The top 10 frameworks for agentic development in 2025, widely recognized for enabling advanced, modular, and collaborative AI agent systems, are:

1. \*\*Microsoft AutoGen\*\* – Enterprise-grade multi-agent orchestration with strong error handling and robust infrastructure[1][2][3][4][5].

2. \*\*LangGraph\*\* – Graph-based orchestration for building complex, stateful multi-agent workflows, part of the LangChain ecosystem[1][2][4][5].

3. \*\*CrewAI\*\* – Role-based multi-agent creation with fast prototyping and collaborative agent workflow focus[1][2][4][5].

4. \*\*Anaconda AI Navigator\*\* – Privacy-preserving, locally managed agentic systems with access to 200+ pre-trained LLMs, emphasizing data security[1].

5. \*\*OpenAI Swarm\*\* – Lightweight, experimental platform for multi-agent scenarios tightly integrated with OpenAI models[1][2][3][5].

6. \*\*Microsoft Semantic Kernel\*\* – Integrates advanced semantic and contextual reasoning into enterprise applications, with strong .NET support[1][2][4].

7. \*\*LangChain\*\* – General-purpose workflow and retrieval orchestration, highly modular and supports multi-agent patterns[2][4][5].

8. \*\*Phidata\*\* – Adaptive agent creation and LLM integration, emerging as a flexible problem-solving framework[3].

9. \*\*PromptFlow\*\* – Visual workflow development with Azure integration, streamlining agentic solution deployment[3].

10. \*\*Atomic Agents\*\* – Open-source toolbox for building decentralized, customizable multi-agent systems ideal for tailored applications[4].

These frameworks stand out for features such as modular configuration, stateful workflow control, secure integrations, fine-grained orchestrations, and strong community or enterprise support[1][2][3][4][5]. The best fit depends on your use case and environment: for enterprise cloud deployments, AutoGen or Semantic Kernel excel; for rapid prototyping or research, CrewAI or Swarm are popular; for graph-based process management or integration with retrieval-augmented generation, LangGraph and LangChain are top choices.

[1] https://www.anaconda.com/guides/agentic-ai-tools

[2] https://www.plivo.com/blog/agentic-ai-frameworks/

[3] https://www.reddit.com/r/AI\_Agents/comments/1hq9il6/best\_ai\_agent\_frameworks\_in\_2025\_a\_comprehensive/

[4] https://www.shakudo.io/blog/top-9-ai-agent-frameworks

[5] https://research.aimultiple.com/agentic-frameworks/

[6] https://www.ibm.com/think/insights/top-ai-agent-frameworks

[7] https://www.reddit.com/r/AI\_Agents/comments/1hzbl20/recommendations\_for\_ai\_agent\_frameworks\_llms\_for/

[8] https://pub.towardsai.net/the-ultimate-guide-to-agentic-ai-frameworks-in-2025-which-one-should-you-choose-to-build-the-a1f861f403d8

Google’s ADK is commonly recognized as one of the most advanced agentic frameworks, but it is occasionally omitted from “top 10” public lists for a few practical—not technical—reasons:

* Recency and Ecosystem Maturity: ADK is a relatively new entrant compared to frameworks like Microsoft AutoGen, LangGraph, or CrewAI. Some rankings or lists prioritize frameworks with longer track records, larger user communities, or broader third-party tool integration. This can result in ADK being left off generalized lists, especially those compiled before it gained significant traction.
* Subjectivity in List Construction: Many "top 10" lists are subjective and based on the creator’s experience, biases, or what frameworks are most widely adopted or requested by their audience. For example, crewAI or Atomic Agents might be included due to ease of use or open-source popularity, even if ADK is technically more advanced in areas like deployment or scalability.
* List Focus or Criteria: Some lists intentionally focus on Python-first frameworks, open source tools, or platforms with exceptional community support or ease of prototyping (which can favor older or lighter-weight frameworks). ADK, with its focus on enterprise readiness, strong cloud deployment, and memory/session management, may score lower in categories like “minimal code to hello world,” even though it excels in production deployment options.
* Rapid Evaluation Dynamics: Framework rankings change rapidly in agentic development; analysts repeatedly state that the “best” framework depends on the dimension (developer usability, deployment, orchestration features, ecosystem, etc.) most relevant to your use case.
* ADK is Frequently Compared Alongside Top Choices: In most expert or interactive comparisons, including code-level evaluations, ADK is analyzed side by side with AutoGen, LangGraph, OpenAI Agents SDK, and others. It is often discussed as a “top tier” choice, especially if your requirements center on cloud deployment, session management, or scaling.

In summary, Google’s ADK might not appear on a static “top 10” list due to its newness, the list-maker’s priorities, or temporary gaps in ecosystem maturity, but it is widely regarded as a top-level, production-ready multi-agent framework in 2025 and is featured prominently in direct expert comparisons

How does Google’s ADK enable flexible multi-agent system deployment compared to traditional models

How does Google’s ADK enable flexible multi-agent system deployment compared to traditional models

Google’s Agent Development Kit (ADK) enables flexible multi-agent system deployment in ways that surpass traditional models by focusing on modularity, orchestration, scalability, and production readiness:

Modular, Hierarchical Multi-Agent Architecture

ADK lets you group and compose multiple specialized agents into complex, hierarchical systems. This encourages independent upgrades, clear boundaries between agents, and reusability of components. You can easily swap or update individual agents without overhauling the whole system.

Traditional models often use monolithic or tightly coupled designs, where modifying one part typically requires changes throughout the system.

Flexible Orchestration & Delegation

ADK supports dynamic and static workflows: agents can coordinate tasks via orchestrator patterns (sequential, parallel, looping, etc.), and hand off responsibilities in real time based on current context and requirements.

Dynamic delegation is LLM-driven, allowing for adaptable decision-making as new information appears.

Traditional models tend to follow fixed pipelines or rigid execution orders, making it difficult to add dynamic control or data-driven task routing without major redevelopment.

Production-Grade Deployment & Cloud Integration

ADK includes built-in support for cloud platforms like Vertex AI, Kubernetes, and Cloud Run, as well as containerized deployment for both local and cloud environments. It enables developers to select the best-fit deployment model (pooled, siloed, hybrid) matching compliance, scale, and security requirements.

Traditional models usually lack such deployment tools, requiring custom solutions for scaling, orchestration, and state management.

Integrated Evaluation, Tooling, and Model Interoperability

ADK offers tools for workflow visualization, debugging, evaluation, and quality assurance, as well as easy integration with multiple foundation models and APIs (Google Gemini, Anthropic, Meta, etc.).

Traditional models seldom include such comprehensive frameworks out-of-the-box, so interoperability and evaluation are often manual add-ons.

Real-Time, User-Centric Experience

ADK allows for real-time streaming (audio, video), live context sharing among agents, and seamless handoff between agents and users. The orchestrator pattern presents a single interface to users while routing tasks internally to the most suitable agent—simplifying user experience.

Traditional models are typically limited to static, request-response cycles and lack native support for advanced, streaming, or interactive workflows.

In summary, Google’s ADK enables modular, orchestrated, and scalable multi-agent deployments with robust production, cloud integration, and developer tooling—making it faster and easier to build, test, and evolve complex agentic systems compared to traditional, monolithic, or inflexible agent frameworks

LangGraph and Google ADK (Agent Development Kit) are both AI agent frameworks but have different architectures, design philosophies, and use cases:

**Main Differences:**

* **Architectural Philosophy:**
  + ADK is a more full-stack application framework focused on model-driven orchestration and fluid abstraction. It prioritizes trusting the large language model (LLM) for decisions and routing dynamically during runtime.
  + LangGraph is a graph-based domain-specific language (DSL) that prioritizes explicit control, transparency, and determinism by structuring agent behavior as a graph with nodes and edges, emphasizing the graph over the model.
* **Code Example (RAG pipeline):**
  + ADK example:  
      
     python

agent = Agent(model="gemini-1.5",

tools=[my\_retriever],

instruction="Use the retriever to gather context before answering.")

response = agent.run("Tell me about photosynthesis")

* + LangGraph example:  
      
     python

graph = Graph()

graph.add\_node("retrieve", my\_retriever)

graph.add\_node("generate", my\_llm\_node)

graph.connect("retrieve", "generate")

result = graph.run({"query": "Tell me about photosynthesis"})

* LangGraph gives you programmatic control over each step in the agent workflow, while ADK abstracts much of this behind the model’s decisions.
* **Developer Experience and Tooling:**
  + ADK offers a UI for running, testing, and deploying agents to the cloud, making it easier to use for teams comfortable with trusting the LLM's routing and decision-making.
  + LangGraph is more code-centric, embedded into Python services, and suits teams that prefer strong observability, explicit control over flow, and managing state transitions in complex workflows.
* **State Management:**
  + ADK uses session-based short-term state and memory stores for long-term context and supports guardrails (pre/post LLM validation callbacks).
  + LangGraph offers detailed graph state management, enabling branching, error recovery, loops, and complex workflow orchestration with full transparency.
* **When to Use:**
  + Use **ADK** if you want a simpler, model-driven approach that is easier to set up, supports dynamic routing, and integrates well with Google Cloud ecosystem and out-of-the-box guardrails.
  + Use **LangGraph** if your application requires complex, stateful workflows, fine-grained control over agent logic, composability of tasks, and you prefer explicit graph-based models over model-driven abstractions.
* **Interoperability:** Both support the Model Context Protocol (MCP), enabling integration with external tools and interoperability between agents.

**Using A2A (Agent-to-Agent) with LangGraph:**

A2A is a Google protocol enabling AI agents to delegate tasks to other agents. In LangGraph, A2A can be integrated by explicitly modeling agent nodes in the graph that delegate sub-tasks to other agents using A2A communication. This fits well with LangGraph’s graph-based workflow control, allowing custom handling of multi-agent orchestration, delegation, error recovery, and state passing.

**Summary Table:**

| **Aspect** | **Google ADK** | **LangGraph** |
| --- | --- | --- |
|  |  |  |
| Philosophy | Model/LMM-driven orchestration | Graph-based, explicit workflow control |
| Complexity Suitability | Simple to moderate agent workflows | Complex, stateful, branching workflows |
| Developer Experience | UI tooling, cloud deployment, model-agnostic | Code-centric, embedded in Python, deterministic |
| State Management | Session-based short/long-term memory | Graph state, loops, branching, error recovery |
| Control & Transparency | Higher abstraction, LLM makes routing decisions | Full control over steps, flow, and state |
| When to Use | Quick setup, trusting LLM decisions | Need fine-grained, complex workflow control |
| A2A Use | Supports via the Google ecosystem and MCP | Explicit multi-agent delegation modeled in a graph. |

This side-by-side helps choose the right framework based on your agent complexity, control needs, and preferred development style[1](https://www.linkedin.com/pulse/googles-new-adk-langgraph-both-technical-deep-dive-kai-werther-n9xdf)3[5](https://langwatch.ai/blog/best-ai-agent-frameworks-in-2025-comparing-langgraph-dspy-crewai-agno-and-more)[10](https://www.linkedin.com/posts/an-pham-atl_ai-agentframeworks-langgraph-activity-7350569998651703297-rXjy)[11](https://blog.arcade.dev/agent-handoffs-langgraph-openai-google).

If you want, I can provide a basic code example for integrating A2A protocol with LangGraph as well. Would you like that?

1. <https://www.linkedin.com/pulse/googles-new-adk-langgraph-both-technical-deep-dive-kai-werther-n9xdf>
2. <https://duplocloud.com/langchain-vs-langgraph/>
3. <https://www.youtube.com/watch?v=XL6Fq3dY0sc>
4. <https://cloud.google.com/products/agent-builder>
5. <https://langwatch.ai/blog/best-ai-agent-frameworks-in-2025-comparing-langgraph-dspy-crewai-agno-and-more>
6. <https://pub.towardsai.net/multi-agent-systems-redefined-a1a73860a0f8>
7. <https://www.youtube.com/watch?v=FGew0KGWhR0>
8. <https://infinitelambda.com/compare-crewai-autogen-vertexai-langgraph/>
9. <https://www.reddit.com/r/LangChain/comments/1jw87c0/langgraph_google_adk_or_llamaindex_how_would_you/>
10. <https://www.linkedin.com/posts/an-pham-atl_ai-agentframeworks-langgraph-activity-7350569998651703297-rXjy>
11. <https://blog.arcade.dev/agent-handoffs-langgraph-openai-google>

Can I develop ADK agents without relying on Google Cloud services

Yes, you can develop ADK agents without relying on Google Cloud services. The Google Agent Development Kit (ADK) is designed to be flexible and supports development and local execution entirely on your own machine or environment. You can write and run your agent code locally using a Python environment, with no mandatory dependency on Google Cloud during development and testing.

However, ADK is optimized for integration within the Google Cloud ecosystem, especially for deploying and scaling agents with Vertex AI Agent Engine and accessing Gemini models and other Google Cloud tools. This optimization provides enhanced capabilities and a managed production runtime, but it is not a requirement for initial development or experimentation.

In summary:

* You can write, run, and test ADK agents fully locally without Google Cloud.
* Deployment to Google Cloud services like Vertex AI is optional and mainly for production scaling and advanced features.
* ADK supports local development environments using Python, including command line and development UIs.
* The open-source nature and design of ADK enable flexible deployment to various container runtimes or platforms beyond Google Cloud if desired.

This means you do not need to rely on Google Cloud services to develop ADK agents, but cloud integration provides additional benefits if you choose to use it later[1](https://developers.googleblog.com/en/agent-development-kit-easy-to-build-multi-agent-applications/)[2](https://github.com/google/adk-samples/issues/30).

Add to follow-up

Check sources

1. <https://developers.googleblog.com/en/agent-development-kit-easy-to-build-multi-agent-applications/>
2. <https://github.com/google/adk-samples/issues/30>
3. <https://cloud.google.com/blog/topics/developers-practitioners/tools-make-an-agent-from-zero-to-assistant-with-adk>
4. <https://www.googlecloudcommunity.com/gc/Cloud-Product-Articles/Using-Google-s-Agent-Development-Kit-ADK-with-MCP-Toolbox-and/ta-p/898512>
5. <https://cloud.google.com/blog/products/ai-machine-learning/build-and-manage-multi-system-agents-with-vertex-ai>
6. <https://www.reddit.com/r/LangChain/comments/1jvsvs7/just_did_a_deep_dive_into_googles_agent/>
7. <https://google.github.io/adk-docs/tools/google-cloud-tools/>
8. <https://codelabs.developers.google.com/instavibe-adk-multi-agents/instructions>

Yes, you can write and run Google ADK (Agent Development Kit) code without using the full Google Cloud environment. ADK supports local development entirely on your machine using a local IDE such as VS Code, PyCharm, or IntelliJ IDEA with Python 3.9+ or Java 17+. You install the ADK package via pip and run your agents locally in your development environment. This approach is recommended for internal development and testing before deploying to any cloud environment.

Specifically, you can:

* Install the ADK Python package locally in a virtual environment (pip install google-adk).
* Write your agent code and run it locally in your terminal or IDE.
* Use a .env file to configure required API keys and environment variables.
* The ADK package includes tooling for local execution and a browser-based dev UI for testing and debugging your agents locally.

Therefore, you do not need to be directly connected to or dependent on the Google Cloud environment to develop and run ADK agents. The Google Cloud environment is primarily relevant when deploying and scaling the agents for production use, but local development and experimentation are fully supported without it[1](https://www.siddharthbharath.com/the-complete-guide-to-googles-agent-development-kit-adk/)[2](https://google.github.io/adk-docs/get-started/quickstart/)[3](https://cloud.google.com/vertex-ai/generative-ai/docs/agent-development-kit/quickstart).

1. <https://www.siddharthbharath.com/the-complete-guide-to-googles-agent-development-kit-adk/>
2. <https://google.github.io/adk-docs/get-started/quickstart/>
3. <https://cloud.google.com/vertex-ai/generative-ai/docs/agent-development-kit/quickstart>
4. <https://developers.googleblog.com/en/agent-development-kit-easy-to-build-multi-agent-applications/>
5. <https://google.github.io/adk-docs/agents/models/>
6. <https://www.reddit.com/r/LocalLLaMA/comments/1jvsvzj/just_did_a_deep_dive_into_googles_agent/>
7. <https://www.kubiya.ai/blog/agent-development-kit>
8. <https://www.googlecloudcommunity.com/gc/Cloud-Product-Articles/Using-Google-s-Agent-Development-Kit-ADK-with-MCP-Toolbox-and/ta-p/898512>
9. <https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/develop/adk>