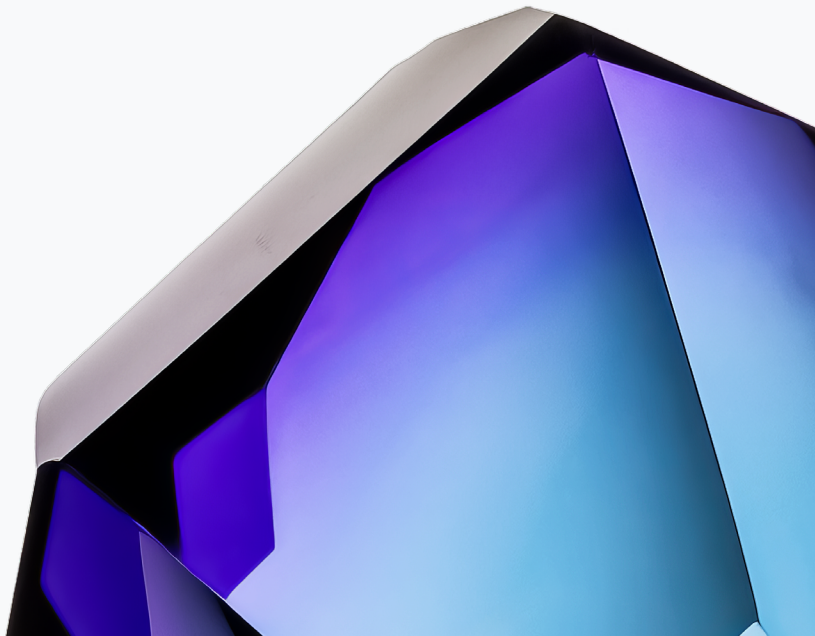
Agents Companion

Authors: Antonio Gulli, Lavi Nigam,
Julia Wiesinger, Vladimir Vuskovic,
Irina Sigler, Ivan Nardini, Nicolas Stroppa,
Sokratis Kartakis, Narek Saribekyan,
Anant Nawalgaria, and Alan Bount

Google



Acknowledgements

Editors & curators

Anant Nawalgaria

Content contributors

Steven Johnson

Hussain Chinoy

Designer

Michael Lanning

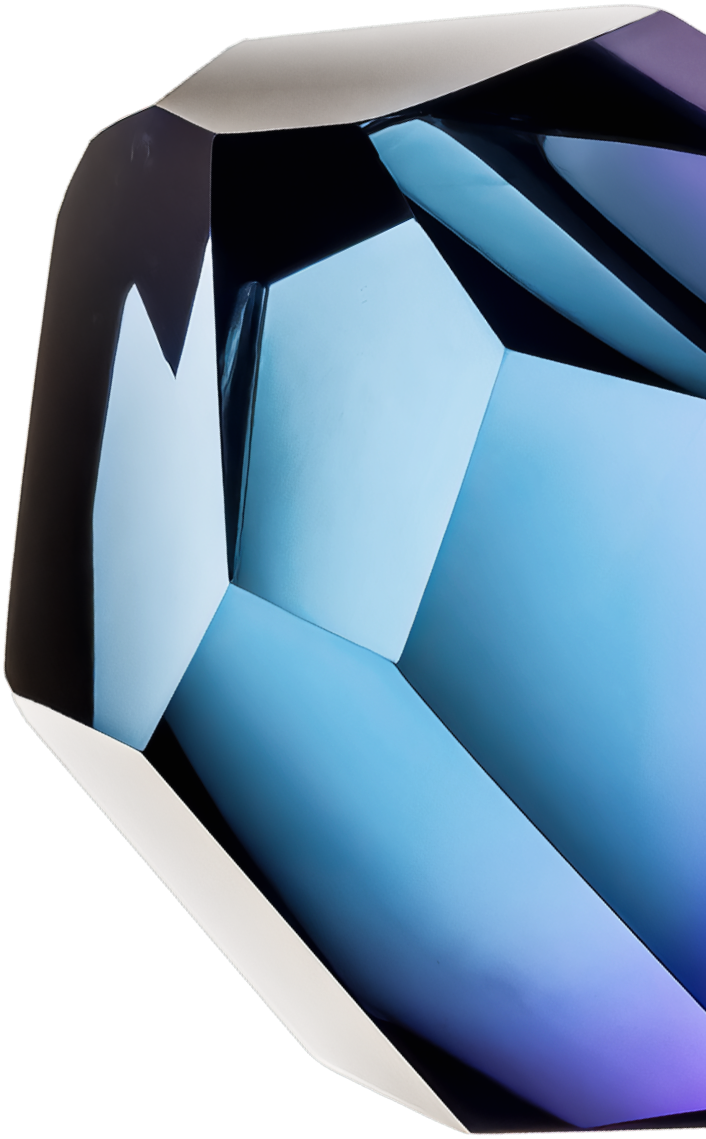
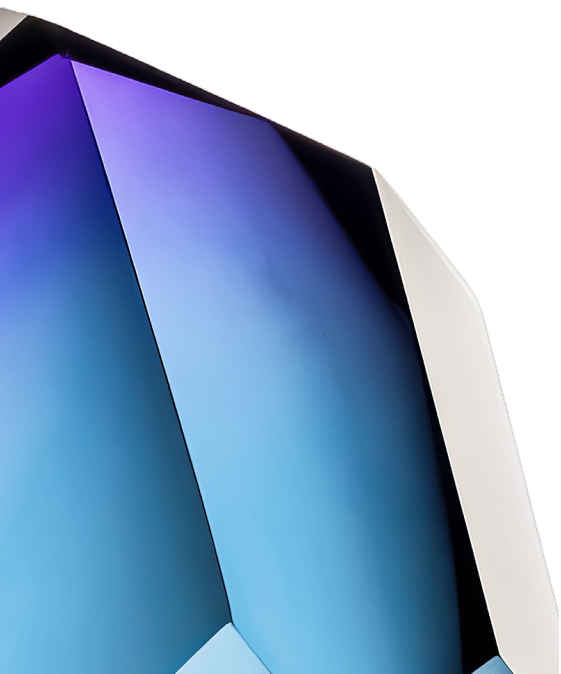


Table of contents



The future of AI is agentic.

Introduction

Generative AI agents mark a leap forward from traditional, standalone language models, offering a dynamic approach to problem-solving and interaction. As defined in the original Agents paper, an agent is an application engineered to achieve specific objectives by perceiving its environment and strategically acting upon it using the tools at its disposal. The fundamental principle of an agent lies in its synthesis of reasoning, logic, and access to external information, enabling it to perform tasks and make decisions beyond the inherent capabilities of the underlying model. These agents possess the capacity for autonomous operation, independently pursuing their goals and proactively determining subsequent actions, often without explicit instructions.

The architecture of an agent is composed of three essential elements that drive its behavior and decision-making:

- **Model:** Within the agent's framework, the term "model" pertains to the language model (LM) that functions as the central decision-making unit, employing instruction-based reasoning and logical frameworks. The model can vary from general-purpose to multimodal or fine-tuned, depending on the agent's specific requirements.
- **Tools:** Tools are critical for bridging the divide between the agent's internal capabilities and the external world, facilitating interaction with external data and services. These tools empower agents to access and process real-world information. Tools can include extensions, functions, and data stores. Extensions bridge the gap between an API and an agent, enabling agents to seamlessly execute APIs. Functions are self-contained modules of code that accomplish specific tasks. Data stores provide access to dynamic and up-to-date information, ensuring a model’s responses remain grounded in factuality and relevance.
- **Orchestration layer:** The orchestration layer is a cyclical process that dictates how the agent assimilates information, engages in internal reasoning, and leverages that reasoning to inform its subsequent action or decision. This layer is responsible for maintaining memory, state, reasoning, and planning. It employs prompt engineering frameworks to steer reasoning and planning, facilitating more effective interaction with the environment and task completion. Reasoning techniques such as ReAct, Chain-of-Thought (CoT), and Tree-of-Thoughts (ToT) can be applied within this layer.

Building on these foundational concepts, this companion paper is designed for developers and serves as a "102" guide to more advanced topics. It offers in-depth explorations of agent evaluation methodologies and practical applications of Google agent products for enhancing agent capabilities in solving complex, real-world problems.

While exploring these theoretical concepts, we'll examine how they manifest in real-world implementations, with a particular focus on automotive AI as a compelling case study. The automotive domain exemplifies the challenges and opportunities of multi-agent architectures in production environments. Modern vehicles demand conversational interfaces that function with or without connectivity, balance between on-device and cloud processing for both safety and user experience, and seamlessly coordinate specialized capabilities across navigation, media control, messaging, and vehicle systems. Through this automotive lens, we'll see how different coordination patterns -- hierarchical, collaborative, and peer-to-peer -- come together to create robust, responsive user experiences in environments with significant constraints. This case study illustrates the practical application of multi-agent systems that businesses across industries can adapt to their specific domains.

Anyone who has built with gen AI quickly realizes it’s easy to get from an idea to a proof-of-concept, but it can be quite difficult to ensure high quality results and get to production - gen AI agents are no exception. Quality and Reliability are the most cited concerns for deploying to production, and the “AgentOps” process is a solution to optimize agent building.

AgentOps

Over the past two years, the field of Generative AI (GenAI) has undergone significant changes, with enterprise customers focusing on how to operationalize related solutions. This has resulted in various terms describing the operationalization of GenAI, such as MLOps for GenAI, LLMOps, FMOps, and GenAIOps.

Agent and Operations (AgentOps) is a subcategory of GenAIOps that focuses on the efficient operationalization of Agents. Its main additional components include internal and external tool management, agent brain prompt (goal, profile, instructions) and orchestration, memory, and task decomposition.

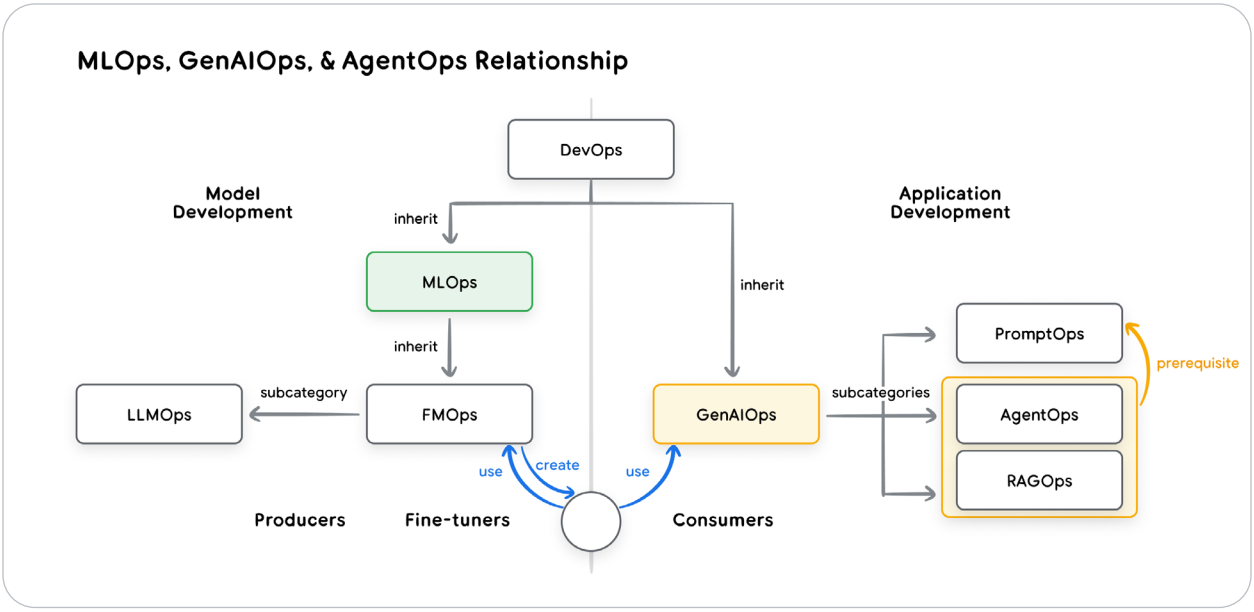


Figure 1. Relationship between DevOps, MLOps, and AgentOps.[13]

Each of these "Ops" requires capabilities like version control, automated deployments through CI/CD, testing, logging, security and (critically) metrics. Each system often implements some form of optimization based on metrics – measuring what your system is and isn't doing, measuring the outcomes and business metrics, and automating the processes for more holistic metrics, and incrementally improving step by step. This practice might be called "A/B experimentation" or "ML Ops" or "Metrics Driven development", but they derive from the same general approach and we will rely on those principles for AgentOps as well.

Remember that new practices don't replace the old. DevOps and MLOps best practices are still necessary for AgentOps, as they are dependencies. For example, Agent tool use, where APIs are invoked based on agent orchestration, often uses the same APIs you would

invoke with non-agentic software. Authentication and secret management, security, privacy, exception handling, throttling, quotas, and scalability are still critical and require careful API design in addition to Agent design.

Let's go ahead and define these "ops" terms to help distinguish between them:

- **Development and Operations (DevOps)** is the practice of efficiently productionizing deterministic software applications by integrating the elements of people, processes, and technology. DevOps serves as the foundation for all the following terms.
- **Machine Learning Operations (MLOps)** builds upon the capabilities of DevOps and concentrates on the efficient productionization of ML models. The primary distinction is that the output of an ML model is non-deterministic and relies on the input data (garbage in, garbage out).
- **Foundation Model Operations (FMOps)** expands upon the capabilities of MLOps and focuses on the efficient productionization of pre-trained (trained from scratch) or customized (fine-tuned) FMs.
- **Prompt and Operations (PromptOps)** is a subcategory of GenAIOps that focuses on operationalizing prompts effectively. Its main additional capabilities include prompt storage, lineage, metadata management (including evaluation scores), a centralized prompt template registry, and a prompt optimizer.
- **RAG and Operations (RAGOps)** is a subcategory of GenAIOps that centers on efficiently operationalizing RAG solutions. Its primary additional capabilities include the retrieval process through offline data preparation (encompassing cleaning, chunking, vectorization, similarity search, and re-ranking) and the generation process through prompt augmentation and grounding.

- **Agent and Operations (AgentOps)** is a subcategory of GenAIOps that focuses on the efficient operationalization of Agents. Its main additional components include internal and external tool management, agent brain prompt (goal, profile, instructions) and orchestration, memory, and task decomposition.

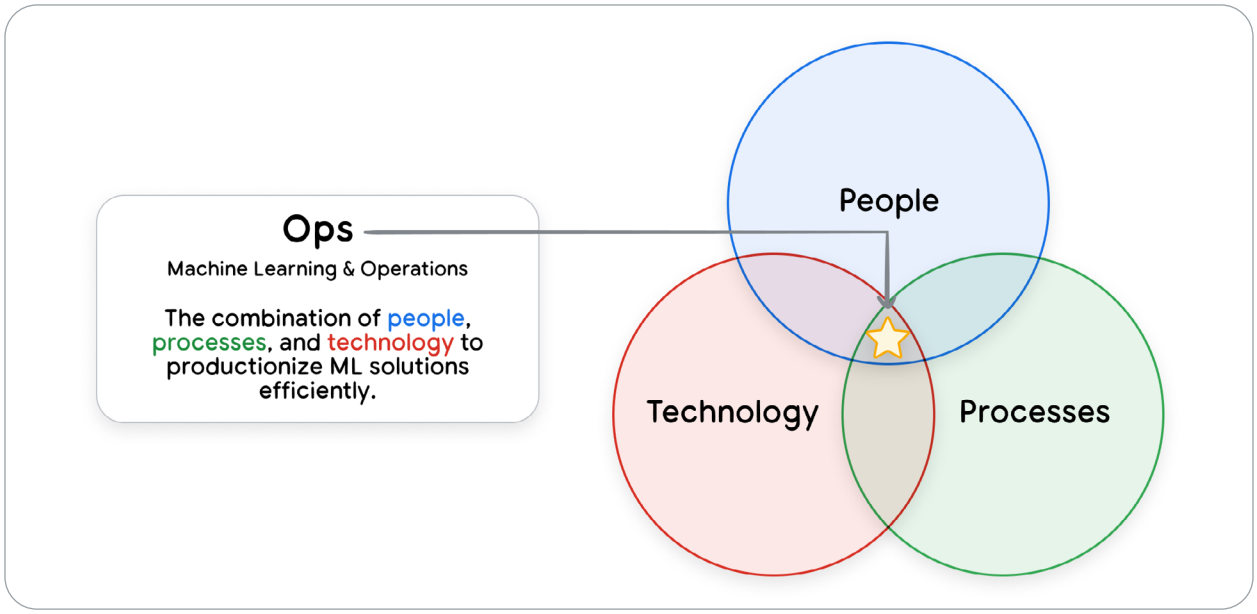


Figure 2. Each of these "Ops" are about technology, processes, and people[14]

All of these "Ops" are, in essence, the harmonious blend of people, processes, and technologies working together to efficiently deploy machine learning solutions into a live production environment. It's crucial to recognize that Ops extends beyond mere technology; it's not simply about constructing and delivering a ML pipeline. Successful Ops implementations delve deeper, considering the customer's operational model, their existing business units, and their overall organizational structure. This holistic approach ensures that the technology is tailored to their specific needs, seamlessly integrating into the business and maximizing value.

The next section will cover Agent Evaluation in detail, which is a significant part of the story for AgentOps and automation to capture useful metrics. Before we go there, let's start with a thought experiment; imagine setting up an A/B experiment in production for your new Agent. The treatment arm gets your new agent and the control arm does not. In that scenario, what metrics are you measuring to determine if the treatment arm is doing better? What metrics are you measuring to determine ROI for the project? Is it a goal being accomplished, or sales totals, or a set of critical steps in a user journey? Those metrics must be understood, instrumented and easily analyzed in addition to more detailed Agent Evaluation metrics.

Agent Success Metrics

Metrics are critical to building, monitoring, and comparing revisions of Agents. Business metrics, like revenue or user engagement, are probably outside of the scope of the agent itself but these should be the **north star metric** for your agents.

Most Agents are designed around accomplishing goals, so **goal completion rate** is a key metric to track. Similarly, a goal might be broken down into a few critical tasks or critical user interactions. Each of these critical tasks and interactions should be independently instrumented and measured.

So before we get into the details of the Agent itself, we already have several metrics identified which you should be able to easily track on a dashboard. Each business metric, goal, or critical interaction, will be aggregated in a familiar fashion: attempts, successes, rates, etc. Additionally, metrics you should be able to get from any application telemetry system are very important to track for agents as well, metrics like latency, errors, etc.

None of these metrics are specific to Agents, you could track them for any software, but they are even more important for Agent builders. Deterministic code does only what you tell it to do, whereas an agent can do a lot more, relying on LLMs which are trained on huge amounts of data. Instrumentation of these high level metrics is an important part of observability. You can think of them as Key Performance Indicators (KPI) for the agent, and they allow for observability in the aggregate, a higher level perspective of your agents.

Human feedback is one of the more critical metrics to track as well. A simple 👍👎 or user feedback form, within the context of an agent or task can go a long way to understanding where your agent does well and where it needs improvement. This feedback can come from end users of a consumer system, but also employees, QA testers, and process or domain experts reviewing the agent.

More detailed observability is also very important for agent building, being able to see and understand what the agent is doing and why it's doing that. An agent can be instrumented with "trace" to log all of the inner workings of the agent, not only the critically important tasks and user interactions. You *could* conceptually measure every internal step as metrics, but that is rarely done. Instead these detailed traces are used to debug an agent when metrics or manual testing show a problem, you can dig into details and see what went wrong.

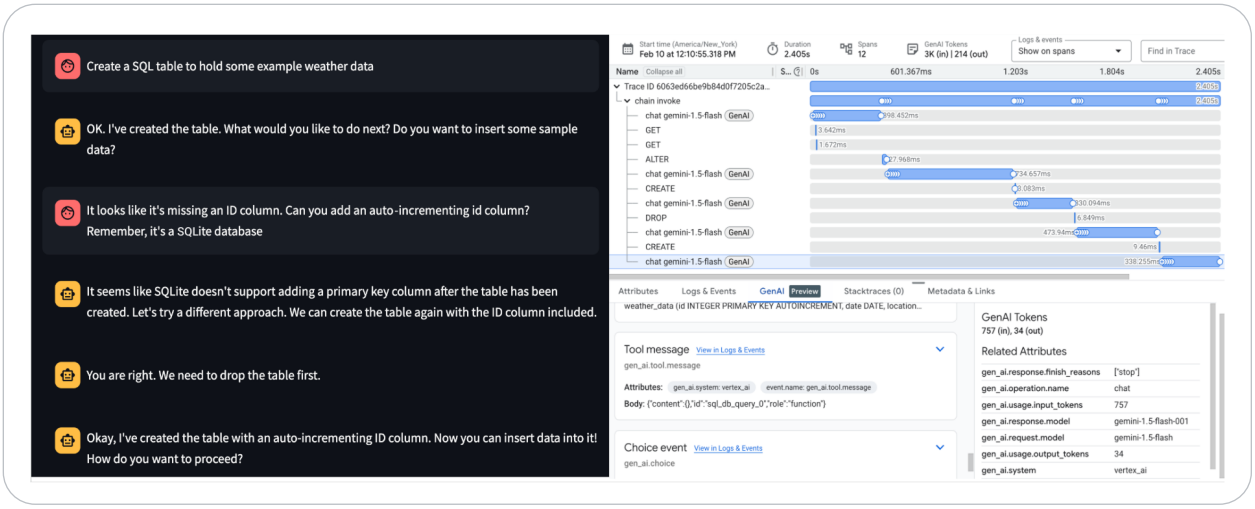


Figure 3: An example of Cloud Observability showing traces for an agent with tools and LLM OpenTelemetry spans.[15]

So far we've been talking about business metrics, goals, tasks, human feedback, and traces – those are all ways of understanding the actions and impact of your agents, in production. While developing an agent, in addition to manual testing, automated testing will be much more efficient in the long run and provide greater insights into the behavior of agents.

Agent Evaluation

To bridge the gap between a proof-of-concept and a production-ready AI agent, a robust and automated evaluation framework is essential. Unlike evaluating generative models, where the focus is primarily on the final output, agent evaluation requires a deeper understanding of the decision-making process. Agent evaluation can be broken down into three components that we discuss in this chapter:

1. **Assessing Agent Capabilities:** Evaluating an agent's core abilities, such as its capacity to understand instructions and reason logically.

2. **Evaluating Trajectory and Tool Use:** Analyzing the steps an agent takes to reach a solution, including its choice of tools, strategies, and the efficiency of its approach.
3. **Evaluating the Final Response:** Assessing the quality, relevance, and correctness of the agent's final output.

Assessing Agent Capabilities

Before evaluating your specific agentic use cases, publicly available benchmarks and technical reports can provide insight into core capabilities and limitations to consider when building out your agentic use cases. Public benchmarks exist for most fundamental agentic capabilities like model performance, hallucinations, tool calling and planning. For example, tool calling, the ability to select and use appropriate tools, is demonstrated by benchmarks like the Berkeley Function-Calling Leaderboard (BFCL)[16] and τ-bench[17] that also outlines common mistakes. Another example, PlanBench[18] aims to assess planning and reasoning, across several domains and specific capabilities.

But tool calling and planning is not the only capability you should consider. Agents inherit behaviors from their LLMs and each of their other components. Likewise, agent and user interactions have a history in traditional conversational design systems and workflow systems, and therefore can inherit the set of metrics and measurements that are used to determine the efficacy of these systems.