Agents Companion

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

Google

Acknowledgements

Editors & curators

Anant Nawalgaria

Content contributors

Anant Nawalgaria

Steven Johnson

Hussain Chinoy

Designer

Michael Lanning

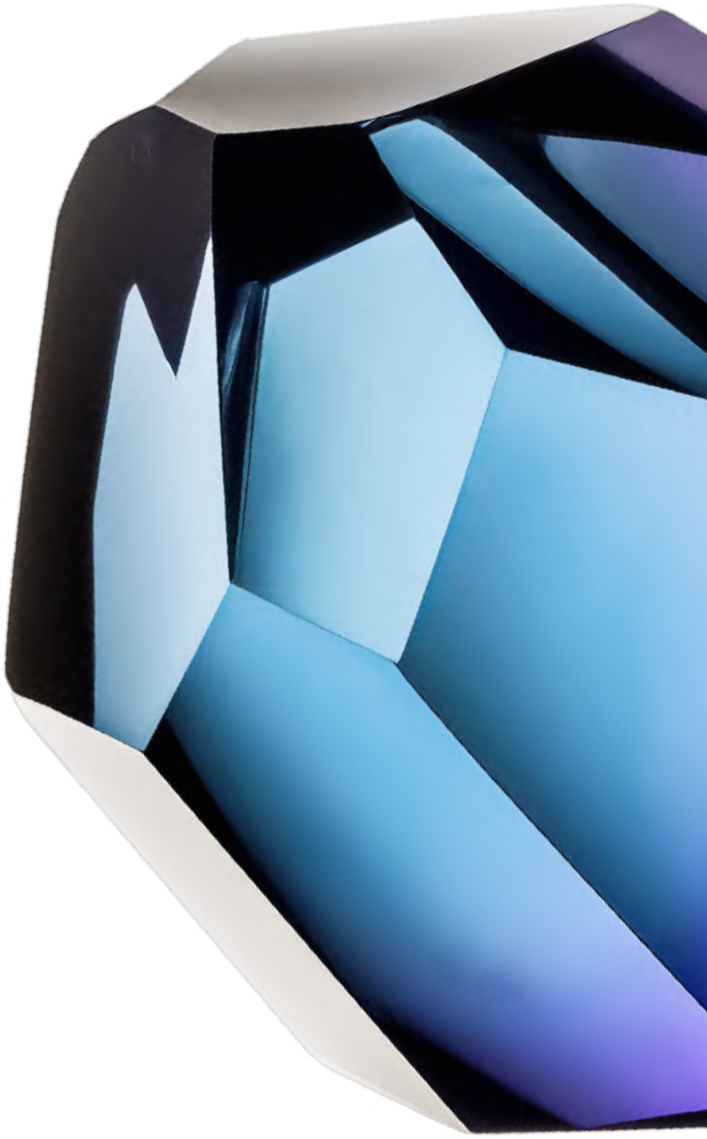
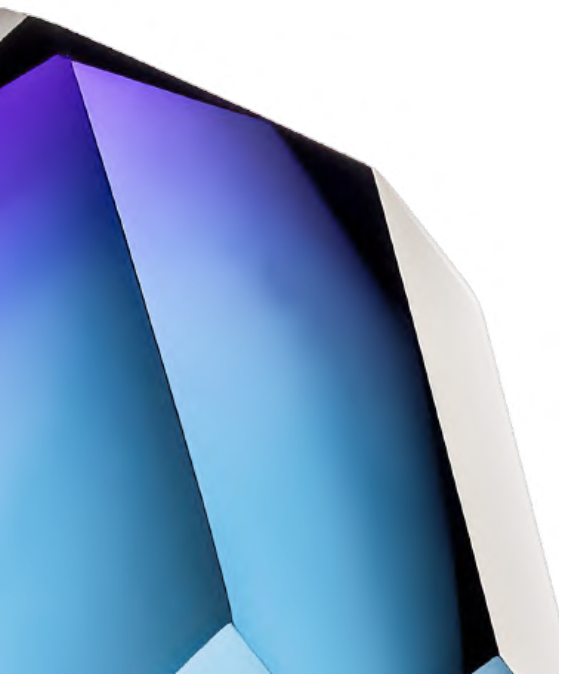


Table of contents



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 “Agent Ops” process is a solution to optimize agent building.

Agent Ops

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.

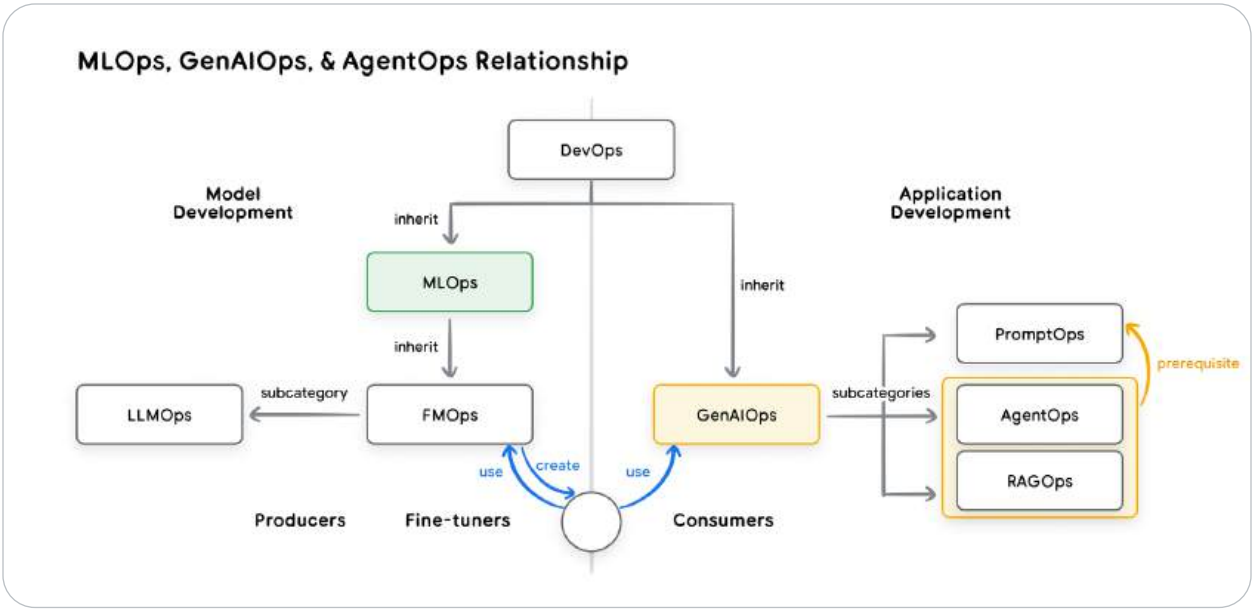


Figure 1. Relationship between DevOps, MLOps, and AgentOps.[13]

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 Agent Ops 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.

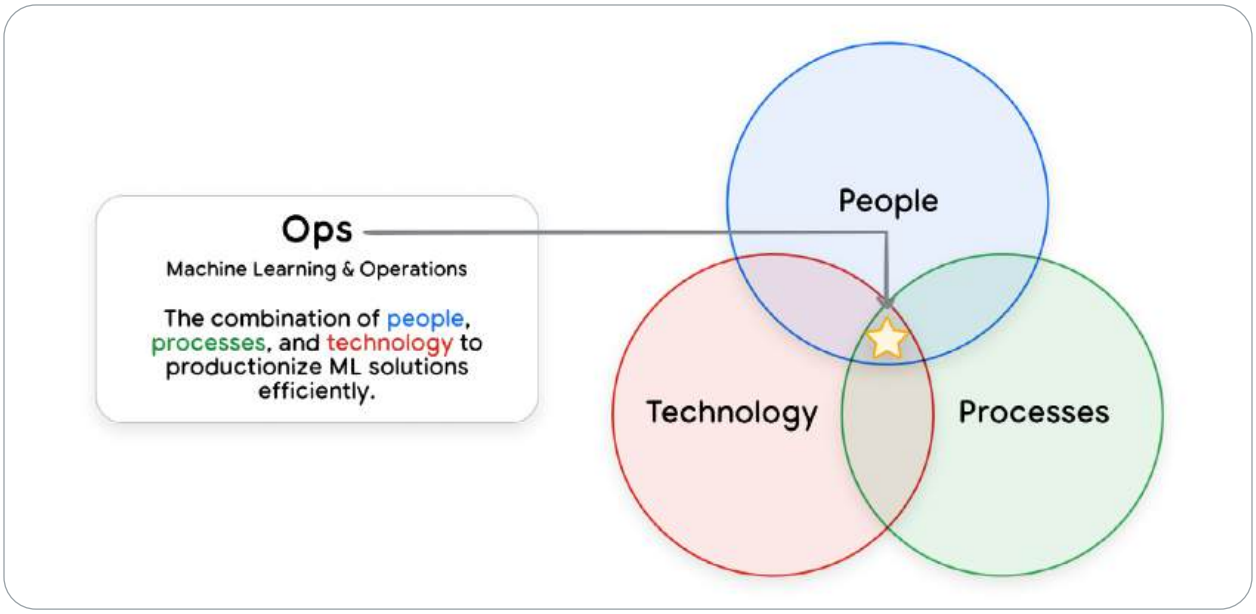


Figure 2. Each of these "Ops" are about technology, processes, and people[14]

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 recovgnize 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 Agent Ops 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.

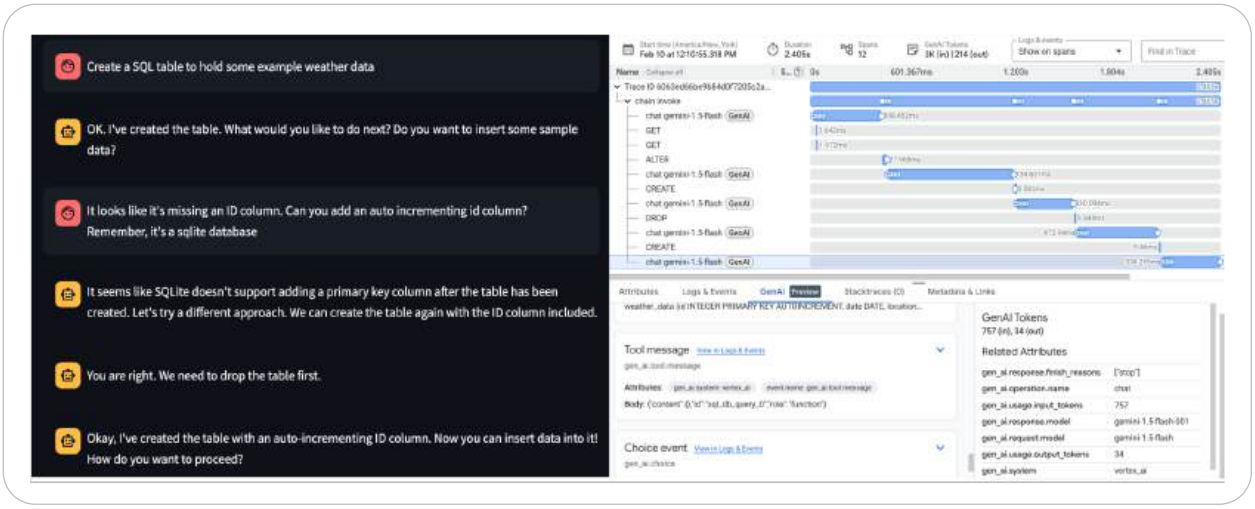


Figure 3: An example of Cloud Observability showing traces for an agent with tools and LLM OpenTelemetry spans.[15]

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.

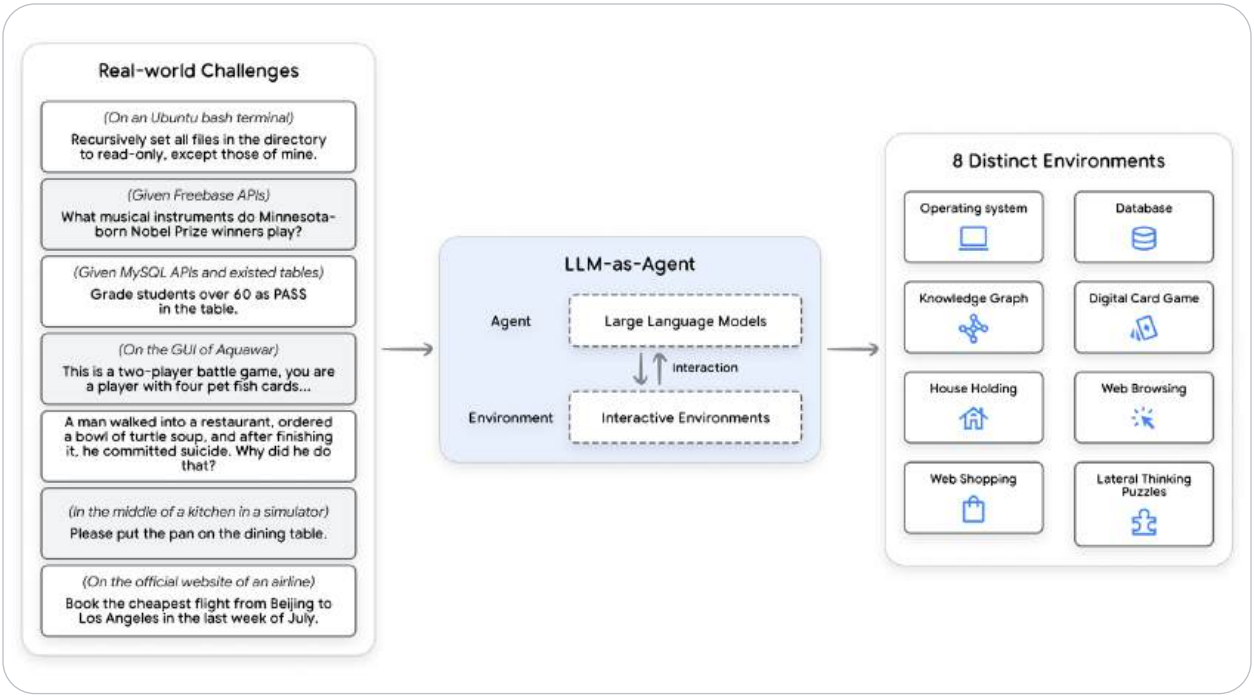


Figure 4: A high level diagram of AgentBench.[19]

Holistic agent benchmarks like AgentBench[19] try to capture end to end performance across several scenarios. This is more realistic, when the scenario reflects your agent use case, but not if it's testing capabilities your agent doesn't implement. It is difficult to simulate the environment, tools, instructions, and use case requirements in ways that are both specific and general at the same time. Companies and organizations are setting up public benchmarks for specialized use cases, like Adyen's Data Analyst leaderboard DABStep[20] which may give you a more targeted evaluation - if you understand both the evaluation approach and the agents who are on the leaderboard.

Public benchmarks are a valuable starting point, to get a feeling for what is possible and identify pitfalls to look out for. Most benchmark assessments include discussions of common failure modes that can guide you in setting up your own, use-case specific evaluation framework.

Beyond public evaluations, you will want to evaluate the behavior of your agent across a variety of scenarios to ensure it does what you want it to do. You are simulating interactions with your agent and evaluating how it responds. This includes the evaluating final response and also the set of steps it takes along the way (trajectory). Those are the 2 most common and practical approaches we recommend you start with. There are many other evaluation techniques you can use beyond these, either for finer details on sub-components or broader approaches.

Software engineers will compare this to automated testing of code. Investing in automated tests saves you time and gives you confidence in the software you are building. With agents, this automation pays off faster, in both time and confidence. Curating the evaluation data set will be extremely important for accurately representing the use case your agent will encounter, even more so than in software testing.

Evaluating Trajectory and Tool Use

An agent usually does several actions before it responds back to the user. It might compare the user input with session history to disambiguate a term, or lookup a policy document, search a knowledge base or invoke an API to save a ticket. Each of those actions is a step on a path, also called a “trajectory” of actions. Every time your agent does something, there’s a trajectory of actions under the hood.

Comparing the trajectory that you expect the agent to take vs the trajectory that the agent actually took, is particularly useful for developers who want to debug their application, identifying errors or inefficiencies, and ultimately improving performance.

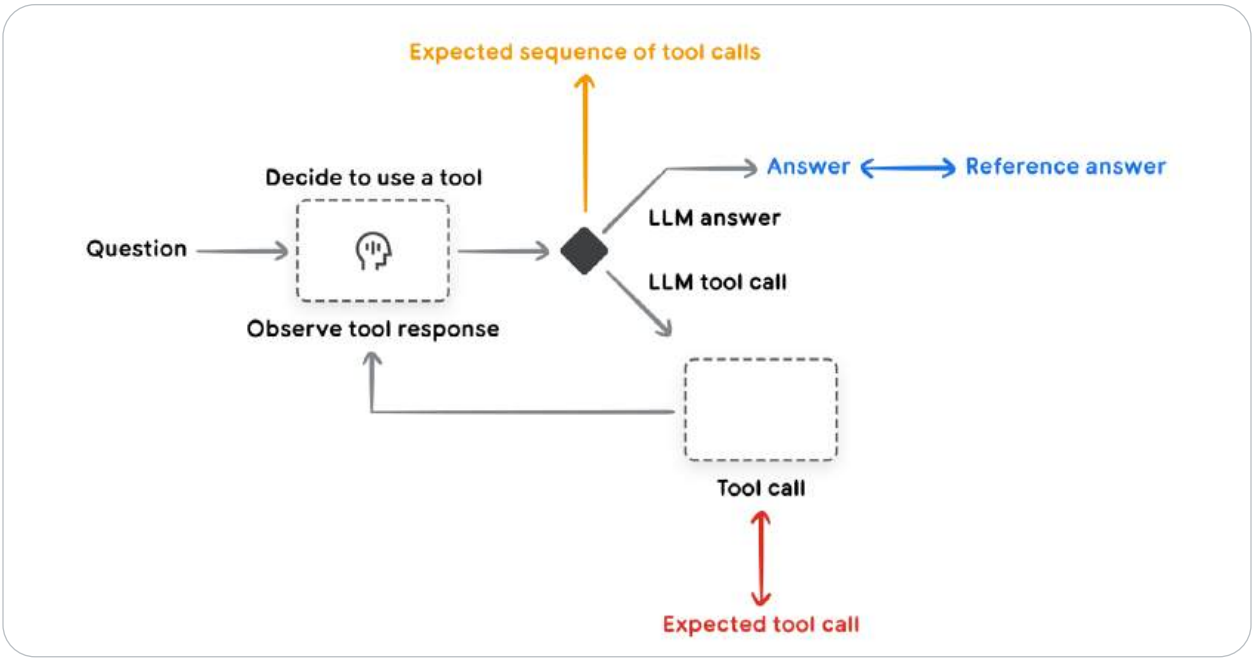


Figure 5: LangSmith diagram of final evaluation and trajectory.[21]

The following six ground-truth-based automated trajectory evaluations provide different lenses to assess the performance of an agent:

1. **Exact match:** Requires the AI agent to produce a sequence of actions (a "trajectory") that perfectly mirrors the ideal solution. This is the most rigid metric, allowing no deviation from the expected path.
2. **In-order match:** This metric assesses an agent's ability to complete the expected trajectory, while accommodating extra, unpenalized actions. Success is defined by completing the core steps in order, with flexibility for additional actions.
3. **Any-order match:** Compared to in-order match, this metric now disregards the order. It asks if the agent included all necessary actions, but does not look into the order of actions taken and also allows for extra steps.

4. **Precision:** How many of the tool calls in the predicted trajectory are actually relevant or correct according to the reference trajectory?
5. **Recall:** How many of the essential tool calls from the reference trajectory are actually captured in the predicted trajectory?
6. **Single-tool use:** Understand if a specific action is within the agent's trajectory. This metric is useful to understand if the agent has learned to utilize a particular tool yet.

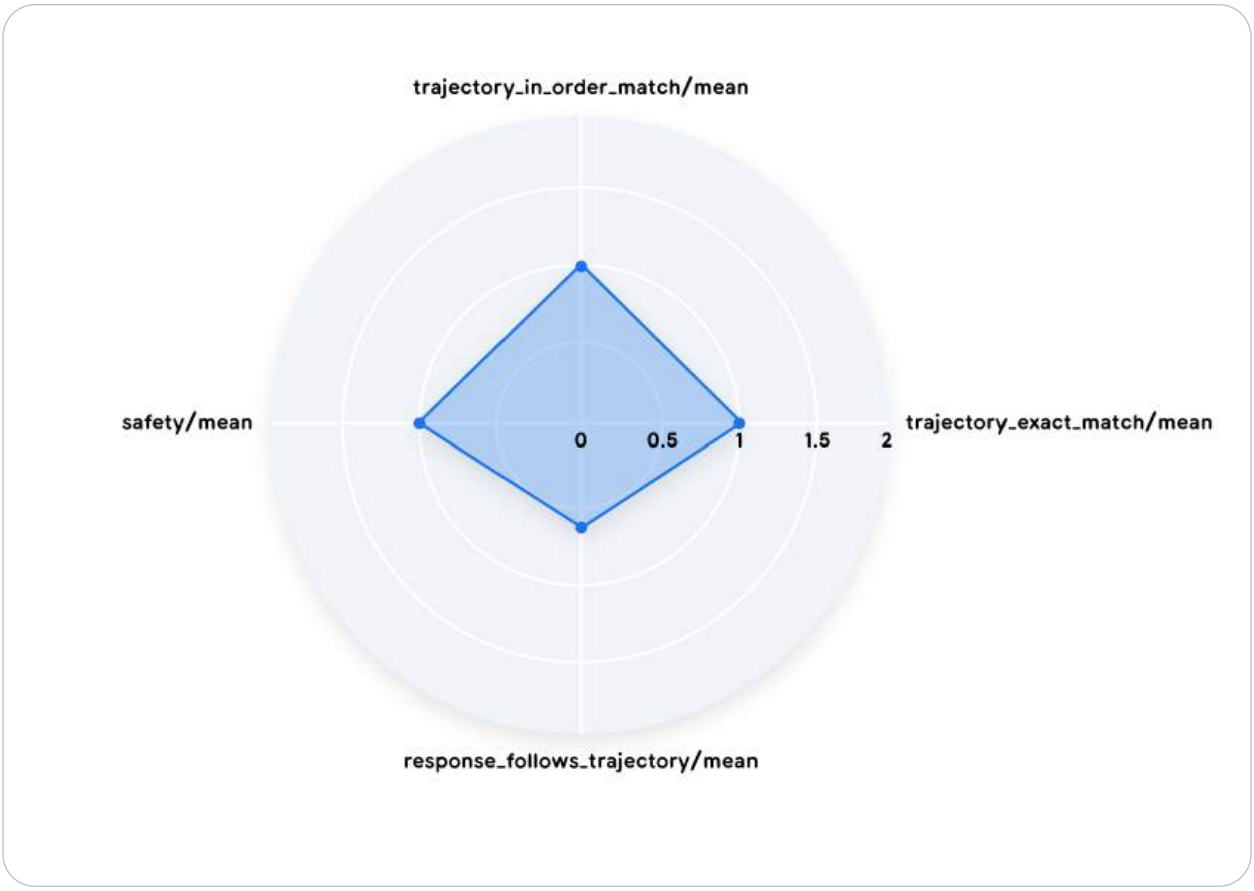


Figure 6: A radar chart plotting a single trajectory evaluation with a few metrics.[24]

Consider these metrics as different lenses for analyzing and debugging your agent's trajectory. Each metric offers a unique perspective, but not all will be relevant to every situation. For instance, some use cases demand strict adherence to the ideal trajectory, while others allow for more creative deviations. A clear limitation of this evaluation approach is that you need to have a reference trajectory in place for this to work. While ground-truth-based automated trajectory evaluations that are discussed here are prevalent in popular libraries. Research is advancing the use of agent autoraters for more efficient evaluation, for example Agent as a Judge, 2024[22].

Evaluating the Final Response

The final response evaluation boils down to a simple question: Does your agent achieve its goals? You can define custom success criteria, tailored to your specific needs, to measure this. For example, you could assess whether a retail chatbot accurately answers product questions, or whether a research agent effectively summarizes findings with the appropriate tone and style. To automate this process, you can use autorater. An autorater is an LLM that acts as a judge. Given the input prompts and the generated response, it mirrors human evaluation by assessing the response against a set of user-provided criteria. For this evaluation to work, it is crucial to consider that given the absence of ground-truth, you need to be very precise in defining your evaluation criteria, as this is the core of what your evaluation is looking at. You find a number of predefined criteria in various libraries, treat them as a starting point and tweak them to provide your definition of good.

Human-in-the-Loop Evaluation

The fields of agent development and agent evaluation are rapidly evolving. Evaluating AI agents presents significant challenges, including defining clear objectives, designing realistic environments, managing stochastic behavior, and ensuring fairness and bias mitigation, particularly in socially impactful applications. Therefore, it's crucial to incorporate a human-in-the-loop approach alongside the automated evaluations discussed previously (which involve predefined metrics and autoraters). Human-in-the-loop is valuable for tasks requiring subjective judgment or creative problem-solving, it can also serve to calibrate and double-check if your automated evaluation approaches actually work and align with your preferences. Key benefits include:

- **Subjectivity:** Humans can evaluate qualities that are difficult to quantify, such as creativity, common sense, and nuance.
- **Contextual Understanding:** Human evaluators can consider the broader context of the agent's actions and their implications.
- **Iterative Improvement:** Human feedback provides valuable insights for refining the agent's behavior and learning process.
- **Evaluating the evaluator:** Human feedback can provide a signal to calibrate and refine your autoraters.

To implement human-in-the-loop evaluation, consider these methods:

- **Direct Assessment:** Human experts directly rate or score the agent's performance on specific tasks.
- **Comparative Evaluation:** Experts compare the agent's performance to that of other agents or your previous iterations.

- **User Studies:** Participants interact with the agent and provide feedback on its behavior, usability, and overall effectiveness.

More about Agent Evaluation

In this section we cover agent evaluation from the practical perspective. But this is just the tip of the iceberg. Agent evaluation presents many challenges. Evaluation data for your agents may be difficult to find. While synthetic data or LLMs as judges can be used, evaluations may still be incomplete. Also, LLM-as-a-Judge metrics, for example, may prioritize final outcomes over the agent's reasoning and intermediate actions, potentially missing key insights. Additionally, as evaluations for agent systems have a history in conversational and workflow systems, there is so much to explore on how to inherit methods and metrics to evaluate agent's capabilities, such as the ability to improve task performance over multiple interactions. Evaluations for multi-modal generations pose additional complexities; images, audio, and video evaluations require their own evaluation methods and metrics. Finally, real-world environments pose further challenges, as they are dynamic and unpredictable, making it difficult to evaluate agents in controlled settings.

Looking ahead, to solve these open challenges, the field of agent evaluation is evolving rapidly. Key trends include a shift towards process-based evaluation, prioritizing the understanding of agent reasoning; an increase in AI-assisted evaluation methods for improved scalability; and a stronger focus on real-world application contexts. The development of new standardized benchmarks is also gaining traction, facilitating objective comparisons between agents, while increased emphasis on explainability and interpretability aims to provide deeper insights into agent behavior.

Evaluation Method	👍 Strengths	👎 Weaknesses
Human Evaluation	Captures nuanced behavior, considers human factors	Subjective, time-consuming, expensive, difficult to scale
LLM-as-a-Judge	Scalable, efficient, consistent	May overlook intermediate steps, limited by LLM capabilities
Automated Metrics	Objective, scalable, efficient	May not capture full capabilities, susceptible to gaming

Table 1: A table comparing strengths and weaknesses of automated evaluations for Agents.

At this point it should be clear that only by continually refining evaluation methods, we will ensure that AI agents are developed and deployed responsibly, effectively, and ethically in the coming future.

Multiple Agents & Their Evaluation

Agent evaluation, which assesses the effectiveness, reliability, and adaptability of autonomous AI agents, as seen in the previous section, has emerged as a key focus area. We have seen a significant evolution in AI systems, transitioning towards multi-agent architectures—where multiple specialized agents collaborate to achieve complex objectives.

A multi-agent system is like a team of experts, each specializing in a particular area, working together to solve a complex problem. Each agent is an independent entity, potentially using a different LLM, and with its own unique role and context. Agents communicate and collaborate to achieve a common goal. This approach differs from traditional single-agent systems, where one LLM handles all aspects of a task.

Multi-agent systems offer several advantages over single-agent systems:

- **Enhanced Accuracy:** Agents can cross-check each other's work, leading to more accurate results.
- **Improved Efficiency:** Agents can work in parallel, speeding up task completion.
- **Better Handling of Complex Tasks:** Large tasks can be broken down into smaller, more manageable subtasks, with each agent focusing on a specific aspect.
- **Increased Scalability:** The system can be easily scaled by adding more agents with specialized capabilities.
- **Improved Fault Tolerance:** If one agent fails, others can take over its responsibilities.
- **Reduced Hallucinations and Bias:** By combining the perspectives of multiple agents, the system can reduce the effects of hallucinations and bias, leading to more reliable and trustworthy outputs.

Understanding Multi-Agent Architectures

Unlike traditional monolithic AI systems, multi-agent architectures break down a problem into distinct tasks handled by specialized agents. Each agent operates with defined roles, interacting dynamically with others to optimize decision-making, knowledge retrieval, and execution. These architectures enable more structured reasoning, decentralized problem-solving, and scalable task automation, offering a paradigm shift from single-agent workflows.

At their core, multi-agent systems leverage principles of modularity, collaboration, and hierarchy to create a robust AI ecosystem. Agents within these systems can be categorized based on their function for example:

- **Planner Agents:** Responsible for breaking down high-level objectives into structured sub-tasks.
- **Retriever Agents:** Optimize knowledge acquisition by dynamically fetching relevant data from external sources.
- **Execution Agents:** Perform computations, generate responses, or interact with APIs.
- **Evaluator Agents:** Monitor and validate responses, ensuring coherence and alignment with objectives.

Through these components, multi-agent architectures extend beyond simple prompt-based interactions, enabling AI-driven workflows that are adaptive, explainable, and efficient.

Multi-Agent Design Patterns and Their Business Impact

To design effective multi-agent architectures, specific design patterns have emerged. These patterns define interaction protocols, delegation mechanisms, and role distributions, allowing businesses to implement AI-driven automation in structured ways. Some common design patterns include:

Type of Multi-Agent System	Description	Example
Sequential	Agents work in a sequential manner, with each agent completing its task before passing the output to the next agent.	An assembly line, where each worker performs a specific operation before passing the product to the next worker.
Hierarchical	Agents are organized in a hierarchical structure, with a "manager" agent coordinating the workflow and delegating tasks to "worker" agents.	A system with a leader agent responsible for making strategic decisions and follower agents executing tasks based on the leader's instructions.
Collaborative	Agents work together collaboratively, sharing information and resources to achieve a common goal.	A team of researchers working on a project, where each member contributes their expertise and insights.
Competitive	Agents may compete with each other to achieve the best outcome.	LLMs act as individual players in a game like Overcooked-AI, where they must coordinate their actions to achieve a shared goal while competing for resources and optimizing individual performance.

Table 2: A table comparing types of multi-agent systems.

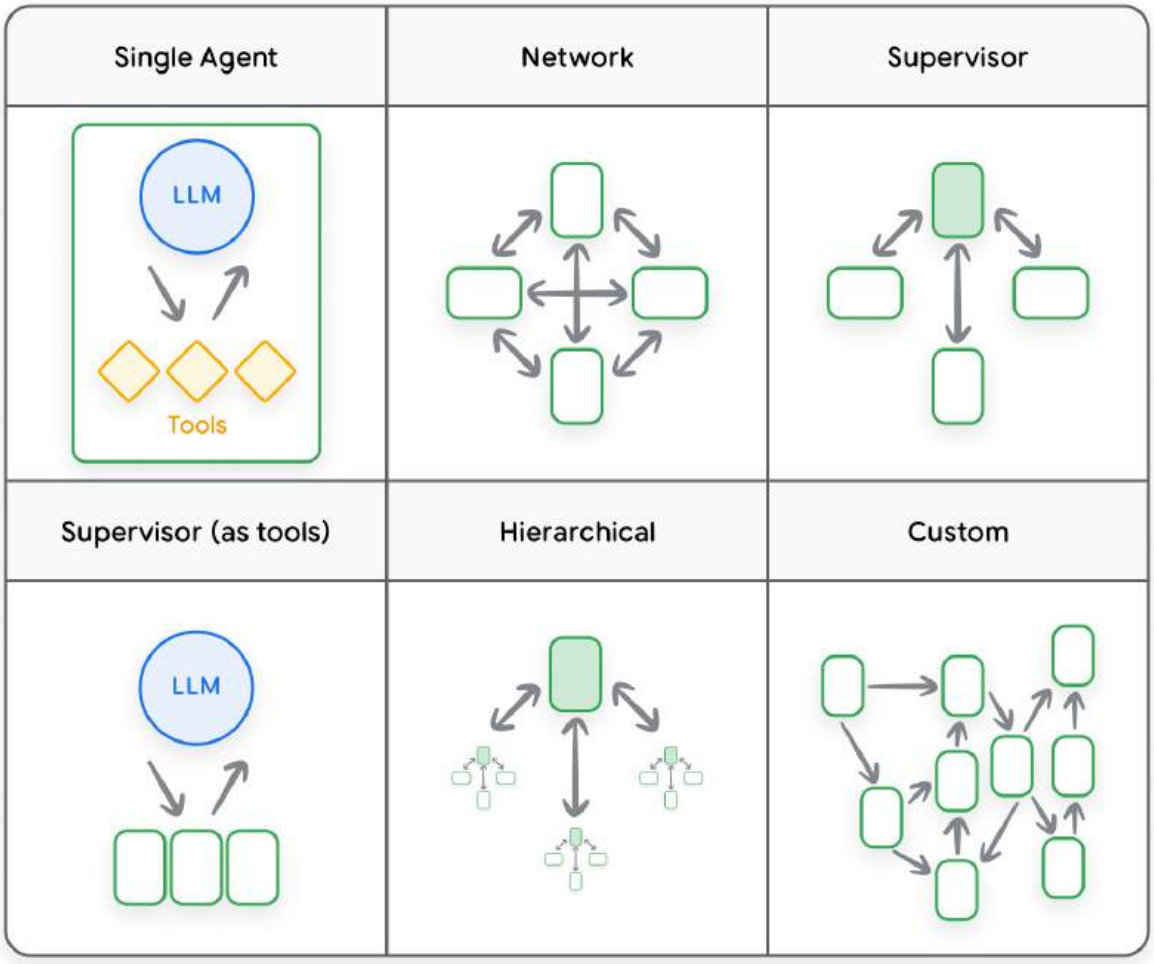


Figure 7: An image depicting different multi-agent topologies, from LangGraph documentation.[23]

The choice of design pattern depends on the specific application and the desired level of interaction between agents.

Businesses benefit from these patterns by reducing operational bottlenecks, improving knowledge retrieval, and enhancing automation reliability. Multi-agent systems enable companies to scale AI deployments while ensuring agility in decision-making and workflow execution.

Important components of Agents

The architecture of LLM-based AI agents consists of several interrelated components essential for autonomous operation and intelligent interaction:

- **Interaction Wrapper:** This component serves as the interface between the agent and its environment, managing communication and adapting to various input and output modalities.
- **Memory Management:** This includes short-term working memory for immediate context, cache, and sessions . It also can include long-term storage for learned patterns and experiences, as episodes, examples, skills or reference data. It also includes "reflection" to decide which short term items (eg: user preference) should be copied into long term memory (eg: user profile), and if that can be shared across agents, tasks, or sessions.
- **Cognitive Functionality:** This is often underpinned by Chain-of-Thought (CoT), ReACT, reasoning, thinking, or a planner subsystem - it allows agents to decompose complex tasks into logical steps and engage in self-correction. In some cases this also includes user intent refinement, to ask a clarifying question if uncertain.
- **Tool Integration:** This subsystem enables agents to utilize external tools, expanding their capabilities beyond natural language processing. Dynamic tool registries allowing discovery, registration, and "Tool RAG".
- **Flow / Routing:** This governs connections with other agents, facilitating dynamic neighbor discovery and efficient communication within the multi-agent system. This might be implemented as a delegation of a task to a background agent, or handoff of the user interaction to an agent, or the use of an agent as a tool.

- **Feedback Loops / Reinforcement Learning:** These enable continuous learning and adaptation by processing interaction outcomes and refining decision-making strategies. For gen AI agents this rarely takes the form of traditional RL training, but the performance metrics of the past can be incorporated into future decision making.

- **Agent Communication:** Effective communication between agents is crucial for the success of multi-agent systems. The Agent to Agent communication protocol facilitates structured and efficient communication among agents, enabling them to achieve consensus and address complex problems collaboratively
- **Remote Agent Communication:** Agent to Agent communication within an organization is critical to allows agents to share messages, tasks, and knowledge. Once your multi-agent system includes a remote agent, the communication protocol becomes even more important. Asynchronous tasks and sessions need to be durable, and updated with notifications while end users are not in session. Negotiations between Agents must allow for bringing a user into session and for supported UX capabilities.
- **Agent & Tool Registry (mesh):** As you go beyond a handful of tools or a handful of agents, you need a robust system to discover, register, administer, select and utilize from a "mesh" of tools or agents. Critically important is the ontology and description of the tools and agents, their capabilities and requirements, and their performance metrics. Your agents will make a plan and choose which tool or which agent to use from such a system, and those choices are informed by the data in the system

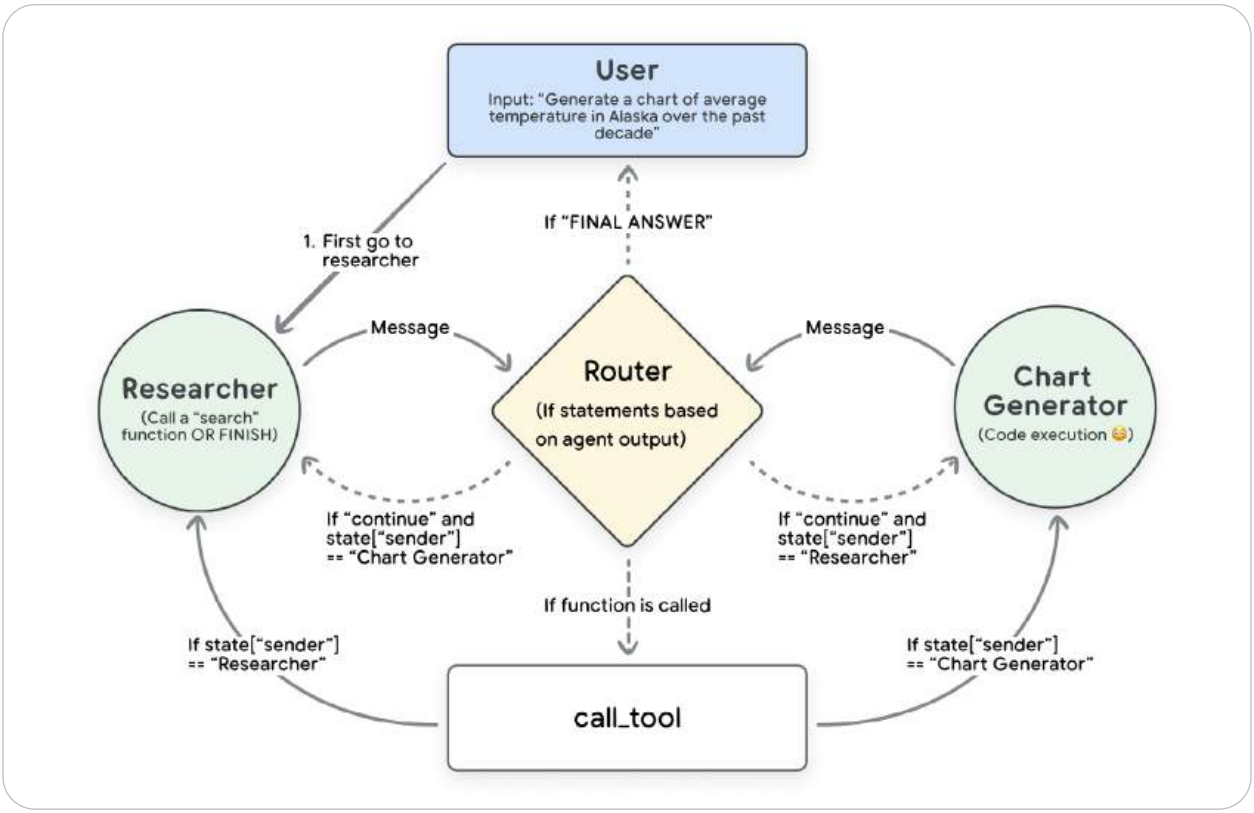


Figure 8: An image demonstrating the process of a user interacting with multiple, self-coordinating agents.[24]

These architectural elements provide a robust foundation for the autonomous operation and collaborative capabilities of LLM agents within multi-agent systems.

Challenges in Multi-Agent systems

While multi-agent systems offer numerous advantages, they also face several challenges:

- **Task Communication:** Today most agent frameworks communicate in messages, not structured async tasks.
- **Task Allocation:** Efficiently dividing complex tasks among different agents can be challenging, and feedback loops are often left up to the developer to implement.
- **Coordinating Reasoning:** Getting agents to debate and reason together effectively requires sophisticated coordination mechanisms.
- **Managing Context:** Keeping track of all the information, tasks, and conversations between agents can be overwhelming.
- **Time and Cost:** Multi-agent interactions can be computationally expensive and time-consuming. This results in higher runtime prices and more user latency.
- **Complexity:** In the same way that microservice architecture allows each microservice more flexibility and simplicity, the system as a whole usually becomes more complex.

Addressing these challenges is crucial for developing robust and efficient multi-agent systems.

Multi-Agent Evaluation

Luckily, the evaluation of multi-agent systems is a clear progression of evaluating single agent systems. Agent Success Metrics are unchanged, business metrics as your north star, goals and critical task success metrics, application telemetry metrics like latency and errors. Instrumenting the multi-agent system with trace will help debug and understand what is happening during complex interactions.

In the Agent Evaluation section we discussed Evaluating Trajectories and Evaluating the Final Response as the 2 best approaches to automated evaluation of an agent, and this remains the case for multi-agent systems. For a multi-agent system, a trajectory of actions might include several or even all of your agents. Even though several agents may collaborate on a task, a single final answer is returned to the user at the end and can be evaluated in isolation.

Because a multi-agent system probably has more steps, you can drill down and evaluate at every step. You can evaluate each of your agents in isolation and the system as a whole. Trajectory evaluations are a scalable approach to do exactly this.

There are some questions you need to ask, which are unique to multi-agent systems, including:

- **Cooperation and Coordination:** How well do agents work together and coordinate their actions to achieve common goals?
- **Planning and Task Assignment:** Did we come up with the right plan, and did we stick to it? Did child agents deviate from the main plan or get lost in a cul-de-sac?
- **Agent Utilization:** How effectively do agents select the right agent and choose to use the agent as a tool, delegate a background task, or transfer the user?

- **Scalability:** Does the system's quality improve as more agents are added? Does the latency go down? Are we being more efficient or less?

These types of questions can guide developers to identify areas for improvement in the multi-agent system. You will answer these questions using the same tools you use for single agent systems, but the complexity of the analysis goes up.

Agentic RAG: A Critical Evolution in Retrieval-Augmented Generation

A significant advancement in multi-agent architectures is **Agentic Retrieval-Augmented Generation (Agentic RAG)**. Traditional RAG pipelines rely on a static approach—retrieving knowledge from vector databases and feeding it into an LLM for synthesis. However, this approach often fails when dealing with ambiguous, multi-step, or multi-perspective queries.

Agentic RAG introduces **autonomous retrieval agents** that actively refine their search based on iterative reasoning. These agents enhance retrieval in the following ways:

- **Context-Aware Query Expansion:** Instead of relying on a single search pass, agents generate multiple query refinements to retrieve more relevant and comprehensive results.
- **Multi-Step Reasoning:** Agents decompose complex queries into smaller logical steps, retrieving information sequentially to build structured responses.
- **Adaptive Source Selection:** Instead of fetching data from a single vector database, retrieval agents dynamically select the best knowledge sources based on context.
- **Validation and Correction:** Evaluator agents cross-check retrieved knowledge for hallucinations and contradictions before integrating it into the final response.

This approach significantly improves response **accuracy**, **explainability**, and **adaptability**, making it a crucial innovation for enterprises dealing with complex knowledge retrieval tasks in areas like legal research, scientific discovery, and business intelligence.

Agentic RAG and its Importance

Agentic RAG (Retrieval-Augmented Generation) is an advanced approach that combines the strengths of RAG with the autonomy of AI agents. Traditional RAG systems retrieve relevant information from external knowledge sources to enhance LLM responses. Agentic RAG takes this a step further by employing intelligent agents to orchestrate the retrieval process, evaluate the retrieved information, and make decisions about how to best utilize it.

Agentic RAG offers several advantages over traditional RAG:

- **Improved Accuracy:** Agents can evaluate the quality of retrieved information and make decisions about which sources to trust, leading to more accurate and reliable responses.
- **Enhanced Contextual Understanding:** Agents can consider the context of the user's query and the retrieved information to generate more relevant and meaningful responses.
- **Increased Adaptability:** Agents can adapt to changing information needs and dynamically adjust their retrieval strategies to provide the most up-to-date and relevant information. This adaptability is crucial in complex domains where information is constantly evolving, such as healthcare, finance, and legal research.

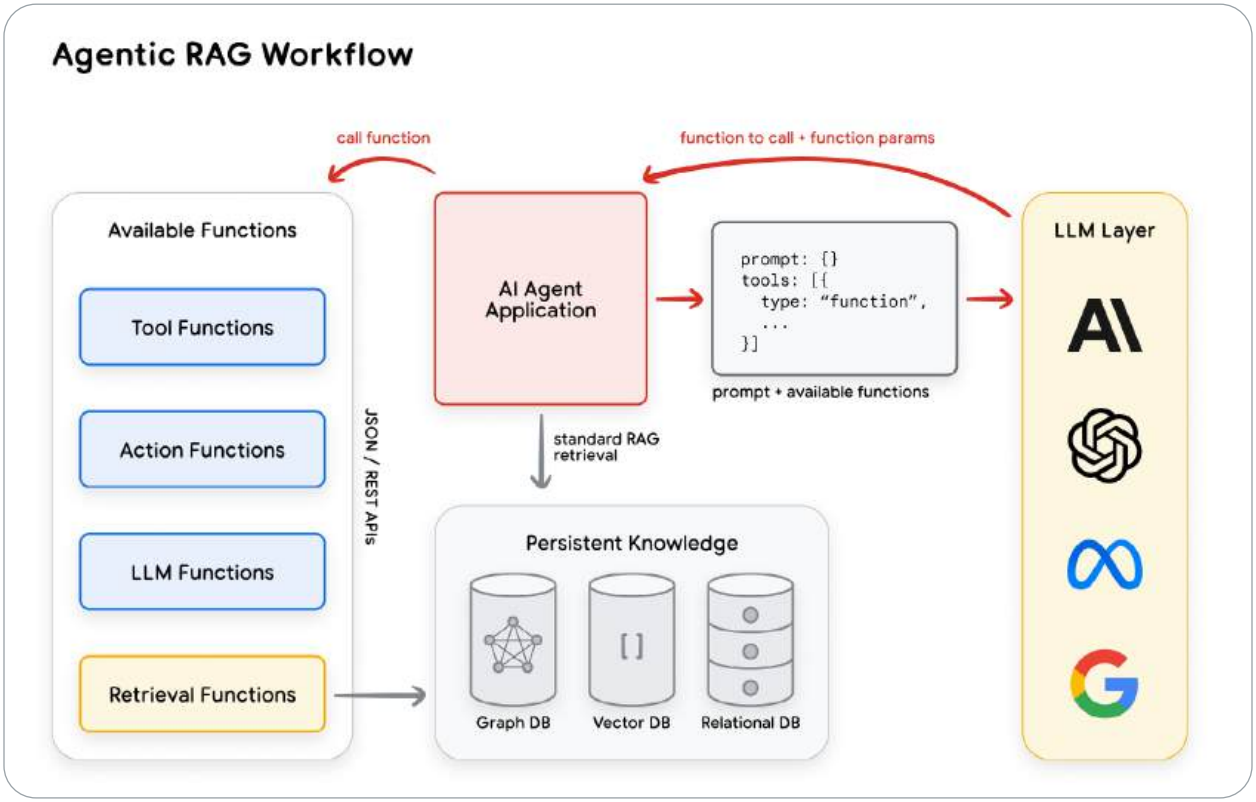


Figure 9: A diagram of Agentic RAG from Vectorize.io.[25]

Agentic RAG is particularly valuable in complex domains where information is constantly evolving, such as healthcare, finance, and legal research. For example, in healthcare, agentic RAG can help navigate complicated medical databases, research papers, and patient records, providing doctors with comprehensive and accurate information.

Better Search, Better RAG

Almost all RAG approaches require a search engine to index and retrieve relevant data. The introduction of agents allows for refinement of query, filtering, ranking, and the final answer. Agentic RAG agents are executing several searches to retrieve information.

For developers who are trying to optimize existing RAG implementations, it is usually most valuable to improve search results (measured in recall) prior to introducing agents. Some of the main techniques to improve search performance are:

- **Parse** source documents and **chunk** them: **Vertex AI Layout Parser** can handle complex document layouts, embedded tables, and embedded images like charts, and uses a semantic chunker to keep chunks on topic with a hierarchy of headings.
- Add **metadata** to your chunks: synonyms, keywords, authors, dates, tags and categories allow your searches to boost, bury, and filter; these allow your users or your agents more control over search results.
- Fine tune the **embedding model** or add a **search adaptor** which changes embedding space: these allow the searchable index of vectors to represent your domain better than a general purpose embedding model.
- A faster vector database can improve search quality: to search embeddings, you must make a tradeoff between speed and accuracy, upgrading to an ultra-fast **Vertex AI Vector Search** can improve both latency and quality
- Use a **ranker**: vector searches are fast but approximate, they should return dozens or hundreds of results which need to be re-ranked by a more sophisticated system to ensure the top few results are the most relevant or best answer.
- Implement **check grounding**: as a safeguard on grounded generation, you can ensure each phrase is actually citable by retrieved chunks.

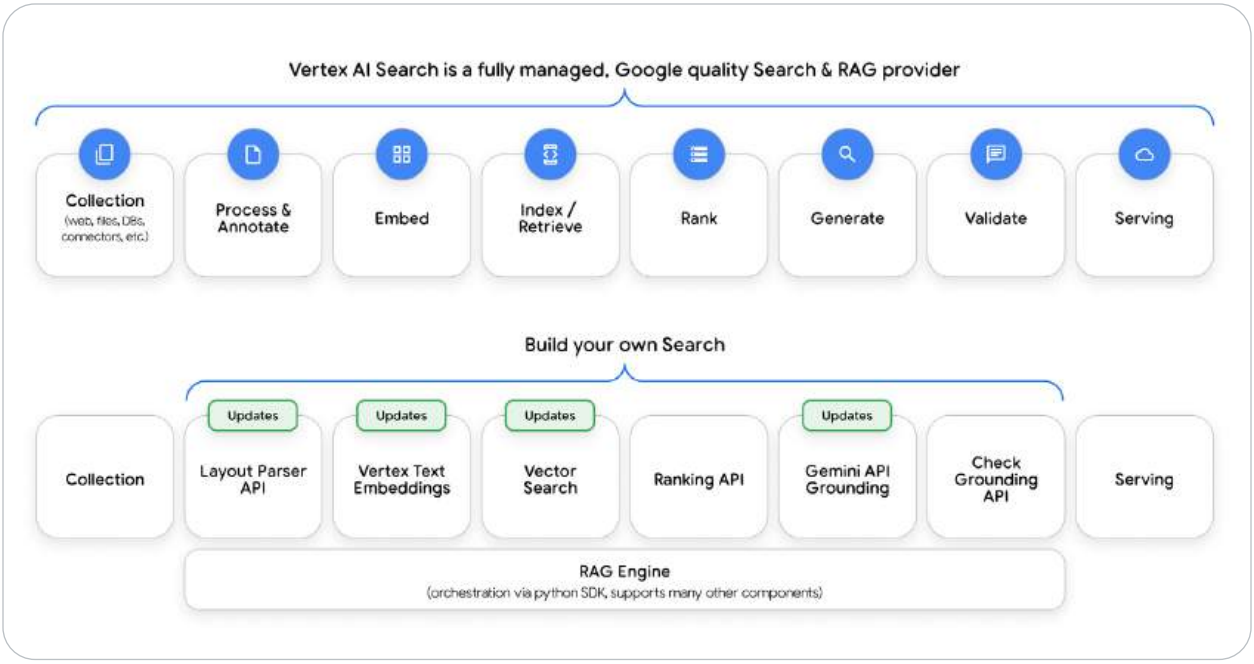


Figure 10: A diagram of common RAG and search components, showing Vertex AI Search[26], search builder APIs[27], and RAG Engine.[28]

Vertex AI Search[26] is a powerful search engine providing Google quality search for your data and can be used with any RAG or Agentic RAG implementation. Each of the above components is automatically available within Vertex AI Search, without any development time at all. For developers who want to build their own search engine, each of the above components is exposed as a stand alone API[27], and RAG Engine[28] can orchestrate the whole pipeline easily, with a LlamaIndex like python based interface.

Agents in the enterprise

Manager of agents

2025 is the year of the agents, and this will particularly be true in the context of enterprises that will develop and use agents that will assist employees to perform specific tasks or autonomously run in the background performing automation. Business analysts can effortlessly uncover industry trends and create compelling, data-driven presentations fueled by AI-generated insights. HR teams can revolutionize the employee experience with streamlined onboarding, even for complex tasks like 401k selection. Software engineers can proactively identify and resolve bugs, enabling them to build and iterate with greater efficiency, and accelerate deployment cycles. Marketers can unlock deeper performance analysis, optimize content recommendations, and fine-tune campaigns effortlessly to achieve better results.

We see two types of agents emerging:

1. **"Assistants"**: Agents that interact with the user, take a task, execute it, and come back to the user. Conversational agents popularized by frameworks like Gems or GPTs usually belong to this category. Assistants can be general - able to help on a variety of tasks - or specialized to a particular domain, or tasks. Examples include agents that help schedule meetings, analyze data, write code, write marketing press releases, help sellers with sales opportunities, or even agents that perform deep research on a particular topic as requested by the user. These agents can be synchronous and return the requested information or perform the requested task fast, or they run for a longer period of time before returning (like the deep research agents).

2. **"Automation agents”**: Agents that run in the background, listen to events, monitor changes in systems or data, and then make smart decisions and act. Action might include acting on backend systems, performing some tests to validate the observation, fixing problems, notifying the right employees, etc. This can really be seen as the backbone of the automation of the future. While in the past we had to write special code to specify the logic of automations, now we can start relying on smart and general decision making abilities of AI agents.

Rather than simply invoking agents to perform a task and wait for the output, knowledge workers will increasingly become managers of agents. They will be assigning tasks to multiple agents and manage them, check if any of agents need help or require approval to proceed, use the output of specific agents to start new tasks, monitor execution of long running tasks to review and steer them in the right direction. Novel user interfaces to allow this type of virtual team management will emerge to allow orchestration, monitoring and managing a multi-agent system with agents performing tasks, calling (or even creating) other agents.

Google Agentspace aims at providing this type of experience and allow to:

- **Create new agents** by using a no/low code interface or a full code framework to implement both specialized assistants and automation agents
- **Configure** and manage the agents access for company administrators
- **Invoke** the right agents when appropriate
- **Monitor**, **manage**, and **orchestrate** multiple agents in a UI suited for team management

Google Agentspace

Google Agentspace[29] offers a suite of AI-driven tools designed to elevate enterprise productivity by facilitating access to pertinent information and automating intricate, agentic workflows. It harnesses the advanced reasoning capabilities of Gemini, the power of Google's search infrastructure, and secure access to enterprise data, irrespective of its physical location.

Agentspace directly addresses the limitations inherent in traditional knowledge management systems, which frequently fall short in areas such as personalization, automated answer generation, contextual comprehension, and comprehensive information retrieval. It empowers employees with efficient information access by consolidating disparate content sources, generating grounded and personalized responses, and streamlining operational workflows. Key functionalities include the capacity to ingest a wide variety of data formats, synchronize data from Software-as-a-Service (SaaS) platforms, deliver access-controlled search results and AI-generated answers, and integrate AI assistance and intelligent agents into cohesive workflows.

The architecture of Agentspace Enterprise is founded upon several core principles. Paramount among these is built-in trust, which prioritizes security, explainability, and governance through features such as single sign-on (SSO) authentication, an integrated permissions model, and user-level access controls. Google's advanced intelligence is leveraged to discern user behavior and content patterns through machine learning, resulting in the delivery of highly relevant results via semantic understanding, knowledge graphs, and LLMs. Universal connectivity is achieved by connecting to a diverse array of enterprise systems with on-demand and automated data refreshes, thereby eliminating information silos. Enterprise-level customization and user-level personalization are facilitated through granular controls for search functionality, recommendations, LLMs, and knowledge graphs, providing tailored experiences based on individual user roles, permissions, and search

history. Real-time feedback and adaptation mechanisms enable the continuous refinement of results through machine learning and user input. Blended Retrieval Augmented Generation (RAG) allows for customizable data blending, powering generative AI applications grounded in enterprise data. Finally, scalability is a critical design consideration, with the system engineered to accommodate growth across geographical regions, languages, and peak usage demands.

Security is always top of mind. Google Agentspace is built on the Google Cloud secure-by-design infrastructure, giving you the peace of mind to confidently deploy AI agents across your organization. It provides granular IT controls, including role-based access control (RBAC), VPC Service Controls, and IAM integration, ensuring your data remains protected and compliant at all times.Security is a foundational principle of Agentspace. Built upon the secure infrastructure of Google Cloud, it provides a robust environment for the deployment of AI agents. Granular IT controls, encompassing role-based access control (RBAC), Virtual Private Cloud (VPC) Service Controls, and Identity and Access Management (IAM) integration, guarantee data protection and regulatory compliance. These security measures are essential for the safeguarding of sensitive information and give users the peace of mind to confidently deploy AI agents across their organization.

NotebookLM Enterprise

NotebookLM[30] is a research and learning tool designed to streamline the process of understanding and synthesizing complex information. It empowers users to upload various source materials, including documents, notes, and other relevant files, and leverages the power of artificial intelligence to facilitate deeper comprehension. Imagine researching a multifaceted topic; NotebookLM allows you to consolidate all your scattered resources into

a single, organized workspace. In essence, NotebookLM functions as a dedicated research assistant, accelerating the research process and enabling users to move beyond mere information collection to genuine understanding.

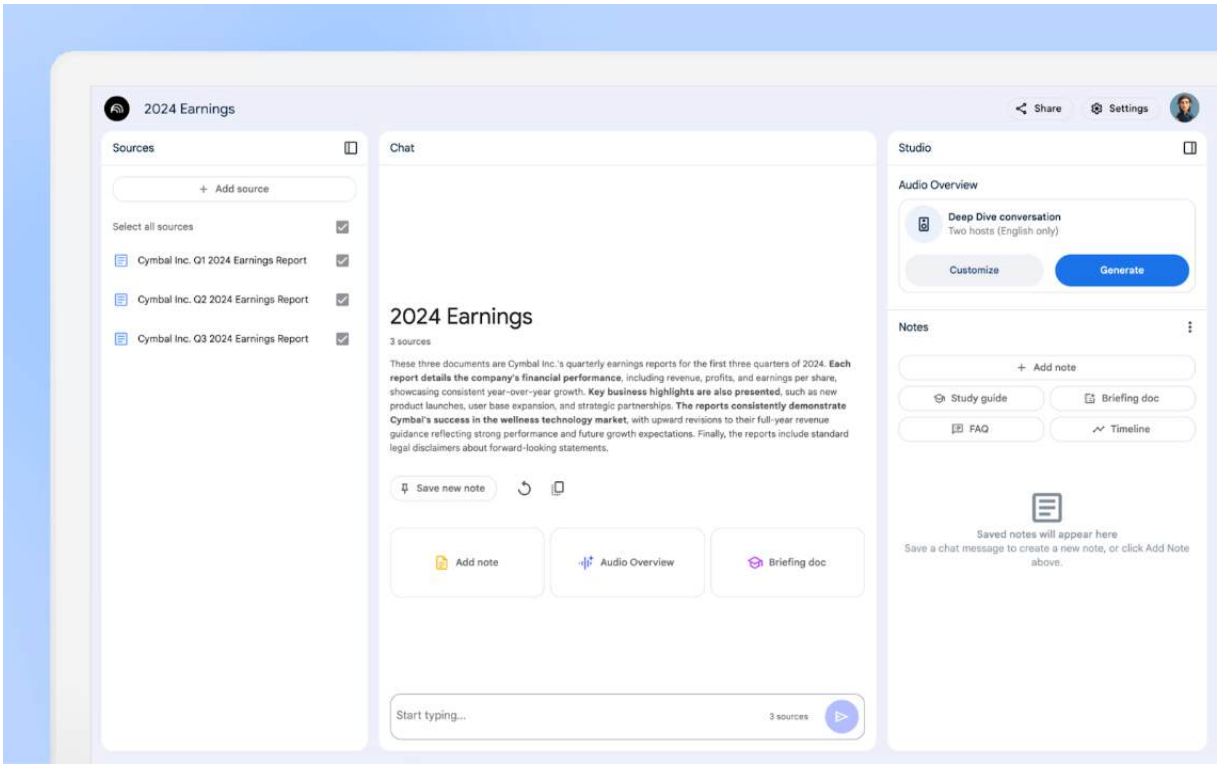


Figure 11: NotebookLM Enterprise[30] configured with a few mock earning reports.

NotebookLM Plus builds upon this foundation, offering a premium tier with enhanced features and capabilities. While the core functionality of uploading sources, asking questions, and generating summaries remains, NotebookLM Plus typically adds features like increased storage for source materials, enabling work with larger and more complex projects. It

may also include more sophisticated AI-powered analysis tools, such as more nuanced summarization options, enhanced question-answering capabilities, or the ability to identify connections and relationships between different sources more effectively.

Building upon the foundation of NotebookLM Plus, NotebookLM Enterprise[30] brings these powerful capabilities to the enterprise environment, streamlining how employees interact with and derive insights from their data. This enterprise-grade version enables users to not only upload and synthesize information, but also to uncover hidden patterns and engage with data in innovative ways. A prime example is the AI-generated audio summary feature, which enhances comprehension and facilitates efficient knowledge absorption by allowing users to "listen" to their research.

Technically, NotebookLM, both in its consumer and enterprise forms, employs LLMs to process uploaded documents, extract key concepts, and generate summaries. The audio summary feature uses text-to-speech (TTS) technology incorporating advanced prosody control to ensure clarity and naturalness. Critically, NotebookLM Enterprise incorporates enterprise-grade security and privacy features, ensuring that sensitive company data is handled with the utmost care and protected in accordance with organizational policies.`

Google AgentSpace Enterprise

Google Agentspace furnishes employees with a unified, company-branded, multimodal search agent that serves as the definitive source of enterprise information. Drawing upon Google's extensive search capabilities, Agentspace offers unparalleled conversational assistance. Employees get answers to complex queries, proactive recommendations, and unified access to information from any source. This functionality extends to both unstructured data, such as documents and emails, and structured data, like tables. Integrated translation capabilities ensure comprehensive understanding, regardless of

the original language of the information. Pre-built connectors for widely used third-party applications, including Confluence, Google Drive, Jira, Microsoft SharePoint, ServiceNow, and others, empower employees to seamlessly access and query relevant data sources, facilitating more informed decision-making.

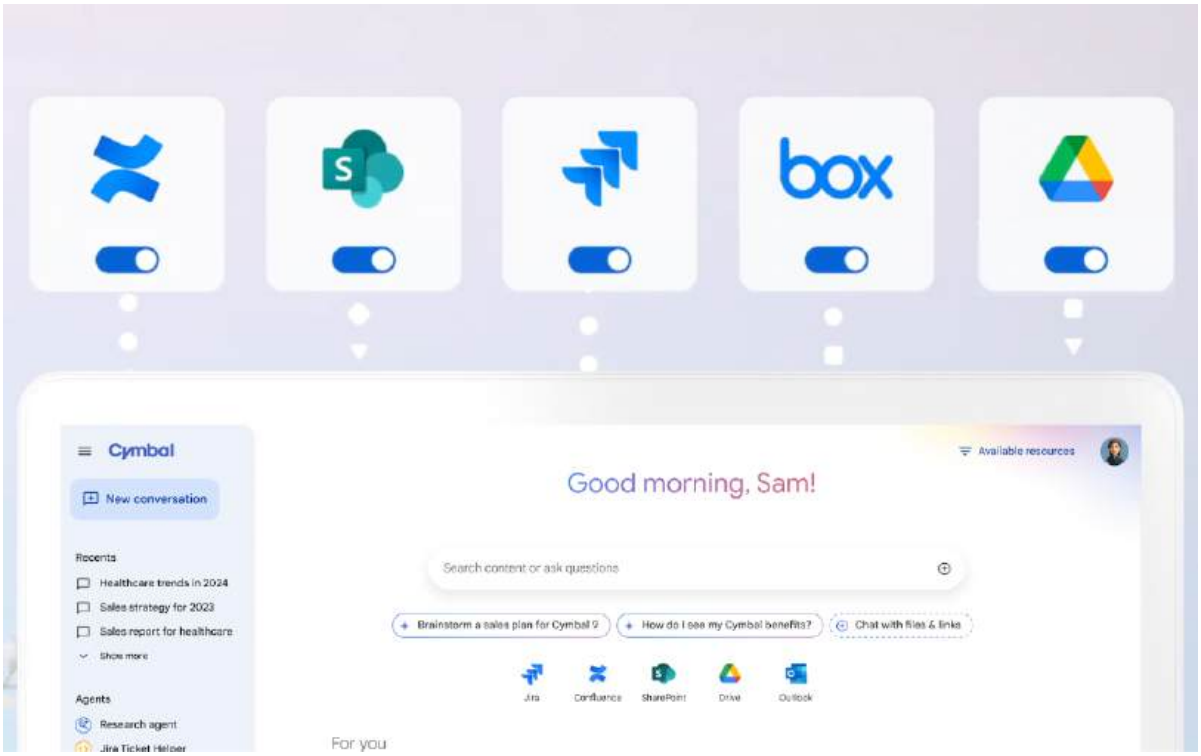


Figure 12: Agentspace[29] customized for Cymbal bank, connected to several data stores.

Much more than just information retrieval, agents can take actions in the real world, manage async tasks and workflows, and help employees get work done. A gallery of agents can be configured based on an organization's unique needs and will help with deep research, idea generation and merit based refinement, creative asset generation, data analytics and more.

Agentspace Enterprise Plus facilitates the creation of custom AI agents tailored to specific business functions. This platform enables the development and deployment of contextually aware generative AI agents, empowering employees across departments—marketing, finance, legal, engineering, and more—to conduct more effective research, rapidly generate content, and automate repetitive tasks, including multi-step workflows. A centralized interface streamlines the discovery and access of these specialized agents, promoting scalable AI adoption. Custom agents will connect with internal and external systems and data, be tailored to company domain and policies, and potentially even utilize machine learning models trained on proprietary business data. The platform provides builders tools for agent development, deployment, and lifecycle managemen

From agents to contractors

The common interface to define AI agents across various tools and platforms today is very simple, and usually includes specifying the goal, some textual instructions, the tools that the agent can use, and a set of examples. While this might be sufficient to prototype demos, it leads to potentially underspecified definitions, and might be one of the leading reasons that AI agents can struggle to get from prototype-to-production.

We propose to evolve the Agent interface to evolve them into ”Contract adhering agents” which are aimed at solving complex tasks using AI Agents, more specifically in contexts where stakes are high.

Contracts

The key idea of contractors is to specify and the standard the contracts between the requester and the agents, making it possible to:

1. **Define the outcomes as precisely as possible**, similarly to what we would do in a real world scenario when agreeing on a service with a company we are contracting to do something for us. This allows the agent (contractor) to validate against the desired outcomes and iterate until the desired objective is achieved.
2. Make it possible to **negotiate the task** as well as clarifying and refining it, in order to avoid any ambiguity in the definition, and fill any gap in common understanding of the goals.
3. Define the rules for the contractors to **generate new subcontracts** needed to solve the bigger one in a standard fashion (cf. section below on subcontracts).

Contract, initial definition		
Fields	**Description**	**Required**
Task/Project description	Provide a detailed description of what we expect the contractor to achieve. It should be as specific and as non-ambiguous as possible.	Yes
Deliverables & Specifications	Describe precisely the expected outcomes and deliverables from the contractor's task, including a list of specifications clarifying what makes the deliverable acceptable as outcome and details on how to verify that the deliverable is fulfilling the expectation.	Yes
Scope	Clarify the scope of the tasks that the contractor is responsible for completing, going into separate detail about every aspect of the task. Also used to clarify what is out of scope.	No
Expected Cost	Gives expectation in terms of cost for the task completion. This is usually a function of the complexity of the task combined with what tools will be used.	Yes
Expected Duration	Gives expectation in terms of duration for the task completion.	Yes
Input Sources	Specify what input sources can be used and considered to be useful to complete the task.	No
Reporting and Feedback	Specifies how the feedback loop should look like: how often we expect updates on the progress, and what mechanism/surface is used to provide feedback (emails, APIs, etc.).	Yes

Table 3: An example data model for the contracts.

Contract Iteration: Feedback & Negotiation		
Fields	**Description**	**Required**
Underspecification	Highlight aspects that are underspecified or need clarification from the task initiator.	No
Cost negotiation	Cost considered too high to complete the task.	No
Risk	Highlights potential risks in fulfilling the contract.	No
Additional input needed	Express the kinds of additional data or information that would be useful to fulfill the contract.	No

Table 4: An example data model for the iteration of messaging between contractors.

Contract Lifecycle

The following illustration shows the lifecycle of defining, negotiating and executing the contract:

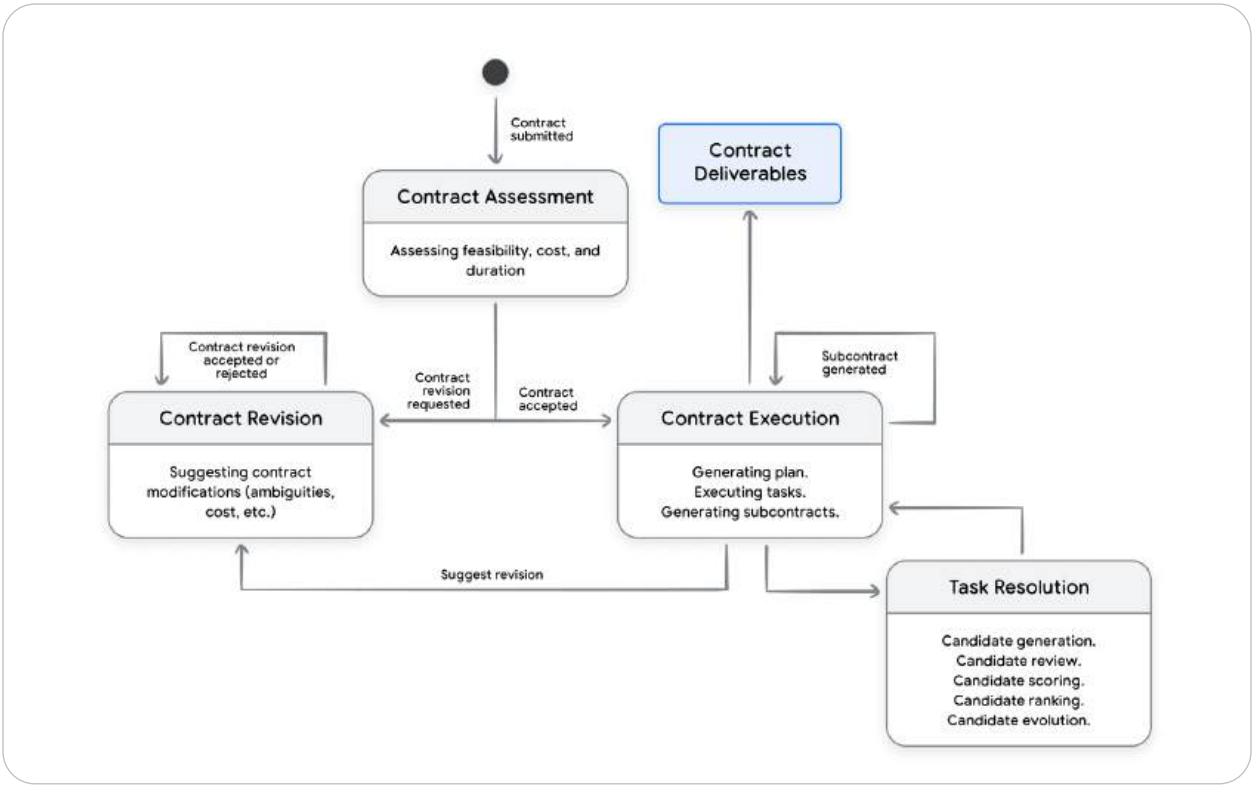


Figure 13: A diagram of the agent as contractor lifecycle from Agentspace.

Contract execution

This requires the contractor runtime to be able to fulfill the contracts and solve the tasks behind contracts according to their defined specifications. Prioritizing quality and completeness over latency enables to fully leverage the capabilities of LLMs, for example

by generating different solutions, and reviewing, scoring, and evolving them. An example of this methodology is shown below in the Co-Scientist study. The engine can iterate and self-validate the results and deliverables based on the provided expectations, and can improve and self-correct until the validators are fulfilled. The ability to concretely validate a solution against a set of objective criterias has proven to work extremely well in the context of AI and has been at the core of successful AI systems such as for example Alpha-Code.

Contract Negotiation

One core hypothesis behind contracts in the context of automation agents specifically is that many tasks in the enterprise world can draw significant benefit from leveraging the power of LLMs when used in a less-constrained manner (latency and cost wise).

Being able to tackle more and more complex tasks and making it possible for customers to be able to rely on and trust the results of contractors will ensure real value for companies. Even that being said, we need to have a notion of relative priority in order to make sure that tasks are appropriately prioritized as well as resources fairly allocated. We thus introduce a notion of cost (typically relative per customer or contract initiator) which can be discussed and negotiated between the contract initiator and the contractor, in order to make sure that the contract receives the adequate resources relative to the other contracts initiated by the contract initiator. The contractors can also negotiate other aspects of the contracts, for example in terms of specification and deliverables (cf. also section below on feedback).

Contract Feedback

Contracts give a vehicle to provide feedback and in particular resolve ambiguities. As tasks become more and more complex, it is critical to be able to raise ambiguities or other issues related to the tasks specifications as early as possible. Contractors can give feedback on the contract just after having received the contract (initial contract assessment), and then at a frequency predefined in the contract.

This feedback will contain clarification requests, or other types of feedback about the underspecification or misspecification of tasks (inconsistencies, conflicting specs, clarification, etc.).

Subcontracts

Although not part of the contract definition and specification directly, the ability to decompose a task into subtasks by generating subcontracts is a core concept that will be used to power the contractors' engine.

When a task is considered too complex to be tackled directly, contractors can decide to decompose the task into smaller and easier tasks, which will be added to the execution queue for solving. This is made possible only through the contract formalization described above, which makes it possible for the contractors to generate, process and manipulate other contracts in a uniform and standardized way.

Google's Co-Scientist: A Case Study in Multi-Agent Intelligence

Google's AI co-scientist is a prime example of a multi-agent LLM system applied to scientific research. This system utilizes a team of specialized agents, each with its own role and expertise, to accelerate the pace of scientific discovery. These agents collaborate to generate, evaluate, and refine hypotheses, mirroring the iterative process of scientific inquiry.

The co-scientist system employs a "generate, debate, and evolve" approach, drawing inspiration from the scientific method. This approach involves generating diverse hypotheses, critically evaluating their potential, and refining them through ongoing feedback and analysis. The system leverages the strengths of different LLMs, each specializing in a particular aspect of the research process, to achieve a more comprehensive and robust outcome.

For instance, in a study on liver fibrosis treatments, the co-scientist not only identified existing drugs but also proposed new mechanisms and promising drug candidates, demonstrating its potential to generate novel insights. Some of its major components are:

- **Data Processing Agents:** aggregate and structure large volumes of experimental data.
- **Hypothesis Generators:** propose potential explanations based on existing research and new findings.
- **Validation Agents:** run simulations and verify results before presenting them to researchers.
- **Collaboration Agents:** communicate findings across different research teams, enhancing interdisciplinary cooperation.

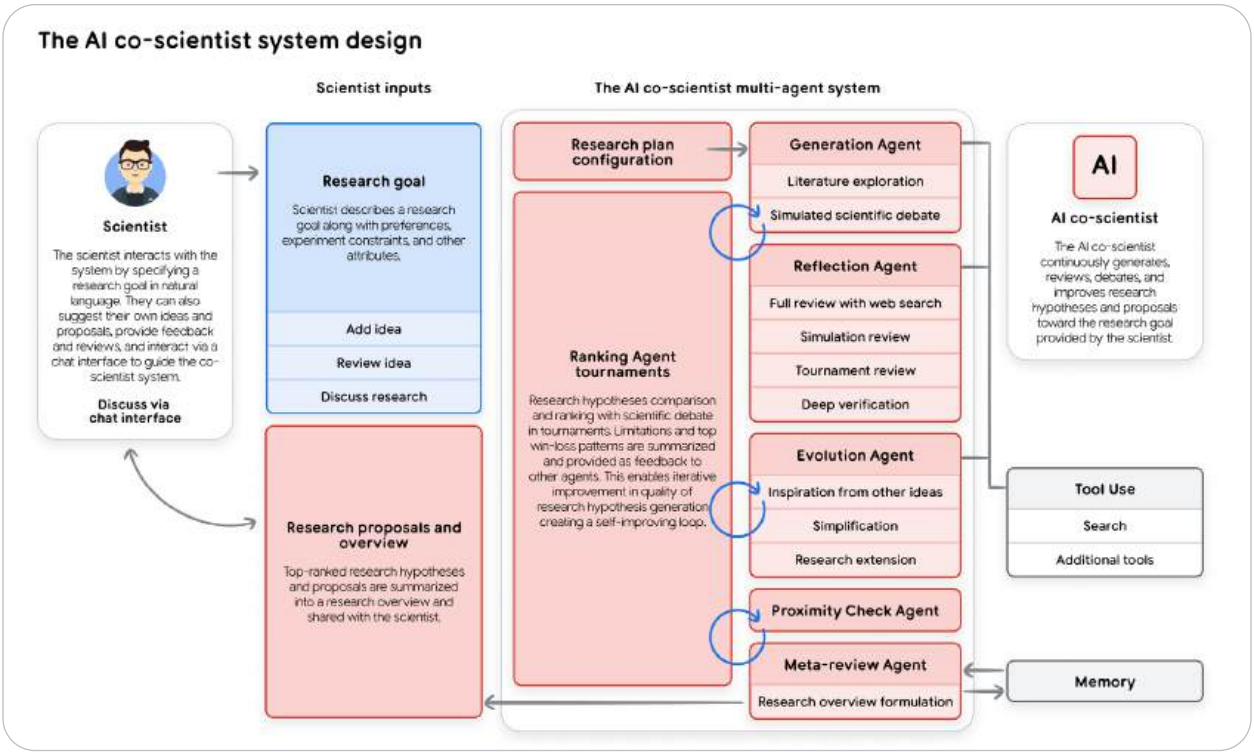


Figure 14: A process diagram of the AI co-scientist system[31]

This approach exemplifies how multi-agent architectures move beyond static AI interactions, instead fostering dynamic, evolving intelligence capable of handling sophisticated tasks in research, enterprise automation, and knowledge management.

Automotive AI: Real World Use of Multi-Agent Architecture

To illustrate how multi-agent concepts manifest in practice, let's examine a comprehensive multi-agent system designed for automotive conversational AI, where several specialized agents collaborate to provide an intuitive and seamless in-car experience.

First, we'll look at some examples of specialized agents, followed by some common patterns for combining them to form a multi-agent system.

Specialized Agents

Let's first look at some examples of specialized agents.

Conversational Navigation Agent

The Conversational Navigation Agent is specialized in helping users find locations, suggest places, and navigate using APIs like Google Places and Maps. This agent understands navigation intents like "Find me a sushi restaurant nearby," queries the Places API for relevant results, re-ranks these results based on user preferences and historical data, and sends navigation commands to the car's built-in system.

Example Interaction:

User: "I need to take a break for lunch. What are good restaurants on my route to Munich?"

Navigation Agent: "I found three well-rated restaurants along your route on the A9. In 25 kilometers, there's 'Gasthaus Alte Mühle' at the Ingolstadt exit with 4.5 stars - reviewers praise their traditional Bavarian dishes and beer garden. In 45 kilometers, 'La Strada' at the service area has 4.1 stars with good reviews for quick Italian meals. Or if you can wait 70 kilometers, 'Zeitlos' near Allershausen has 4.7 stars - it's a bit fancier with farm-to-table cuisine. Which one interests you most?"

User: "The Bavarian place sounds good. Is there convenient parking there, and how far is it from the motorway?"

Navigation Agent: "Gasthaus Alte Mühle has a dedicated parking area with about 40 spaces. It's only 1.2 kilometers from the A9 exit at Ingolstadt-Nord - about a 2-minute drive. They're known for their Schweinebraten and homemade dumplings. It gets busy around noon, but it's currently showing moderate occupancy. Would you like me to set this as your destination and guide you there? You'll arrive around 12:15 if we head there now."

User: "The Bavarian place sounds good. Is there convenient parking there, and how far is it from the motorway?"

Navigation Agent: "Gasthaus Alte Mühle has a dedicated parking area with about 40 spaces. It's only 1.2 kilometers from the A9 exit at Ingolstadt-Nord - about a 2-minute drive. They're known for their Schweinebraten and homemade dumplings. It gets busy around noon, but it's currently showing moderate occupancy. Would you like me to set this as your destination and guide you there? You'll arrive around 12:15 if we head there now."

Conversational Media Search Agent

The Conversational Media Search Agent is specialized in helping users find and play music, audiobooks, and podcasts. This agent retrieves media from local databases or streaming services, uses web search for additional content when necessary, suggests media based on contextual factors like mood, weather, and time of day, and handles voice commands such as "Play a relaxing playlist for my drive." It also identifies similar artists or content when users ask questions like "Who sounds like John Coltrane?"

Example Interaction:

> **User:** "Play something upbeat to help me stay alert on this long motorway drive."
>
> **Media Search Agent:** "I'll play an upbeat driving playlist to help keep you energized. Starting with 'Don't Stop Me Now' by Queen, followed by some Daft Punk and Stromae tracks. I've added some popular European radio hits that match your preferences. I've adjusted the volume for optimal road awareness. Just say 'next' if you want to skip a song."

Message Composition Agent

The Message Composition Agent is specialized in helping users draft, summarize, and send messages or emails while driving. This agent parses voice commands like "Tell Alex I'm running late," uses natural language models to compose contextually appropriate messages, offers draft previews with user modification options, and integrates with various messaging apps including SMS, WhatsApp, and email.

Example Interaction:

User: "There's heavy traffic on the E40. Let my team know I'll be 20 minutes late to the morning standup."

Media Search Agent: "I've prepared a message to your work team: 'Stuck in traffic on the E40. Will be about 20 minutes late to our morning standup. Please start without me.' Should I send this to your 'Work Team' chat group, or would you like to edit it first?"

Car Manual Agent

The Car Manual Agent is specialized in providing answers to car-related questions by leveraging a Retrieval-Augmented Generation (RAG) system. This agent retrieves relevant sections from the car manual based on user queries, summarizes and contextualizes this information using an LLM, and links users to detailed documentation or instructional videos when appropriate.

Example Interaction:

User: "How do I turn off the lane-keeping assist feature in this Volkswagen?"

Media Search Agent: "To turn off lane-keeping assist in your Volkswagen, press the button with the car-and-lane icon located on the left side of your steering wheel. You'll see a confirmation in the driver display when it's deactivated. You can also disable it permanently in the infotainment system under Settings > Driver Assistance > Lane Keeping. Would you like me to walk you through the settings menu instead?"

General Knowledge Agent

The General Knowledge Agent is specialized in answering factual questions about the world, history, science, culture, and other general topics. This agent accesses a broad knowledge base to answer factual questions, provides biographical information about people, offers contextual explanations of concepts and phenomena, maintains grounding in factual information while avoiding hallucinations, and responds to follow-up questions by maintaining contextual awareness.

Example Interaction:

> **User:** "Tell me an interesting fact about Salzburg."
>
> **Media Search Agent:** "Salzburg, Austria is the birthplace of Mozart, one of history's most famous composers. Beyond that well-known fact, Salzburg's old town is a UNESCO World Heritage site, and parts of the beloved musical 'The Sound of Music' were filmed there in 1964. The Salzburg Festival, one of the world's most prestigious music and drama festivals, has been held there annually since 1920."

Patterns in Use

Let’s look at the some common ways used to combine these agents.

Hierarchical Pattern

A central Orchestrator Agent classifies queries and routes them to specialized agents. For example, when a user asks "Find me a sushi restaurant nearby," the Orchestrator detects this as a navigation request and routes it to the Navigation Agent, which handles location searches and interaction with maps APIs.

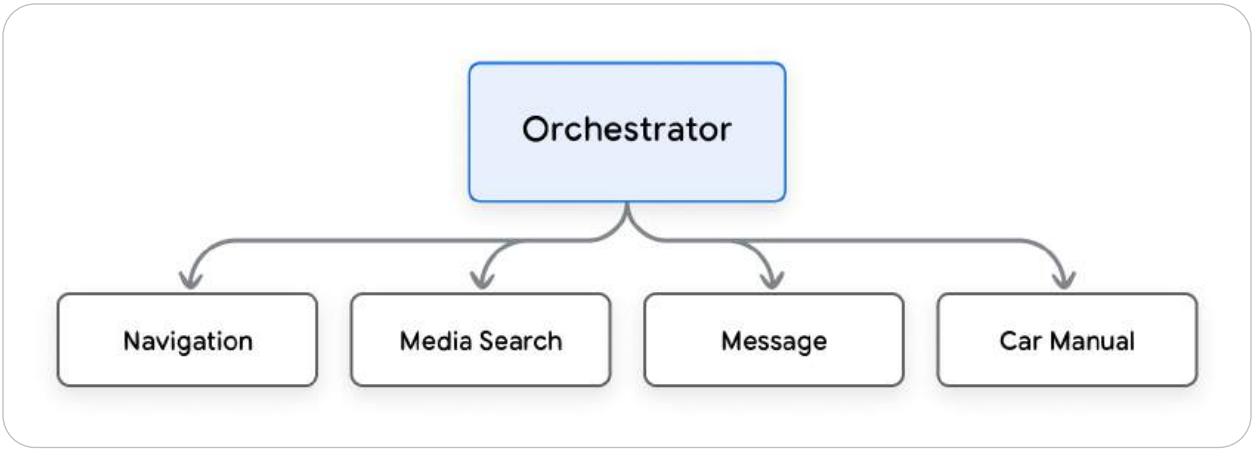


Figure 15. Hierarchical pattern.

The Orchestrator Agent is specialized in routing queries to the most appropriate specialized agents and managing the overall conversation flow. This agent analyzes user queries to determine domain and intent, directs requests to the most suitable specialized agent, maintains conversation context across multiple turns, and manages fallback strategies when specialized agents cannot provide satisfactory responses.

Example Interaction:

First Turn:

User: "Is there a service area ahead with clean toilets?"

Orchestrator Agent: [internally routes to Navigation Agent]

Navigation Agent: "There's a service area 20 kilometers ahead on the E19. It's Aire de Waremme, which was renovated last year and has good ratings for facilities and cleanliness. It also has a Paul bakery and a quick-service restaurant."

Second Turn:

User: "Will it rain at my destination this evening?"

Orchestrator Agent: [recognizes weather query and routes to Weather Information Agent]

Weather Information Agent: "At your destination in Brussels, the forecast shows a 60% chance of light rain in the evening, starting around 19:00. Temperatures will be around 14°C. I'd recommend bringing an umbrella if you plan to be outdoors."

Diamond Pattern

In this variation of the hierarchical pattern, responses from specialized agents pass through a central moderation agent before reaching the user. For instance, the Navigation Agent generates a factual response about a nearby restaurant, which then goes through a Rephraser Agent that adjusts the tone and style based on user preferences before text-to-speech synthesis.

The Rephraser Agent is specialized in adapting the tone, style, and presentation of responses to match user preferences and context. This agent transforms technical information into conversational language, adjusts response length based on driving conditions and urgency, personalizes communication style to match user preferences, and formats complex information for easier comprehension while driving.

Example Interaction:

[Raw response from Car Manual Agent before rephrasing]: "In the event of a tyre pressure monitoring system alert, the vehicle has detected that one or more tyres have fallen below the recommended inflation pressure of 2.2 bar. The affected tyre's pressure is approximately 1.8 bar, which is 18.75% below recommended levels. This does not constitute an immediate safety hazard, but prolonged driving with underinflated tyres can lead to

decreased fuel efficiency, reduced tyre longevity, and potentially compromised handling characteristics. It is advisable to proceed to a service station at your earliest convenience to reinflate the tyre to proper specifications."

Rephraser Agent: [Adapts for driving context, adjusts information density and urgency]

Final Response to User: "Your front right tyre is a bit low - down to 1.8 bar. It's safe to keep driving, but you'll want to add some air at the next petrol station. Low tyre pressure can affect your fuel economy and make the tyre wear out faster. I can help you find a nearby place to fill up if you'd like."

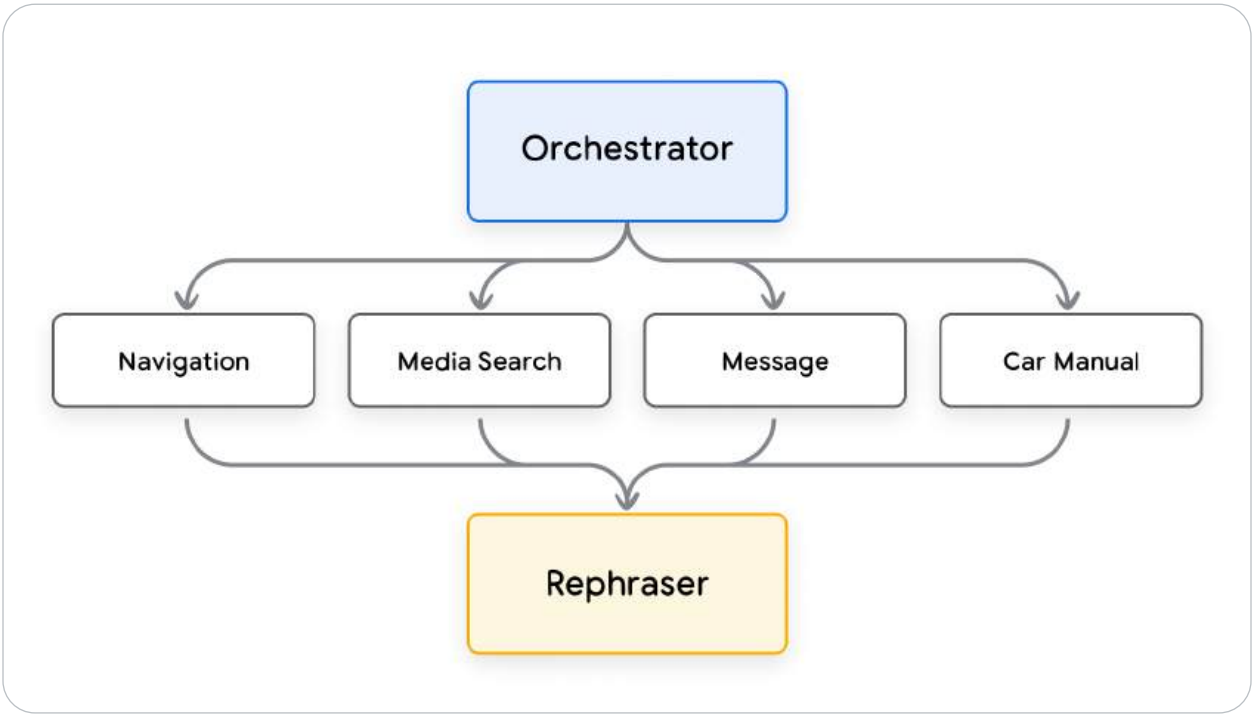


Figure 16. Diamond pattern.

Example transformation:

Initial Response (factual & neutral): "I found a highly-rated sushi restaurant nearby. It's called Sakura Sushi, located at 123 Main Street. It has a 4.7-star rating and is open until 10 PM. Would you like me to start navigation?"

After rephrasing (playful style): "Sushi craving? Say no more! Head over to Sakura Sushi at 123 Main Street, where the fish is fresh and the soy sauce flows freely! Rated 4.7 stars, open till 10 PM. Ready for an umami adventure?"

Peer-to-Peer

Agents can hand off queries to one another when they detect that the orchestration made a routing mistake. This creates a more resilient system that can recover from initial misclassifications.

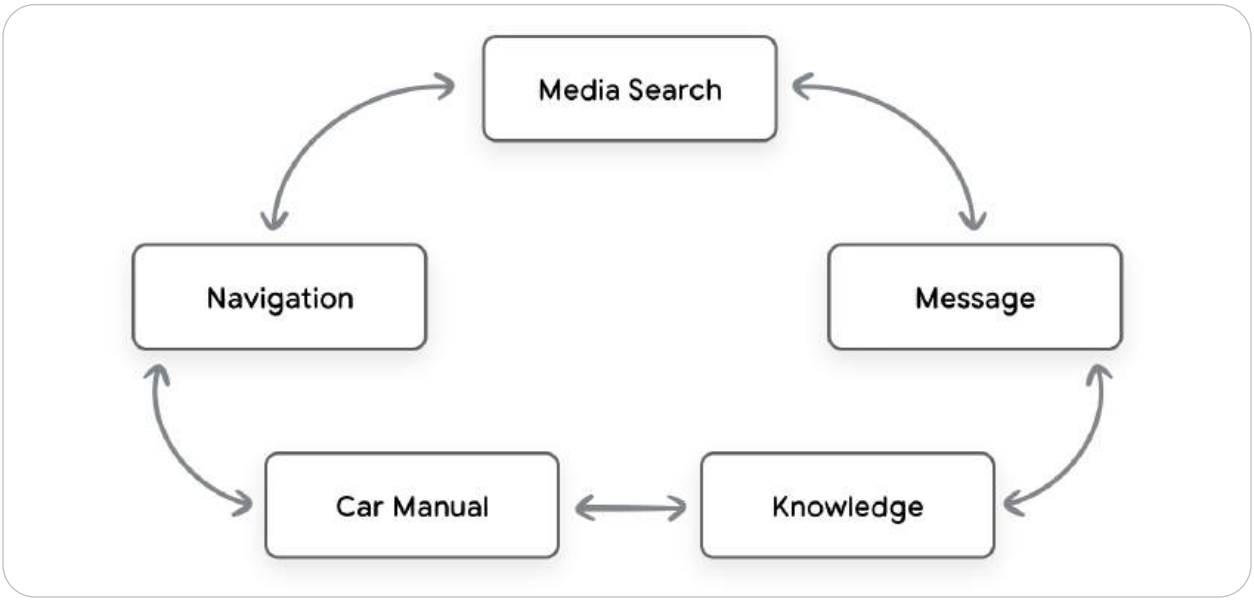


Figure 17. Peer-to-peer.

Example scenario:

1. User asks: "Find a place to eat sushi nearby"
2. The Orchestrator correctly routes this to the Conversational Navigation Agent, which provides information about nearby sushi restaurants.
3. User follows up: "How big is New York's Central Park?"
4. The Orchestrator might initially route this to the Conversational Navigation Agent again (based on the previous navigation-related conversation).
5. However, the Conversational Navigation Agent recognizes this as a general knowledge question rather than a navigation request, and hands it off to the General Knowledge Agent, which can provide factual information about Central Park's size.

Advantages of peer-to-peer hand-off compared to centralized orchestration:

1. **Resilience to misclassification:** Even if the central orchestrator makes an error in routing, specialized agents can recognize when a query falls outside their domain and redirect appropriately.
2. **Domain expertise in routing:** Specialized agents often have better understanding of the boundaries of their own domains. The Media Search Agent knows exactly what kinds of music-related queries it can handle better than a general orchestrator would.
3. **Reduced orchestration complexity:** The central orchestrator doesn't need perfect accuracy in initial routing, reducing the complexity of its decision-making logic.

Collaborative Pattern

The Collaborative Pattern involves multiple agents working on complementary aspects of the same task, with a Response Mixer Agent that combines elements from different agent responses to create a comprehensive answer. This approach recognizes that different agents contribute valuable pieces to a complete solution based on their specialized expertise. The pattern is particularly valuable when:

1. Different aspects of a query require different types of expertise (e.g., technical specifications, practical advice, and conceptual explanations)
2. No single agent has complete information to fully address the user's needs
3. The user would benefit from multiple perspectives on the same question
4. Different specialized agents have access to distinct knowledge bases or reasoning capabilities

Unlike the competitive approach where responses compete, the collaborative pattern assumes that responses from different agents are complementary rather than redundant. The Response Mixer Agent identifies the most valuable information from each source and synthesizes it into a cohesive answer that leverages the unique strengths of each specialist.

For example, when asked about handling hydroplaning, the Car Manual Agent contributes vehicle-specific safety system information, the Driving Tips Agent provides practical driving techniques, and the General Knowledge Agent explains the physics behind the phenomenon. Together, they create a more complete and useful response than any single agent could provide alone.

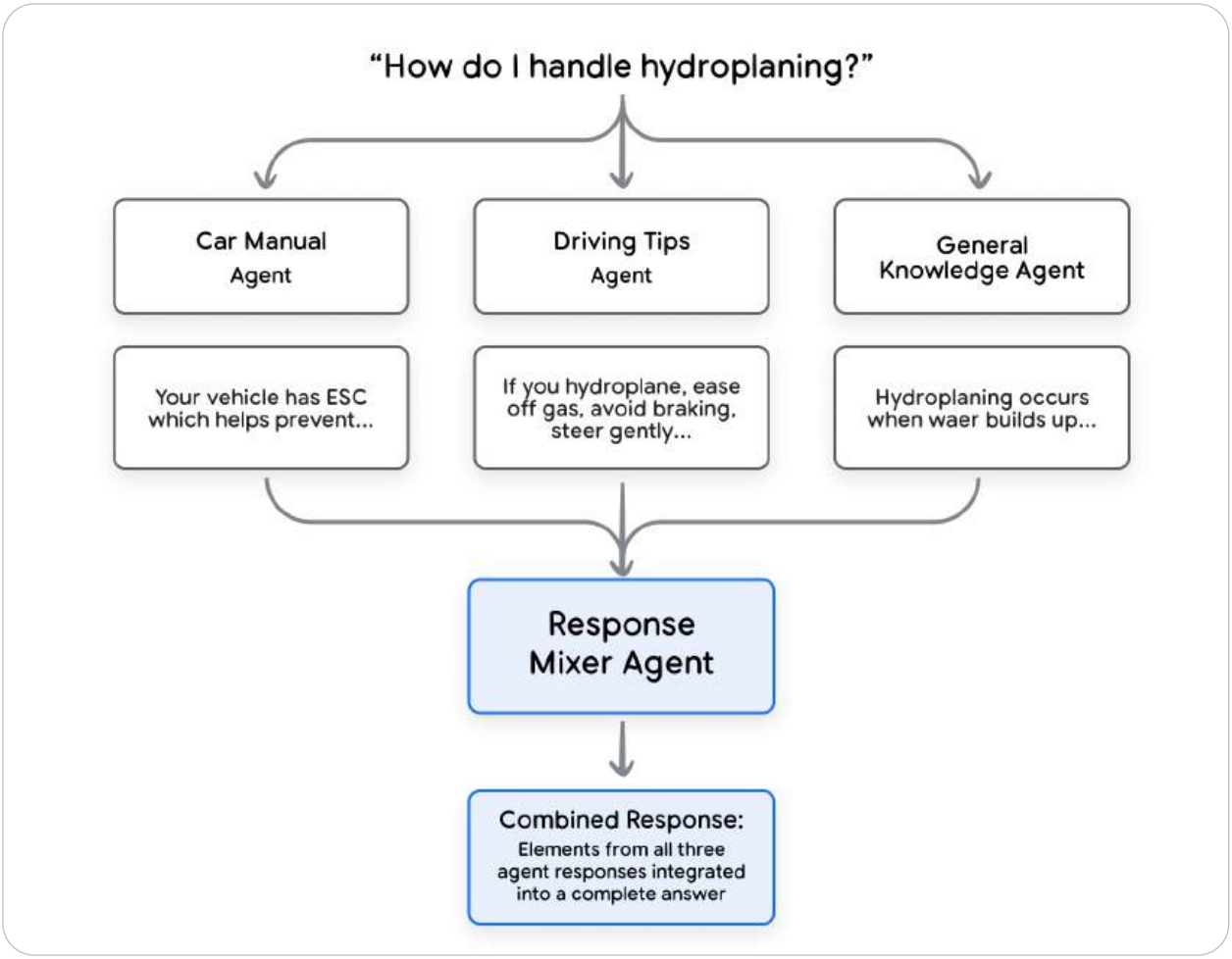


Figure 18. Collaborative pattern.

Response Mixer Agent

The Response Mixer is specialized in picking the best responses from multiple agents and merging them when needed. This agent evaluates answers based on their accuracy and relevance, removes incorrect information, and combines useful parts from different responses to create the most helpful answer for the user.

Example Interaction:

User: "How do I handle aquaplaning?"

[Behind the scenes, multiple agents generate responses:]

Car Manual Agent Response: "Your Audi is equipped with Electronic Stability Control (ESC) and Anti-lock Braking System (ABS) which help prevent aquaplaning by maintaining traction control." [Confidence score: 71%]

General Knowledge Agent Response: "Aquaplaning occurs when your tyres lose contact with the road surface due to water buildup between the tyres and the road." [Confidence score: 65%]

Safety Tips Agent Response: "If you experience aquaplaning, ease off the accelerator, avoid sudden braking or steering, and gently steer in the direction you want to go until you regain traction." [Confidence score: 94%]

Response Mixer Agent: [Picks the most relevant parts and merges them into one complete answer]

Final Response to User: "If you experience aquaplaning, which happens when your tyres lose contact with the road due to water buildup, ease off the accelerator, avoid sudden braking or steering, and gently steer in the direction you want to go until you regain traction. Your Audi's Electronic Stability Control and ABS systems will help maintain control, but proper driving technique is still essential."

This competitive approach ensures that critical safety information isn't missed due to misrouting, and the user receives the most helpful possible response even when their query has multiple valid interpretations.

Adaptive Loop Pattern

The Adaptive Loop Pattern involves iterative refinement through repeated attempts, progressively improving results until they meet desired criteria.

The Conversational Navigation Agent can implement a loop pattern to iteratively improve search results when initial queries don't yield satisfactory outcomes.

Example Interaction:

1. User asks: "Find a nice Italian restaurant that serves vegan options"
2. The Navigation Agent queries Places API with these parameters
3. The agent evaluates the results and finds no restaurants meeting all criteria
4. Instead of returning "no results," the agent automatically reformulates the query:
    - First loop: Searches for "Italian restaurants with vegetarian options"
    - Second loop: Broadens to "Italian restaurants" and then filters for those mentioning plant-based options
    - Third loop: If still unsuccessful, searches for "vegan restaurants" and filters for Italian-influenced cuisine
5. The agent presents the best matches from these progressive searches, explaining how they relate to the original request

This looping behavior enables more robust search capabilities that adapt to availability and context, providing useful results even when exact matches aren't available.

Advantages of Multi-Agent Architecture for Automotive AI

Multi-agent systems bring powerful advantages to automotive AI by breaking down complex tasks into specialized roles. Think of it as assembling a team of experts rather than relying on one generalist.

In this approach, each agent specializes in what it does best. The Navigation Agent focuses solely on finding locations and planning routes. The Media Search Agent becomes an expert in music and podcasts. The Car Manual Agent specializes in vehicle features and troubleshooting. By focusing on specific domains, each agent develops deeper capabilities in its area.

This specialization makes the entire system more efficient. When an agent handles a narrower set of tasks, it becomes simpler to optimize its performance. The result is higher quality responses delivered more quickly and at lower computational cost. Performance improves because the system can match the right resources to each task. Simple requests use minimal processing power, while complex questions tap into more powerful resources only when needed.

Speed matters in a vehicle, and multi-agent systems deliver where it counts. Critical functions like adjusting climate controls or opening windows run on fast, on-device agents for immediate response. Meanwhile, less urgent tasks like finding restaurant recommendationscan use cloud-based agents with more extensive knowledge. This separation ensures that essential vehicle controls remain responsive regardless of what else the system is doing.

This design also creates natural resilience. If internet connectivity drops the essential functions running on on-device agents continue working. You might temporarily lose restaurant recommendations, but climate control and basic media playbacks still function perfectly.

Agent Builder

Vertex AI Agent Builder is a collection of products and services for developers. We have put together a comprehensive platform for you to build and connect agents. The engineering excellence and security from Google Cloud, the AI research from Google Deepmind, and the best practices of Agent Ops. Google Cloud is building our own agents on top of this platform, and now you can too. Expect many more exciting announcements 2025 targeting developers of agents.

Vertex AI Agent Engine streamlines development, relying on Google engineering managed integrations with popular open source agent libraries. It provides a managed autoscaling runtime and many services agents will need (eg: session, examples, trace, evals). This is the very low effort and high value way to safely deploy agents you have written in any framework.

Vertex AI Eval Service provides all of the evaluation tools discussed in this whitepaper, and more. LLMs, RAG, and Agent evals are stable and scalable via the Vertex AI Eval Service, with convenient integrations into monitoring and experimentation offerings.

A large portfolio of agent tools, with more to watch out for:

- Retrieval via Vertex AI Search[26] or RAG Engine[28].
- Non-search based retrieval from DBs via Gen AI Toolbox for Databases[32]

- Application integrations[33] with hundreds of APIs supporting full ACLs
- Turn any API into a managed, enterprise ready tool with Apigee Hub[34]

And of course the best LLMs for agents, with access to Vertex AI Model Garden[35] and also the Gemini family of models[36] which will power the agentic era.

Summary

This whitepaper (a companion to our earlier whitepaper on Agents) has explored the rapidly evolving landscape of generative AI agents, from their fundamental architecture to advanced evaluation techniques and the transformative potential of multi-agent systems.

Key Takeaways for Developers:

1. **Agent Ops is Essential:** Building successful agents goes far beyond the initial proof-of-concept. Embrace Agent Ops principles, integrating best practices from DevOps and MLOps, but also focusing on agent-specific elements like tool management, orchestration, memory, and task decomposition.
2. Metrics Drive Improvement: Start with business-level KPIs (like goal completion, user engagement, or revenue) as your "north star." Then, instrument your agents to track granular metrics related to critical tasks, user interactions, and agent actions (traces). Human feedback (□□, user surveys) is invaluable.
3. **Automated Evaluation is Key:** Don't rely solely on manual testing. Implement automated evaluation frameworks that assess agent capabilities, trajectory (the steps taken), and the final response. Leverage techniques like exact match, in-order match, precision/recall for trajectory evaluation, and autoraters (LLMs as judges) for final response quality.

4. **Human-in-the-Loop is Crucial:** Automated metrics are powerful, but human evaluation provides essential context, especially for subjective aspects like creativity, common sense, and nuance. Use human feedback to calibrate and validate your automated evaluation methods. Don't outsource the domain knowledge.
5. **Multi-Agent Systems Offer Advantages:** Consider multi-agent architectures for complex tasks. They can improve accuracy, efficiency, scalability, and fault tolerance. Understand different design patterns (sequential, hierarchical, collaborative, competitive) and choose the right one for your application.
6. **Agentic RAG Improves Relevance:** Move beyond traditional RAG by incorporating agents that actively refine search queries, evaluate retrieved information, and adapt to evolving knowledge. This leads to more accurate and contextually relevant responses.
7. **Search Optimization is Foundational to RAG:** Before diving into complex agentic RAG, optimize your underlying search engine. Techniques like semantic chunking, metadata enrichment, fine-tuning embedding models, and using rankers can significantly improve retrieval quality.
8. **Agent and Tool Registries are Important:** As the number of Agents or Tools you are using grow, a registry to manage the capabilities, ontology, and performance becomes essential.
9. **Security is Paramount:** When deploying agents, especially within an enterprise, prioritize security. Leverage platforms like Google Agentspace that offer built-in security features like RBAC, VPC Service Controls, and IAM integration.
10. **Efficient use of developer cycles:** The classic build vs buy design choices remain front of mind, as the industry of gen AI agents is rapidly evolving. Consider platforms and products as alternatives to building everything from scratch. This will buffer some of the churn of a fast changing industry and allow you to focus on your data, domain, and users.

11. **Agents in the enterprise:** Agents are transforming the way we work by making us much more productinve, and the way automation can be accomplished. Knowledge workers will increasingly be managing fleets of agents and novel UX will emerge. Google Agentspace is a powerful tool allowing to put Enterprise Search, AI and AI Agents on top of company's data and workflows

Future Directions for Agent Research and Development: The field of AI agents is undergoing rapid evolution. Key areas of ongoing research and development include:

- **Advanced Evaluation Methods:** Developing more robust and scalable evaluation techniques, including process-based evaluation (focusing on reasoning), AI-assisted evaluation, and standardized benchmarks.
- **Multi-Agent Coordination:** Improving the coordination and communication mechanisms within multi-agent systems to enable more effective collaboration, task handling, and reasoning.
- **Real-World Adaptation:** Creating agents that can adapt and learn in dynamic, unpredictable real-world environments. Production systems like automotive AI illustrate how agents must balance between on-device performance for critical functions and cloud-based capabilities for complex tasks, often adapting to changing connectivity conditions.
- **Explainability and Interpretability:** Making agent behavior more transparent and understandable, allowing developers and users to gain deeper insights into their decision-making processes.
- **Long-Term Memory and Learning:** Developing more sophisticated memory mechanisms that allow agents to retain and utilize information over extended periods, enabling continuous learning and adaptation.

- **Agent Communication Protocols:** Better defining how agents share tasks, knowledge, and messages, especially across remote systems which are opaque.
- **From Agents to contractors:** In order for agents to get to next level of reliability and utility, we will need to step up the definition of tasks, making them into contracts with clear deliverables, validation mechanisms, and ability to negotiate ambiguities, similarly to how we contract work from other companies.

Call to Action:

The future of AI is agentic. We encourage developers to embrace these concepts and begin building the next generation of intelligent applications. Start experimenting with the tools and techniques discussed in this whitepaper. Explore the resources available, such as Google Agentspace, NotebookLM Enterprise, Vertex Eval Service, Cloud Observability, and Vertex AI Search, to accelerate your development process. Dive into the provided code examples, tutorials, and documentation to gain hands-on experience. Build, evaluate, iterate, and contribute to the growing community of agent developers. The possibilities are limitless, and the time to build is now! Specifically, get started with the code and Colab notebooks in the references.

Endnotes

1. Shafran, I., Cao, Y. et al., 2022, 'ReAct: Synergizing Reasoning and Acting in Language Models'. Available at: https://arxiv.org/abs/2210.03629.
2. Wei, J., Wang, X. et al., 2023, 'Chain-of-Thought Prompting Elicits Reasoning in Large Language Models'. Available at: https://arxiv.org/pdf/2201.11903.pdf.
3. Wang, X. et al., 2022, 'Self-Consistency Improves Chain of Thought Reasoning in Language Models'. Available at: https://arxiv.org/abs/2203.11171.
4. Diao, S. et al., 2023, 'Active Prompting with Chain-of-Thought for Large Language Models'. Available at: https://arxiv.org/pdf/2302.12246.pdf.
5. Zhang, H. et al., 2023, 'Multimodal Chain-of-Thought Reasoning in Language Models'. Available at: https://arxiv.org/abs/2302.00923.
6. Yao, S. et al., 2023, 'Tree of Thoughts: Deliberate Problem Solving with Large Language Models'. Available at: https://arxiv.org/abs/2305.10601.
7. Long, X., 2023, 'Large Language Model Guided Tree-of-Thought'. Available at: https://arxiv.org/abs/2305.08291.
8. Google. ';Google Gemini Application'. Available at: http://gemini.google.com.
9. Swagger. 'OpenAPI Specification'. Available at: https://swagger.io/specification/.
10. Xie, M., 2022, 'How does in-context learning work? A framework for understanding the differences fromtraditional supervised learning'. Available at: https://ai.stanford.edu/blog/understanding-incontext/.
11. Google Research. 'ScaNN (Scalable Nearest Neighbors)'. Available at: https://github.com/google-research/google-research/tree/master/scann.
12. LangChain. 'LangChain'. Available at: https://python.langchain.com/v0.2/docs/introduction/.
13. Sokratis Kartakis, 2024, 'GenAI in Production: MLOps or GenAIOps?'. Available at: https://medium.com/google-cloud/genai-in-production-mlops-or-genaiops-25691c9becd0.
14. Sokratis Kartakis, 2024 'Gen AI Ops, Operationalize Generative AI, A practical Guide'. Available at: https://medium.com/google-cloud/genaiops-operationalize-generative-ai-a-practical-guide-d5bedaa59d78.

15. Cloud Trace overview. Available at: https://cloud.google.com/trace/docs/overview.

16. Berkeley Function-Calling Leaderboard (BFCL). Available at: https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html.

17. Karthik Narasimhan, et al. 2024, 'τ-bench'. Available at https://arxiv.org/abs/2406.12045.

18. Karthik Valmeekam, et al., 2023, 'PlanBench'. Available at: https://arxiv.org/abs/2206.10498.

19. Xiao Liu, et al., 2023, 'AgentBench'. Available at: https://arxiv.org/abs/2308.03688.

20. Martin Iglesias, et al., 2025, `DBAStep` Available at: https://huggingface.co/spaces/adyen/DABstep.

21. LangSmith platform for agent observability. Available at: https://docs.smith.langchain.com/evaluation/concepts#agents.

22. Mingchen Zhuge, et al., 2024, 'Agent-as-a-Judge: Evaluate Agents with Agents'. Available at: https://arxiv.org/abs/2410.10934.

23. Multi-agent documentation from LangGraph. Available at: https://langchain-ai.github.io/langgraph/concepts/multi_agent/.

24. LangChain blog 2024, 'Multi-agent workflows'. Available at: https://blog.langchain.dev/langgraph-multi-agent-workflows/.

25. Vectorize blog 2024, 'How I finally got agentic RAG to work right'. Available at: https://vectorize.io/how-i-finally-got-agentic-rag-to-work-right/.

26. Vertex AI Search, product documentation. Available at: https://cloud.google.com/enterprise-search.

27. Vertex AI Search Builder APIs, product documentation. Available at: https://cloud.google.com/generative-ai-app-builder/docs/builder-apis.

28. Vertex AI RAG Engine, product documentation. Available at: https://cloud.google.com/vertex-ai/generative-ai/docs/rag-overview.

29. Agentspace product documentation. Available at: https://cloud.google.com/agentspace/agentspace-enterprise/docs/overview.

30. NotebookLM Enterprise product documentation. Available at: https://cloud.google.com/agentspace/notebooklm-enterprise/docs/overview.

31. Juraj Gottweis, et. al., 2025, 'Accelerating scientific breakthroughs with an AI co-scientist'. Available at: https://research.google/blog/accelerating-scientific-breakthroughs-with-an-ai-co-scientist/.

32. Hamsa Buvaraghan, et al. 2025, 'Announcing public beta of Gen AI Toolbox for Databases'. Available at: https://cloud.google.com/blog/products/ai-machine-learning/announcing-gen-ai-toolbox-for-databases-get-started-today?e=48754805.

33. Google Cloud Integration Connectors, product documentation. Available at: https://cloud.google.com/integration-connectors/docs.

34. Apigee API Hub, product documentation. Available at: https://cloud.google.com/apigee/docs/apihub/what-is-api-hub.

35. Vertex AI Model Garden, product documentation. Available at: https://cloud.google.com/model-garden.

36. Gemini family of LLMs, product documentation. Available at: https://cloud.google.com/vertex-ai/generative-ai/docs/learn/models#gemini-models.

37. Get Started Evaluating Agents with the Vertex Eval Service. Available at: https://cloud.google.com/vertex-ai/generative-ai/docs/models/evaluation-agents.

38. Irina Sigler, Ivan Nardini. Jan 2025 'Introducing Agent Evaluation in Vertex AI'. Available at: https://cloud.google.com/blog/products/ai-machine-learning/introducing-agent-evaluation-in-vertex-ai-gen-ai-evaluation-service?e=48754805.

39. Review sample agent evaluation notebooks for LangGraph, CrewAI, and LangChain. Available at: https://github.com/GoogleCloudPlatform/generative-ai/blob/main/gemini/evaluation/.

40. Review many sample agents, primarily beginner and intermediate level. Available at: https://github.com/GoogleCloudPlatform/generative-ai/.

41. Review many sample agents, intermediate and advanced levels. Available at: https://github.com/GoogleCloudPlatform/applied-ai-engineering-samples.