth, and Guardrails

Observability and Evaluation

Deployment and Runtime Isolation

Develop

Select the Agent Framework

Framework ecosystems have matured significantly

Examples: LangGraph, CrewAI, AutoGen, Semantic Kernel, WSO2 Agent Builder

Implement Agent Logic

- Prompts
- Planning and reasoning
- Tool orchestration
- Workflow execution
- Context and memory handling

Build Enterprise Integrations

- APIs
- MCP servers
- Databases
- Enterprise systems
- Secure tool access and execution

Add Operational & Production Capabilities Early

- Observability instrumentation
- Evaluation hooks
- Guardrails and authorization controls

Evaluate

Evaluation Timing

- Development time Eval vs Online Eval

Evaluation Methods

- LLM as a Judge
- Human
- Statistical methods

Evaluation Metric

- Ragas for RAG
- Agents, Path efficiency, relevance

Dataset Strategy

- Evaluate With or Without a Reference Dataset
- Golden dataset
- Open-ended evaluations

Multi-Level Evaluation

- Evaluate at different levels: Multiple agents, single agent, LLM level.

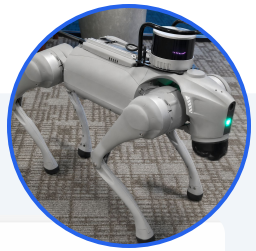
Evaluate

BinFinder — Agent running on
Unitree Go2 EDU

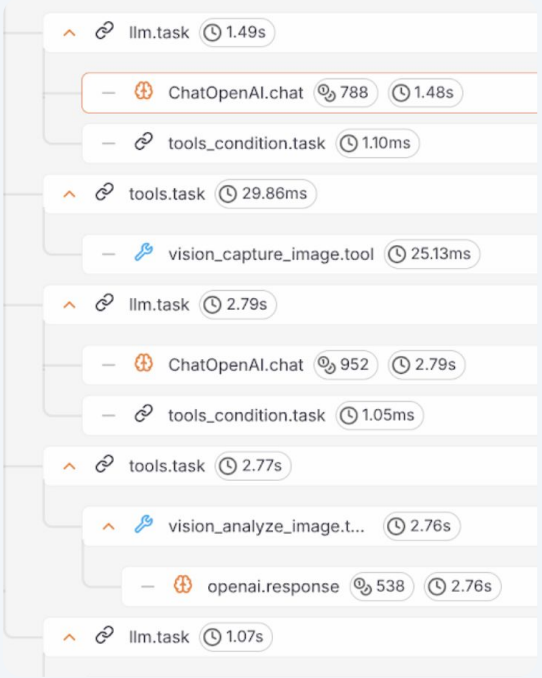
Instruction:
"Find a trash bin in this area, look around if needed, and tell me its color."

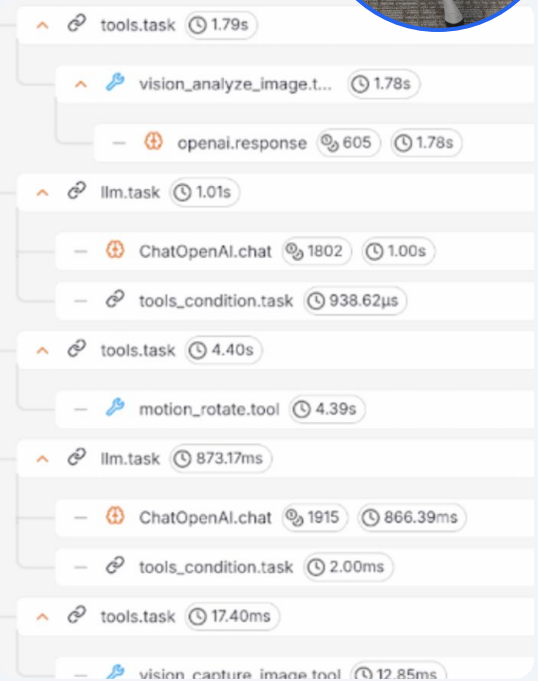
What Happened

- Initial run failed.
- Trace analysis helped identify prompt and behavior issues.
- **BinFinder successfully completed the task**



Trace Details





Evaluate

The Problem

Built-in evaluators failed to capture the intent behind the agent's behavior

For example, low path efficiency due to repeated movement and scanning

Why this happens:

Exploration and repeated tool calls were expected for this task

Repeated movement helped improve task accuracy and environmental understanding



We developed a custom evaluator designed for exploration-based robot-agent traces

Deploy

Deploy And Promote Across Environments

Dev → Stage → Production

- Validation testing
- Production readiness

Configure Runtime Parameters And Secrets

- LLM keys
- API keys
- Environment variables

Configure Guardrails And Runtime Policies

- Safety controls
- Access policies
- Approval rules

Operate and Govern

Runtime Observability

Continuously monitor agent behavior, tool usage, and reasoning flows

Detect Guardrail Violations

Unsafe actions
Policy violations
Sensitive data exposure

Identity And Access Control

Control tool access permissions and runtime actions

Perform Online Evaluations

Periodic evaluations, Sampling runtime quality checks

Detect Drift And Behavioral Changes

Prompt drift, Model drift, Tool behavior changes

Monitor Runtime Cost And Performance

Token usage, Latency, Infrastructure and model costs

Scaling Agentic AI in the Enterprise: Challenges and Emerging Platform Capabilities

WSO2

Current landscape: Multiple frameworks, teams, and internal/external agents

The rate of innovation in AI agents has increased significantly, creating complex management needs

Frameworks

Rapid innovation across diverse tools:

- CrewAI
- LangChain
- LangGraph
- WSO2 Agent Builder

Teams

Scale brings organizational challenges:

- Multiple development teams
- Internal operations
- External integrations

Agents

Distributed execution environments:

- Deployed on-platform
- Managed externally

  e.g., GitHub workflows

Managing agents built across multiple frameworks and spanning internal and external boundaries introduces significant operational complexity

Current landscape: Multiple frameworks, teams, and internal/external agents

WSO2 Internal Areas

- Revenue Generation
- Finance
- Delivery
- People Ops / Admin
- Engineering

Examples at WSO2

- Customer Success Agents
- Sales Engineering Agents
- HR Agents
- WSO2Con Agents

Frameworks Used

- LangGraph
- Anthropic SDK
- WSO2 Agent Builder

Agent Observability

The Observability Gap

Traditional stacks were built for deterministic systems and APIs.

- Debugging AI agents was extremely difficult
- Teams relied heavily on custom implementations
- OpenTelemetry initially lacked agent-specific support

Limited Visibility Into:

- Prompts
- LLM calls
- Tool invocations
- Agent reasoning flows
- Context usage
- Multi-agent interactions

Scaling observability across multiple frameworks and runtimes became challenging

Emergence of the AI Observability Ecosystem

OpenLLMetry

Extended OpenTelemetry for AI workloads.

Added tracing for:

- LLM calls
- Prompts & Tokens
- Embeddings
- Tool invocations

OpenInference

Introduced AI semantic conventions and interoperability standards.

- Standardized AI traces across tools
- Unified framework standards

OTel GenAI

Official OpenTelemetry standardized support for:

- LLMs & Embeddings
- Vector databases
- Agents
- MCP-related tracing

Key Point: The industry began extending observability standards specifically for AI and agentic systems.

Maturing AI-Agent Observability Tooling

SDKs

- Agent-specific observability (e.g. Trace loop)
- LLM/Tool-call/tokens/Execution paths visibility

Platforms & Visualization

- AI-specific dashboards & analysis
- Agent execution & cost visualizations

Gaps Still Remain

- Inconsistencies in implementations across frameworks (missing attributes)
- Limited visibility into reasoning behavior (for different frameworks)
- Large-scale trace management challenges(sampling, scaling, costs)

Evolution of Agent Evaluation

Custom Scripts

Early agent teams relied heavily on custom evaluation scripts with limited tooling and lack of mature standards

Emergent Ecosystem

Evaluation is becoming ecosystem-driven, with agent frameworks providing built-in capabilities

New Metrics & Techniques

- RAG metrics (groundedness, faithfulness, retrieval quality)
- Tool-use correctness
- Agent trajectory & path efficiency
- Safety, alignment, and multi-agent coordination

Online Platform Capabilities

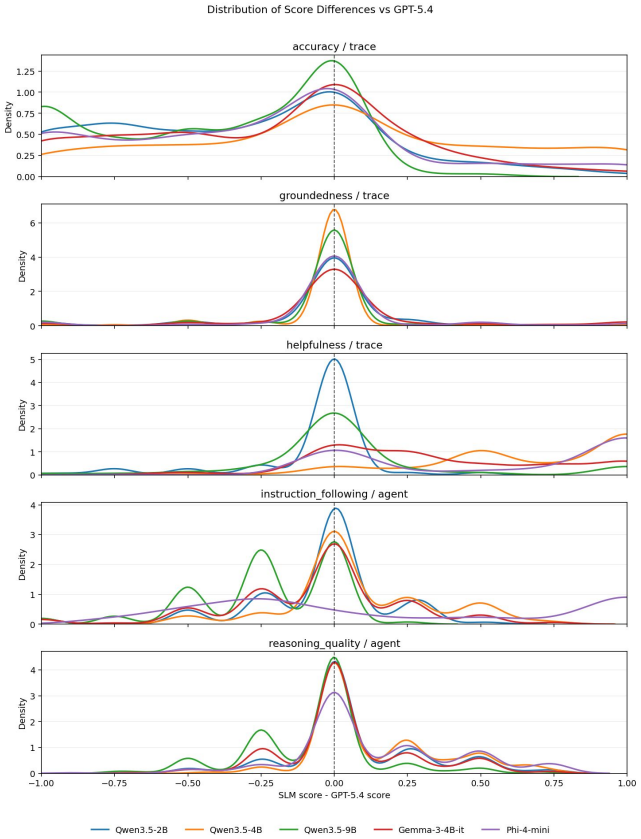
Platforms adding continuous online eval

- Drift detection and regression analysis
- Runtime scoring and production metrics

Evaluation can be expensive: Sampling, Risk-based, SLMs

Strategic Shift: Moving from isolated scripts to standardized, platform-integrated evaluation

Distribution of Differences Between LLM and SLM Evaluation Scores Across Different SLMs



Scaling Challenges in Agent Evaluation

Multi-agent Systems

Harder to evaluate. Failures from coordination, handoffs, tool choices, or cascading issues across agents

Online Evaluation

Necessary as behavior changes with models, tools, context, memory, and user patterns in production

Expense & Resources

Full evaluation is costly. Teams rely on sampling, prioritized sets, and risk-based evaluation to manage time/tokens

Continuous Evolution

Evaluation data must evolve. Production traces, feedback, and edge cases must feed back into datasets

Domain Specificity: Generic evaluators often fail; domain-aware custom evaluators are often required

Scaling challenges in governance

Early AI-agent Systems Were Simpler
Single team, framework, and rules in code (PII masking, prompt filtering)

Enterprise Scale Limitations
Multiple teams and distributed runtimes make governance fragmented and inconsistent

Essential Capabilities

- Centralized governance (Guardrails, Tokens, Budgets)
- Critical Role-based access control (RBAC)
- Operational Analytics, Alerts, and Remediation

The Industry Shift
Capabilities moving into unified platforms:

- AI-GWs / MCP-GWs
- IAM and Agent-ID specialized products

Governance tooling itself is a scaling challenge; organizations must integrate multiple platforms

The Need for a Centralized Control Plane for Agentic AI

- The industry is making progress through language-level abstractions for building agents, agent SDKs/frameworks, and tools for observability, evaluation, and governance
- However, these point solutions are not sufficient at enterprise scale
- Organizations still struggle with:
  - Fragmented visibility across agents and runtimes
  - Inconsistent governance and security controls
  - Operational silos across teams and frameworks
  - Lack of centralized control over agents, models, tools, and MCP interactions
- This is why a centralized Agent Control Plane is needed

Final Thoughts

- Agentic systems behave fundamentally differently from traditional software systems, and SDLC alone is not sufficient for managing agents
- ADLC emerged to address these challenges and continues to evolve, while tooling and platformization of capabilities are rapidly advancing
- As organizations scale, the number of agents, frameworks, and internal/external interactions increases, making management and operations extremely difficult
- Some of these challenges are being addressed through tools such as agent observability platforms
- However, these solutions remain fragmented and do not provide centralized control or visibility across all aspects of agents
- This is why a centralized control plane is needed

WSO2 Agent Manager

The open enterprise control plane for AI Agents to securely deploy, manage, and govern AI agents at scale.

Any Agent, Any Framework

Framework-agnostic:
LangGraph, CrewAI, Microsoft ADK, AWS Strands, custom, +.

Runtime-agnostic:
Run anywhere.

Any Deployment

Agent Manager can be:

- Self-hosted
- SaaS
- Hybrid

No hyper-scaler lock-in.

Open Platform

Open Source & Standards:
Apache 2.0, MCP, A2A, OTEL, +

Cloud Native:
OpenChoreo, K8s

No vendor lock-in.

Platform Mandate: Securely deploy, manage, and govern AI agents at scale for any environment.

WSO2CON NORTH AMERICA

May 20 - 22, 2026 | Austin, Texas, USA

Thank You!



Malith Jayasinghe

VP of AI