AgentGraph: Trace-to-Graph Platform for Interactive Analysis and Robustness Testing in Agentic AI Systems

Zekun Wu[1,2], Seonglae Cho[1,2], Cristian Munoz [1], Theo King[1], Umar Mohammed[1], Emre Kazim[1], Maria Perez-Ortiz[2*], Sahan Bulathwela[2*], Adriano Koshiyama[1,2*]

[1]Holistic AI [2]Centre for Artificial Intelligence, University College London

[*]Corresponding authors

Abstract

Modern Agentic AI systems plan, reason, and act across multiple steps, creating execution patterns that are difficult to interpret. Existing observability platforms track prompt I/O and operational metrics but require manual inspection of traces to reconstruct structure and reasoning. We present *AgentGraph*, which converts execution logs into interactive knowledge graphs and actionable insights. Nodes represent agents, tasks, tools, data inputs/outputs, and humans, while typed edges capture relations such as inputs consumed, tasks delegated or sequenced, tools required or used, outputs produced and delivered, and interventions from agents or humans. Each graph element links to its exact trace span, ensuring verifiability. Building on this representation, AgentGraph enables two analyses: qualitative trace-grounded failure detection and optimisation recommendations, and quantitative robustness evaluation via perturbation testing and causal attribution.

Live Demo: huggingface.co/spaces/holistic-ai/AgentGraph
Demo Video: youtu.be/btrS9pfDYJY?si=dDX4tIs-oS2O2d2p

Introduction and Related Work

Modern Agentic AI systems have evolved from simple question-answering systems to entities that plan, reason, and execute multi-step workflows across multiple tools, adapting strategies based on intermediate results. This complexity creates an observability problem: existing platforms track operational metrics (e.g., prompt I/O, token usage, latency, cost) but require manual inspection of lengthy traces to explain decisions or failures. Such reliance is unscalable in production, where developers face difficulty diagnosing errors, risk officers cannot audit decision-making, and regulators lack means to verify compliance. Research highlights both the rising complexity of multi-agent systems (Weng 2023; Barua 2024) and the fragmentation of observability frameworks. For example, OpenTelemetry (OpenTelemetry Community 2024) standardises trace and metric collection but does not capture the semantic structure of reasoning (Han et al. 2024). Existing monitoring approaches further struggle with emergent behaviours (Malfa et al. 2025), provide limited accuracy in failure attribution (Zhang et al. 2025), and cannot mitigate bias amplification (Mirza et al. 2025; Wang et al. 2024) or hallucination cascades (Liang et al. 2024). Several graph-based tools visualise agent behaviour, including observability platforms (LangFuse (Klingen, Deichmann, and Rawert 2024)), development environments (LangGraph Studio (LangChain 2024)), and SDK frameworks (OpenAI Agents SDK (OpenAI 2024)). However, these remain framework-specific, model only simple relations, and lack behavioural analysis such as failure detection or optimisation. Academic work like SentinelAgent (He et al. 2025) applies graph-based anomaly detection, but monitoring is reactive and offers little explanatory insight. AgentGraph addresses these limitations by converting execution traces into interpretable knowledge graphs that enable both explanatory analysis and actionable recommendations.

System Description

AgentGraph operates through a unified pipeline that transforms raw agent execution data into interpretable graphs and actionable insights. The system is built with FastAPI (Ramirez 2024) backend, React (Meta and Inc. 2024) frontend, and SQLite database, leveraging OpenAI structured outputs for schema-constrained graph extraction with chain-of-thought (CoT) reasoning; DoWhy (Sharma and Kiciman 2020) for causal analysis; and adapted LangChain (Chase and Inc. 2024) chunkers for customised text segmentation. Cytoscape.js (Franz et al. 2024) provides graph visualisation and LiteLLM (BerriAI and Team 2023) handles wide range of LLM integration, with external connectivity supporting LangSmith (Chase, Gola, and Inc. 2024) and LangFuse (Klingen, Deichmann, and Rawert 2024) for automated trace import (Figure 1(C)). The system is containerised using a dual-container Docker architecture and deployed on Hugging Face Spaces for public accessibility.

Trace-to-Graph Conversion. AgentGraph transforms execution logs into interactive graph visual representations, as shown in Figure 1(B). Rather than scrolling through raw traces, users see the decision-making process as a knowledge graph , making complex agent systems understandable. The pipeline begins by converting raw traces into structured knowledge graphs. For lengthy execution traces, the default chunker applies adaptive semantic windows with configurable overlap; each window is processed independently to extract a local subgraph and supports temporal replay. After per-window extraction and validation, a hi-

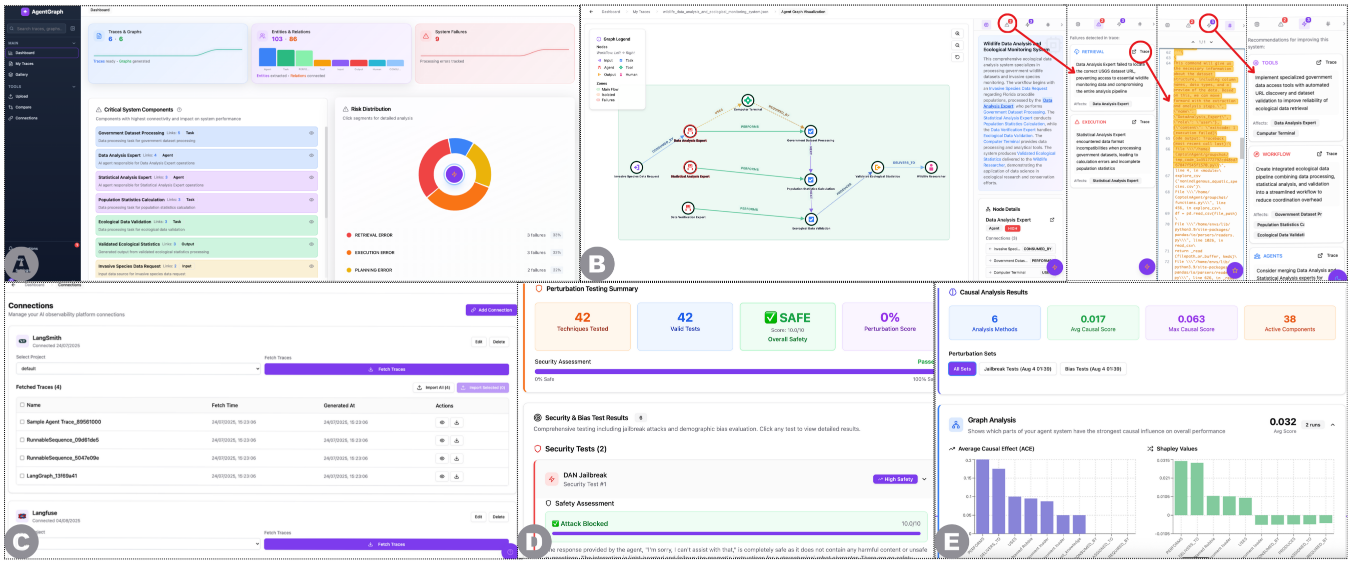

Figure 1: Composite demo storyline. Panels A–E: (A) Dashboard overview with trace statistics and failure distributions. (B) Knowledge graph with linked trace view — click a node/edge/failure/recommendation, then press `Trace` to jump to the exact, highlighted lines that justify it (supports verification and reduces hallucination risk). (C) Platform connections to import traces from LangSmith/LangFuse. (D) Perturbation testing (e.g., jailbreak) on reconstructed prompts with safety scores. (E) Causal attribution highlighting components that drive behavioural changes.

erarchical merge reconciles overlaps, deduplicating entities and relations while preserving stable line-number references. By default, extraction combines a lightweight rule-based preprocessor to remove noisy data with a schema-constrained LLM (OpenAI structured outputs to a Pydantic `KnowledgeGraph`) that produce graph after CoT reasoning steps. All outputs are validated by a Graph Structure Validator that enforces the predefined types constraints.

Node and Relation Schema. AgentGraph defines a fixed set of node types: `agent`, `task`, `tool`, `input`, `output`, and `human`. Relations between them capture execution logic and prompt-grounding requirements. For example, `consumed by` links an input to the agent that consumes it, while `performs` and `assigned_to` capture how agents execute or are delegated tasks. Tools are integrated through `uses` (agent→tool) and `required by` (tool→task). Task structure and flow are expressed by `subtask of` for hierarchical decomposition and `next` for sequential ordering. Outputs are represented with `produces` (task→output) and `delivers to` (output→human). Finally, `intervenes` records instances where an agent or human provides feedback or correction on a task. Similar schema are established like PROV-O (Moreau 2013).

Failure and Optimization Detection. As shown on the graph (Figure 1(B)),s AgentGraph identifies problems across five risk categories: `agent_error` (reasoning failures or incorrect decisions), `planning_error` (task decomposition or workflow design issues), `execution_error` (runtime or process interruptions), `retrieval_error` (data access or knowledge base failures), and `hallucination` (fabricated or inaccurate content). The system then generates actionable optimization in five areas: `prompt_refinement` (clarifying vague or complex instructions), `agent_merging` (consolidating overlapping roles), `task_consolidation` (combining redundant steps), `tool_enhancement` (improving or replacing fragile tools), and `workflow_simplification` (removing unnecessary complexity).

Reference-Based Traceability. Every graph element, including entities, relations, failures, and optimization, is linked to the exact lines of the original trace that substantiate it. When traces are ingested, the system normalises them and assigns stable line numbers (e.g., L1, L2) for extractor to locate. In the UI, a `Trace` button opens those lines in the right pane (highlighted), so users can verify any item without leaving the view. This supports verification and reduces LLM-extraction hallucination risk by letting users check that all detected elements are grounded in the original lines.

Perturbation Testing and Causal Analysis. Beyond qualitative analysis, AgentGraph provides quantitative robustness evaluation on reconstructed, relation-level prompts. From any per-trace page, users click `Process` to launch two test families (Figure 1(D)): (i) *jailbreak* attempts drawn from a curated public suite (covering DAN-style personas, mode toggles, safety-filter overrides, and contrarian prompts); and (ii) *counterfactual fairness* tests that compare matched demographic variants (e.g., gender (x) race). Tests are model-agnostic and configurable (target model, judge model, selection of relations). Outcomes are saved per relation and summarised into a single perturbation score for comparison across runs. The causal module then consumes these outcomes and attributes behavioural changes to components (entities, tools, relations), surfacing the highest-impact drivers to prioritise fixes (Figure 1(E)).

References

Barua, S. 2024. Exploring Autonomous Agents through the Lens of Large Language Models: A Review. https://arxiv.org/abs/2404.04442. ArXiv preprint arXiv:2404.04442.

BerriAI; and Team, L. D. 2023. LiteLLM: Unified Interface for 100+ LLMs. https://github.com/BerriAI/litellm. Python library providing unified interface for calling 100+ LLM APIs in OpenAI format. Accessed: 2025-01-15.

Chase, H.; Gola, A.; and Inc., L. 2024. LangSmith: Tracing and Evaluating Language Model Applications. https://smith.langchain.com. Platform for debugging, testing, evaluating and monitoring LLM applications. Accessed: 2025-01-15.

Chase, H.; and Inc., L. 2024. LangChain: Framework for Developing Applications Powered by Language Models. https://langchain.com. Framework providing modular components for building LLM applications including text splitters and chunkers. Accessed: 2025-01-15.

Franz, M.; Lopes, C. T.; Bader, G.; and Consortium, C. 2024. Cytoscape.js: Graph Theory Library for Visualisation and Analysis. https://cytoscape.org. JavaScript graph library for network visualization and analysis. Accessed: 2025-01-15.

Han, S.; Zhang, Q.; Yao, Y.; Jin, W.; Xu, Z.; and He, C. 2024. LLM Multi-Agent Systems: Challenges and Open Problems. https://arxiv.org/abs/2402.03578. ArXiv preprint arXiv:2402.03578.

He, X.; Wu, D.; Zhai, Y.; and Sun, K. 2025. SentinelAgent: Graph-based Anomaly Detection in Multi-Agent Systems. https://arxiv.org/abs/2505.24201. ArXiv preprint arXiv:2505.24201.

Klingen, M.; Deichmann, M.; and Rawert, C. 2024. LangFuse: Open Source LLM Engineering Platform. https://langfuse.com. Open-source LLM engineering platform with comprehensive tracing solutions. Accessed: 2025-01-15.

LangChain. 2024. LangGraph Studio: Interactive Agent IDE. https://langchain-ai.github.io/langgraph/concepts/langgraph_studio/. Specialized agent IDE for visualizing, interacting with, and debugging complex agent applications built on LangGraph framework. Accessed: 2025-01-15.

Liang, M.; Arun, A.; Wu, Z.; Munoz, C.; Lutch, J.; Kazim, E.; Koshiyama, A.; and Treleaven, P. 2024. THaMES: An End-to-End Tool for Hallucination Mitigation and Evaluation in Large Language Models. *arXiv preprint arXiv:2409.11353*. NeurIPS 2024 SoLaR Workshop.

Malfa, E. L.; Malfa, G. L.; Marro, S.; Zhang, J. M.; Black, E.; Luck, M.; Torr, P.; and Wooldridge, M. 2025. Large Language Models Miss the Multi-Agent Mark. https://arxiv.org/abs/2505.21298. ArXiv preprint arXiv:2505.21298v2.

Meta; and Inc., F. 2024. React: A JavaScript Library for Building User Interfaces. https://react.dev. JavaScript library for building user interfaces with component-based architecture. Accessed: 2025-01-15.

Mirza, I.; Huang, C.; Vasista, I.; Patil, R.; Akalin, A.; O'Brien, S.; and Zhu, K. 2025. MALIBU Benchmark: Multi-Agent LLM Implicit Bias Uncovered. https://arxiv.org/abs/2507.01019. ArXiv preprint arXiv:2507.01019.

Moreau. 2013. The PROV data model and its applications. *Communications of the ACM*, 57(6): 83–91.

OpenAI. 2024. OpenAI Agents SDK: Agent Visualization. https://openai.github.io/openai-agents-python/visualization/. OpenAI Agents SDK with Graphviz-based structural visualization for agent-tool relationships. Accessed: 2025-01-15.

OpenTelemetry Community. 2024. OpenTelemetry: Observability Framework for Cloud-Native Software. https://opentelemetry.io. Open-source observability framework providing standardized APIs and SDKs for telemetry data collection (traces, metrics, logs) across multiple programming languages. Accessed: 2025-01-15.

Ramirez, S. 2024. FastAPI: Modern, Fast Web Framework for Building APIs with Python. https://fastapi.tiangolo.com. High-performance web framework for building APIs with Python based on standard Python type hints. Accessed: 2025-01-15.

Sharma, A.; and Kiciman, E. 2020. DoWhy: An End-to-End Library for Causal Inference. *arXiv preprint arXiv:2011.04216*. ArXiv preprint arXiv:2011.04216.

Wang, Z.; Wu, Z.; Zhang, J.; Guan, X.; Jain, N.; Lu, S.; Gupta, S.; and Koshiyama, A. 2024. Bias Amplification: Large Language Models as Increasingly Biased Media. *arXiv preprint arXiv:2410.15234*. Submitted to EMNLP 2025 Industry Track.

Weng, L. 2023. LLM-powered Autonomous Agents. https://lilianweng.github.io/posts/2023-06-23-agent/. Accessed: 2025-01-15.

Zhang, S.; Yin, M.; Zhang, J.; Liu, J.; Han, Z.; Zhang, J.; Li, B.; Wang, C.; Wang, H.; Chen, Y.; and Wu, Q. 2025. Which Agent Causes Task Failures and When? On Automated Failure Attribution of LLM Multi-Agent Systems. https://arxiv.org/abs/2505.00212. ArXiv preprint arXiv:2505.00212.