Overview

- **Agent Evaluation: Trajectory:** This refers to the **step-by-step sequence of actions, decisions, and tool calls** an agent takes to arrive at a final answer. Evaluating the trajectory assesses the **reasoning process, efficiency, and correctness of tool usage**—not just the final output. It typically involves comparing the agent's actual steps against an "expected" or "ground truth" sequence.
- **Need for a Sub-Agent (When and Why):** A sub-agent is a **specialized, modular, and autonomous LLM agent** delegated a specific part of a larger, complex task by a primary agent (or orchestrator).
    1. **When:** When the primary task is too complex, involves multiple distinct domains/steps, exceeds a single agent's context window, or requires specialized tools.
    2. **Why:** To achieve **separation of concerns** (specialized expertise, dedicated tools, custom prompts), enable **parallel processing** (faster execution), and enhance **context efficiency** (each sub-agent only needs context relevant to its task).
- **Levels of Customization for Medical LLM:** These are the primary techniques used to specialize a general-purpose LLM for a high-stakes, domain-specific area like medicine. The levels typically represent increasing effort and specialization:
    1. **Prompt Engineering:** Structuring the input query and system instructions (e.g., "Act as a specialized cardiologist") to guide the model's behavior.
    2. **Retrieval-Augmented Generation (RAG):** Connecting the LLM to an external, authoritative knowledge base (e.g., medical journals, clinical guidelines) to ground its responses in up-to-date, specific facts.
    3. **Fine-Tuning:** Training the pre-trained model on a smaller, high-quality, domain-specific dataset (e.g., medical transcripts, patient notes) to adapt its internal weights, vocabulary, and response style.
- **Human-in-the-Loop (HITL):** A safety and reliability pattern where a **human user is intentionally integrated into the automated workflow** to review, approve, reject, or provide input at critical decision points. It balances the efficiency of automation with the need for human judgment

and oversight, especially for high-stakes actions like executing a transaction or making a clinical decision.

Google ADK (Agent Development Kit) Details

ADK is an **open-source, code-first Python framework** built by Google for building, evaluating, and deploying sophisticated AI agents and multi-agent systems, optimized for Gemini and the Google ecosystem.

Agent Evaluation Specifically Trajectory (in ADK)

The ADK framework includes a robust evaluation component to assess performance against predefined datasets and criteria.

- **Trajectory Evaluation:** ADK specifically provides the `tool_trajectory_avg_score` metric. This metric assesses the agent's internal thought process by comparing the **actual sequence of tool calls (name and arguments)** against an **expected sequence** defined in an evaluation test case.
- **Match Types:** Developers can set the strictness of the comparison:
  - `EXACT`**:** Requires the tool calls to match the expected ones in name, arguments, and order.
  - `IN_ORDER`**:** Requires the tool calls to appear in the correct sequence, but allows for minor variations in arguments.
  - `ANY_ORDER`**:** Only checks if the required tools were called, regardless of the order.
- **Outcome Evaluation:** The final output is assessed using the `response_match_score` (which typically uses a metric like **ROUGE-1** to compare semantic similarity) and other metrics for safety/grounding.

Need for a Sub-Agent (When and Why) (in ADK)

ADK is designed with **modular multi-agent systems** in mind.

- **When/Why:** ADK supports patterns like **Orchestrator-Specialist Frameworks** or **Multi-Agent Pipelines**.
  - You need a sub-agent when you need a clear **division of labor** (e.g., one agent is a `ResearchAgent` using a search tool, and another is a `SummarizerAgent` using a summarization tool).
  - ADK's `Agent2Agent (A2A) Protocol` facilitates communication and delegation, allowing agents to treat other

agents as callable tools, making sub-agent design highly modular and reusable.

Levels of Customization for Medical LLM (in ADK context)

While ADK is model-agnostic, its deep integration with Google Cloud's Vertex AI ecosystem allows for leveraging the full spectrum of specialization:

1. **Prompt/System Instruction:** Set the `instruction` property in the ADK agent's configuration to define its "medical personality," role, and guardrails.
2. **RAG/Grounding:** ADK integrates with the **Model Context Protocol (MCP)**, allowing the agent to connect to structured and unstructured **enterprise truth data** (like EMR systems, medical databases, or Vertex AI Search) to ground its answers, ensuring up-to-date, authoritative medical information.
3. **Model Specialization:** The ADK agent can utilize a **Fine-Tuned or Pre-Trained Medical LLM** (like a specialized Gemini model) as its core engine, which is the deepest level of customization.

Human in a Loop (HITL) (in ADK)

ADK explicitly supports the HITL pattern for reliability and safety.

- **Tool Confirmation Flow:** ADK provides a `Tool Confirmation` flow (a form of HITL) where the agent's planned use of a high-impact tool (e.g., a tool that executes a transaction, sends a message, or modifies a database) can be paused, requiring **explicit confirmation or custom input** from the user before execution.
- **Approval/Guardrails:** HITL can be used to set a checkpoint, asking a human reviewer to step in when the agent's confidence score is low or when a high-risk action is proposed.

Agent Development Kit (ADK) is a flexible and modular framework for **developing and deploying AI agents**. ADK is **model-agnostic**, **deployment-agnostic**, and is built for **compatibility with other frameworks**.

Core Concepts

- **Agent:** The fundamental worker unit designed for specific tasks. Agents can use language models (`LlmAgent`) for complex reasoning, or act as deterministic controllers of the execution, which are called workflow agents (`SequentialAgent`, `ParallelAgent`, `LoopAgent`).
- **Tool:** Gives agents abilities beyond conversation, letting them interact with external APIs, search information, run code, or call other services.
- **Callbacks:** Custom code snippets you provide to run at specific points in the agent's process, allowing for checks, logging, or behavior modifications.
- **Session Management (`Session` & `State`):** Handles the context of a single conversation (`Session`), including its history (`Events`) and the agent's working memory for that conversation (`State`).
- **Memory:** Enables agents to recall information about a user across *multiple* sessions, providing long-term context (distinct from short-term session `State`).
- **Artifact Management (`Artifact`):** Allows agents to save, load, and manage files or binary data (like images, PDFs) associated with a session or user.
- **Code Execution:** The ability for agents (usually via Tools) to generate and execute code to perform complex calculations or actions.
- **Planning:** An advanced capability where agents can break down complex goals into smaller steps and plan how to achieve them like a ReAct planner.
- **Models:** The underlying LLM that powers `LlmAgent`s, enabling their reasoning and language understanding abilities.
- **Event:** The basic unit of communication representing things that happen during a session (user message, agent reply, tool use), forming the conversation history.

- **Runner:** The engine that manages the execution flow, orchestrates agent interactions based on Events, and coordinates with backend services.

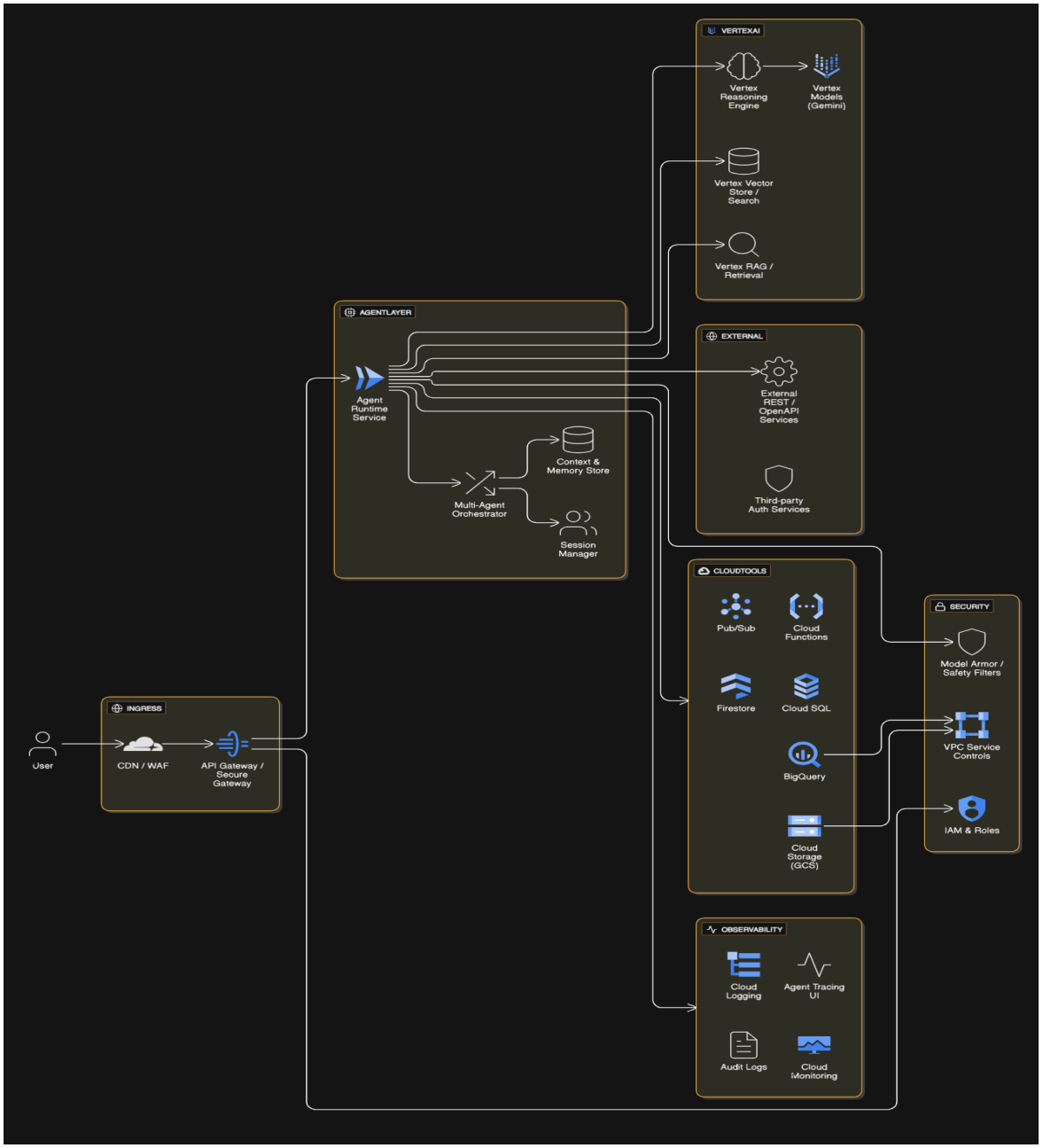
Key Capabilities

- Multi-Agent System Design: Easily build applications composed of multiple, specialized agents arranged hierarchically. Agents can coordinate complex tasks, delegate sub-tasks using LLM-driven transfer or explicit `AgentTool` invocation, enabling modular and scalable solutions.
- Rich Tool Ecosystem: Equip agents with diverse capabilities. ADK supports integrating custom functions (`FunctionTool`), using other agents as tools (`AgentTool`), leveraging built-in functionalities like code execution, and interacting with external data sources and APIs (e.g., Search, Databases). Support for long-running tools allows handling asynchronous operations effectively.
- Flexible Orchestration: Define complex agent workflows using built-in workflow agents (`SequentialAgent`, `ParallelAgent`, `LoopAgent`) alongside LLM-driven dynamic routing. This allows for both predictable pipelines and adaptive agent behavior.
- Integrated Developer Tooling: Develop and iterate locally with ease. ADK includes tools like a command-line interface (CLI) and a Developer UI for running agents, inspecting execution steps (events, state changes), debugging interactions, and visualizing agent definitions.
- Native Streaming Support: Build real-time, interactive experiences with native support for bidirectional streaming (text and audio). This integrates seamlessly with underlying capabilities like the Multimodal Live API for the Gemini Developer API (or for Vertex AI), often enabled with simple configuration changes.
- Built-in Agent Evaluation: Assess agent performance systematically. The framework includes tools to create multi-turn evaluation datasets and run evaluations locally (via CLI or the dev UI) to measure quality and guide improvements.
- Broad LLM Support: While optimized for Google's Gemini models, the framework is designed for flexibility, allowing integration with various LLMs (potentially including open-source or fine-tuned models) through its `BaseLlm` interface.
- Artifact Management: Enable agents to handle files and binary data. The framework provides mechanisms (`ArtifactService`, context

methods) for agents to save, load, and manage versioned artifacts like images, documents, or generated reports during their execution.

- Extensibility and Interoperability: ADK promotes an open ecosystem. While providing core tools, it allows developers to easily integrate and reuse third-party tools and data connectors.
- State and Memory Management: Automatically handles short-term conversational memory (`State` within a `Session`) managed by the `SessionService`. Provides integration points for longer-term `Memory` services, allowing agents to recall user information across multiple sessions.

Architecture



Components

Users (UI): Web, mobile, or chat UI that sends user requests and receives agent responses.

API Gateway / Secure Gateway: Entry point providing authentication (OAuth2 / API keys), rate limits, and WAF/edge protection.

Agent Runtime Service: Executes agent plans, coordinates tool calls, and manages session state and context windows.

Multi-Agent Orchestrator: Routes tasks between specialized agents (e.g., Search Agent, Planner Agent, Execution Agent).

Context & Memory Store: Stores short-term session context and longer-term memories (could be Firestore / Cloud SQL / Vertex vector store depending on requirements).

Vertex Models (Gemini): Primary LLMs used by agents for reasoning and generation.

Vertex Reasoning Engine: Executes stepwise reasoning and tool-calling plans when complex workflows are required.

Vertex Vector Store / RAG: Managed vector DB + search for retrieval-augmented generation and citations.

Cloud Tools: GCS, BigQuery, Pub/Sub, Cloud Functions, Firestore, Cloud SQL — used as callable tools by the agent.

External APIs / OpenAPI: Third-party APIs integrated via HTTP / OpenAPI tools.

Agent Tracing / Observability: Token-level traces, latency breakdowns, tool call logs, Cloud Monitoring dashboards, and audit logs.

Security & Governance: IAM roles, VPC Service Controls, encryption, and Model Armor for prompt-injection and content filtering.

Deployment & Flow

1. **Dev → Staging → Prod**: Build agents locally, test in a staging Vertex environment, then deploy to production endpoints with autoscaling.
2. **Tool Attachment**: Expose Cloud and external services as ADK Tools (OpenAPI / HTTP / Cloud-specific connectors). Tools declare schemas so the Reasoning Engine can call them reliably.
3. **Observability**: Enable Agent Tracing early to capture token usage and tool interactions for debugging and cost analysis.
4. **Safety**: Enable Model Armor and content filters for all production endpoints. Run adversarial tests in staging to validate guardrails.

5. **Scaling**: Vertex Agent Deployments handle autoscaling; use Pub/Sub for asynchronous long-running tasks and Cloud Functions for light-weight worker tasks.

Service/Capability	Description
Session Management (State)	Manages the current conversation thread. Session tracks the history, and State stores temporary data relevant only to the active interaction (e.g., items in a cart).
Memory Service	Manages long-term knowledge that spans multiple conversations or external knowledge bases. It allows agents to recall context and preferences from past sessions.
Agent2Agent (A2A) Protocol	An open communication standard that allows different agents (even those built with other frameworks) to publish their capabilities and securely delegate tasks to sub-agents.
Model Context Protocol (MCP)	A protocol that enables agents to connect to enterprise truth data (databases, documents) for Retrieval-Augmented Generation (RAG) and grounding.
Evaluation	Built-in framework to systematically assess agent performance, including the final response quality and the step-by-step execution trajectory (tool usage).
Deployment (Agent Engine)	Services that simplify the path to production, offering a fully managed runtime environment for deploying, securing, and scaling agents on Google Cloud.
Observability	Provides full logging, tracing of task delegation, and visualization of agent decisions and tool usage, promoting a robust debugging experience.
Bidi-Streaming	Native support for bidirectional audio and video streaming, enabling real-time, interactive, and multimodal conversational experiences.

Tool Category	Tool Name	Description
General / Built-in Tools	HTTP / REST Tool	Call any external REST API with schema-driven parameters.
	OpenAPI Tool	Auto-generate tools from OpenAPI specs for API integration.
Google Cloud Tools	Cloud Storage Tool	Read/write files to GCS buckets.
	BigQuery Tool	Run SQL queries and fetch datasets from BigQuery.
	Pub/Sub Tool	Publish or subscribe to Pub/Sub topics for event workflows.
	Firestore / Datastore Tool	Read or write NoSQL documents/collections.
	Cloud Functions Tool	Trigger serverless functions as tool actions.
	Cloud SQL Tool	Run SQL queries on Cloud SQL databases.
	Workflows Tool	Trigger Google Cloud Workflows from the agent.
Vertex AI Tools	Vertex Model Tool	Call Gemini models as sub-agents or tool actions.
	Vertex Vector Store Tool	Insert, query, or delete embeddings in a vector DB.
	Vertex Search Tool	Query enterprise search indices with reranking/citations.
File & Utility Tools	File Read/Write Tool	Operate on temporary files during agent execution.
	JSON / Parsing Tools	Utilities for schema validation, parsing, data formatting.
Custom Tools	User-defined Tools	Python functions or external APIs wrapped as ADK tools.

Agent is a self-contained execution unit designed to act autonomously to achieve specific goals.

Feature	LLM Agent (`LlmAgent`)	Workflow Agent	Custom Agent (`BaseAgent` subclass)
Primary Function	Reasoning, Generation, Tool Use	Controlling Agent Execution Flow	Implementing Unique Logic/Integrations
Core Engine	Large Language Model (LLM)	Predefined Logic (Sequence, Parallel, Loop)	Custom Code
Determinism	Non-deterministic (Flexible)	Deterministic (Predictable)	Can be either, based on implementation
Primary Use	Language tasks, Dynamic decisions	Structured processes, Orchestration	Tailored requirements, Specific workflows

Mechanisms for model integration with ADK

1. **Direct String / Registry:** For models tightly integrated with Google Cloud (like Gemini models accessed via Google AI Studio or Vertex AI) or models hosted on Vertex AI endpoints. You typically provide the model name or endpoint resource string directly to the `LlmAgent`. ADK's internal registry resolves this string to the appropriate backend client, often utilizing the `google-genai` library.
2. **Wrapper Classes:** For broader compatibility, especially with models outside the Google ecosystem or those requiring specific client configurations (like models accessed via Apigee or LiteLLM). You instantiate a specific wrapper class (e.g., `ApigeeLlm` or `LiteLlm`) and pass this object as the `model` parameter to your `LlmAgent`.

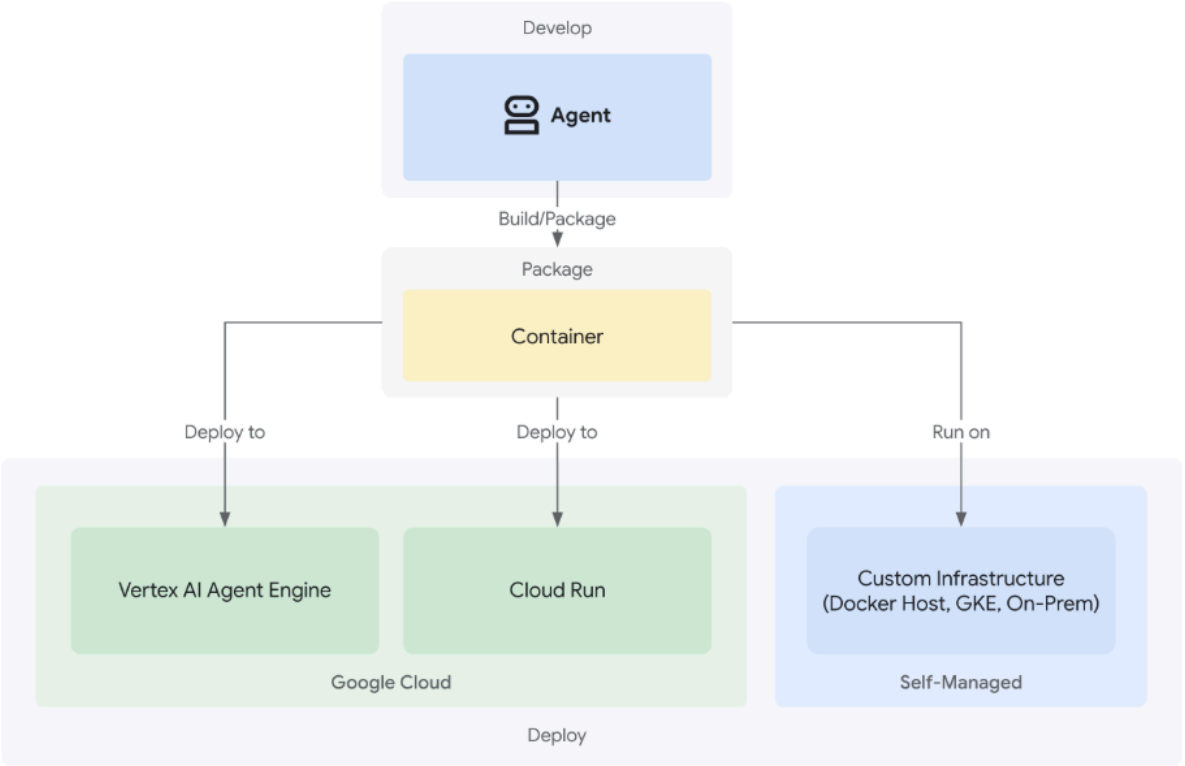
Multi-Agent System:

A multi-agent system is an application where different agents, often forming a hierarchy, collaborate or coordinate to achieve a larger goal.

1. Agent Composition
   - Agent hierarchy (tree structured) -> parent to sub_agent argument
   - Workflow agents -> seq, parallel and loop (passes state/ context)
   - Communication -> shared session state, agent transfer and explicit invocation (agent_tool), A2A protocol, MCP
2. Multi-Agent Patterns
   - Coordinator/ Dispatcher -> LlmAgent (coor) manages sub_agents

- Sequential -> op of one feeds to next (fixed order)
- Parallel -> multiple sub_agents concurrently (independent tasks)
- Hierarchical Task Decomposition -> higher-level to lower-level agents (break complex tasks)
- Human in the Loop -> human intervention for approval, correction

Deployment



Feature Comparison
Primary focus

- ADK: multi-agent orchestration + production deployment on Google Cloud (Vertex) and model/tool integrations.

- LangGraph: durable execution, stateful long-running agents, streaming & checkpoints.

- Haystack: retrieval, RAG, QA pipelines and production-ready document workflows.

Language / SDK support

- ADK: Python, Java, Go (first-class support from Google).
- LangGraph: Python (LangChain ecosystem).
- Haystack: Primarily Python.

Model ecosystem

- ADK: optimized for Gemini & Vertex but model-agnostic (can connect other models/tools).
- LangGraph: integrates with LangChain components / any LLMs you plug in.
- Haystack: model-agnostic; heavy focus on embeddings + vector search backends.

Orchestration & state

- ADK: agent patterns, tool integration, multi-agent coordination.
- LangGraph: durable state management, streaming, retries, long-running tasks.
- Haystack: pipeline components and routing — strong for retrieval+LLM flows.

Observability / deployment

- ADK: CLI + deployment flow to Vertex, built-in observability (token/latency traces, testing playgrounds reported in recent updates) (Vertex AI agent engine, Cloud Run, Google Kubernetes Service).
- LangGraph: focused on orchestration primitives; deployment depends on infra you choose.
- Haystack: production deployment patterns (with OpenSearch/Elasticsearch/FAISS etc.) and pipeline monitoring capabilities.

Features ADK supports best

Seamless Vertex/Gemini + Google Cloud production integration
ADK is purpose-built to plug into Vertex AI/Agent Builder and Gemini, so moving from local dev → managed deployment is very direct (single-command deploy flows and cloud-native hooks). If you plan to host on Google Cloud and want the tightest end-to-end experience, ADK is the best fit.

First-class multi-agent architecture and prebuilt agent roles/plugins
ADK offers structured agent types and prebuilt plugins (tool integrations, "self-heal", etc.) aimed at multi-agent systems and agent collaboration out of the box — this reduces the amount of custom glue code for complex agent interactions versus building those patterns yourself in LangGraph or Haystack.

Deployment + observability toolchain (CLI, traces, dashboards)
ADK adds deployment automation and observability (traces, token/latency usage dashboards, testing playgrounds, trace tabs) that are built into the Google Agent Builder story — this makes debugging, cost/usage monitoring and safe rollouts simpler than stitching together external tools. LangGraph focuses on durable execution primitives and Haystack on pipelines; neither offers the same out-of-the-box cloud observability experience tied to a managed Vertex product.

Security & guardrails integrated for production agents
Google highlights features like Model Armor (prompt-injection screening) and security command controls as part of the upgraded Agent Builder/ADK ecosystem — useful when you need baked-in production security for agents rather than adding ad hoc protections.

Multi-language SDK & enterprise-ready developer ergonomics
ADK provides SDKs and examples in Python, Java and Go plus official codelabs/docs and sample repos, which helps teams that are not Python-only to adopt the framework with standard enterprise languages. LangGraph and Haystack are more Python-centric.

Prebuilt tools / integrations (OpenAPI, MCP, tool plugins)
ADK exposes structured ways to attach tools (OpenAPI integrations, MCP connectors, etc.) as first-class tools for agents so tool-calling and capability wiring can be more standardized than the ad-hoc connectors you typically write in other frameworks. This reduces engineering overhead for common integrations.