Agent2Agent Protocol in Google Agent Development Kit: A Comprehensive Guide to Building Real-Time Agentic Applications

1. Introduction: The Rise of Interoperable AI Agents

The landscape of artificial intelligence is rapidly evolving, with increasingly complex tasks demanding sophisticated solutions that often surpass the capabilities of single, monolithic AI systems. The need for collaboration among specialized AI agents has become paramount to tackle intricate problems requiring diverse expertise and functionalities. This necessity has spurred the development of frameworks that facilitate the creation and management of multi-agent systems. Among these, the Google Agent Development Kit (ADK) stands out as a modular and production-ready framework specifically designed for building applications powered by large language models (LLMs).[1] The ADK provides developers with the flexibility to construct powerful and interoperable multi-agent applications, leveraging built-in structures for state management, callbacks, streaming, and structured input/output.[1] Its open-source nature and the fact that it underpins agents within Google products like Agentspace and the Customer Engagement Suite underscore its maturity and robustness, offering a reliable foundation for developing advanced agentic solutions.[1] Furthermore, the modular design of ADK allows for the creation of specialized agents, a crucial aspect of building effective multi-agent systems where each agent can focus on specific tasks, thereby enhancing the overall efficiency and capability of the system.[1]

Addressing the challenge of enabling seamless communication and collaboration between these diverse agents, Google has introduced the Agent2Agent (A2A) Protocol.[1] This open, vendor-neutral standard is designed to facilitate easy and consistent interaction between AI agents across various platforms and frameworks.[1] A2A tackles the critical issue of fragmented AI ecosystems by providing a common language that allows agents built with different technologies and by different vendors to communicate and collaborate effectively.[1] The vendor-neutral nature of A2A, supported by a growing list of over 50 industry partners [2], signals a strong potential for widespread adoption, promising the development of a rich ecosystem of interoperable agents capable of addressing complex challenges through coordinated actions.

This report aims to provide a comprehensive guide to understanding and implementing the Agent2Agent Protocol using the Google Agent Development Kit for building real-time agentic applications. It will delve into the core principles and architecture of A2A, provide step-by-step instructions for setting up the development environment, illustrate the implementation of a sample real-time application, and discuss best practices for ensuring code correctness and efficiency in such systems.

2. Demystifying the Agent2Agent Protocol

The Agent2Agent (A2A) Protocol is built upon several core principles that guide its design and functionality, primarily aiming to foster seamless collaboration between autonomous AI agents.[6] A fundamental principle is the embrace of inherent agentic capabilities, allowing agents to collaborate in their natural, often unstructured modalities, even when they lack shared memory, tools, or context.[6] This design choice facilitates true multi-agent scenarios without restricting an agent's interaction to merely calling another as a tool.[7] To ensure ease of integration and broad compatibility, A2A leverages established and widely adopted web and internet standards such as HTTP, RESTful principles, OAuth for security, Server-Sent Events (SSE) for streaming, and JSON-RPC for message formatting.[7] This reliance on familiar technologies lowers the barrier to entry for developers and simplifies integration with existing IT infrastructures.[7] Recognizing that many real-world agentic applications involve tasks that can take significant time to complete, A2A is explicitly designed to handle asynchronous and long-running operations, providing mechanisms for managing communication and tracking the state of tasks that might span minutes, hours, or even longer.[7] Furthermore, A2A is modality agnostic, meaning it is not limited to text-based communication but supports diverse data types, including audio and video streaming, ensuring it can accommodate a wide range of applications and agent capabilities.[7] The protocol's focus on task-oriented communication ensures that interactions between agents are structured around specific units of work, facilitating clarity and efficient coordination.[11]

The architecture of the A2A Protocol revolves around three primary components: the Agent Card, the A2A Server, and the A2A Client.[6] The **Agent Card** serves as a public metadata file, typically hosted at /.well-known/agent.json on an agent's server. It acts as a digital identity and capability statement for an AI agent, akin to a standardized business card.[6] This JSON file contains crucial metadata, including the agent's identification (name, version), its capabilities (what tasks it can perform or services it offers), the endpoint(s) through which other agents can communicate with it (e.g., API addresses), and any requirements for interaction, such as security protocols or expected data formats.[6] The primary function of the Agent Card is discovery, allowing potential clients (other agents or systems) to find suitable agents and understand their basic offerings and how to connect.[6] The **A2A Server** is a component that typically runs alongside the agent providing a service or capability, often referred to as the 'remote agent' in an interaction.[6] Its main responsibilities include listening for incoming requests formatted according to the A2A protocol specification, processing these valid requests, interpreting them, and interfacing with the underlying AI agent logic to execute the requested task.[6] The A2A Server also manages the lifecycle of the task, which is particularly important for long-running operations, and sends back responses, status updates, or final results to the requesting client, again using the standardized A2A format.[6] Essentially, the A2A Server acts as the host and execution handler for an agent's capabilities within the A2A framework.[6] The **A2A Client** is the component that initiates the interaction.[6] This could be another AI agent, an application, or a user-facing interface that needs to leverage the capabilities of a remote agent.[6] The A2A Client's role involves discovering the appropriate agent and understanding how to interact with it (using Agent Cards), formulating a request message that adheres to the A2A protocol's standards, specifying the desired task and providing necessary input data, communicating this request to the target agent's A2A Server endpoint, and finally, receiving and processing the response (or updates) from the A2A Server.[6] The A2A Client is thus the initiator and consumer of services within an agent-to-agent interaction.[6]

Within the A2A framework, several key components facilitate the interaction between agents.[6] A **Task** represents the central unit of work. A client initiates a task by sending a message, and each task has a unique ID and progresses through various states such as submitted, working, input-required, completed, failed, and canceled.[6] Communication between the client and the server occurs through **Messages**, which represent turns in the interaction. Each message contains one or more **Parts**, which are the fundamental content units within a Message or an Artifact. Parts can be of different types, including TextPart for plain text, FilePart for files (either inline or via a URI), and DataPart for structured JSON data like forms.[6] **Artifacts** represent the outputs generated by the agent during a task, such as generated files or structured data, and they also contain Parts.[6] For long-running tasks, A2A supports **Streaming** capabilities using Server-Sent Events (SSE). Clients can subscribe to receive real-time updates on the task's status (TaskStatusUpdateEvent) and any generated artifacts (TaskArtifactUpdateEvent).[6] Additionally, servers that support **Push Notifications** can proactively send task updates to a client-provided webhook URL, allowing for efficient, event-driven communication.[6]

The typical flow of communication between agents using the A2A protocol begins with **Discovery**.[6] The A2A Client first fetches the Agent Card from the A2A Server's well-known URL (/.well-known/agent.json) to understand the server's capabilities and how to interact with it.[6] Next, the client initiates the interaction with **Initiation** by sending a tasks/send request for a standard task or a tasks/sendSubscribe request for a streaming task to the server's endpoint.[6] This request contains the initial user message and a unique identifier for the task.[6] During **Processing**, if the task supports streaming, the server sends SSE events back to the client, providing real-time status updates and any generated artifacts.[6] For non-streaming tasks, the server processes the request and returns the complete Task object once finished.[6] Optionally, during **Interaction**, if the server requires more input from the client, the task will enter an input-required state, and the client can send subsequent messages using the same Task ID.[6] Finally, the task reaches **Completion**, which can be a completed, failed, or canceled state, signaling the end of the interaction.[6] This structured communication lifecycle ensures a predictable and manageable interaction pattern between agents, facilitating development and debugging.[6]

3. Setting the Stage: Development Environment for Google ADK

To begin building agentic applications leveraging the Agent2Agent Protocol with the Google Agent Development Kit, it is essential to set up the appropriate development environment. This involves ensuring that the necessary software and libraries are installed and configured correctly. The primary prerequisites include having Python 3.10 or a later version installed on your system, as indicated in a tutorial for ADK.[14] Additionally, you will need access to a terminal or command prompt to execute installation commands and manage your project.

The first crucial step is to install the Google ADK and its associated dependencies. This is typically done using pip, the Python package installer. A common set of libraries required for building web-based agentic applications with ADK, especially those involving A2A communication and user interfaces, includes google-adk, litellm, fastapi, uvicorn, httpx, pydantic, openai, and streamlit.[1] To install these, you would generally execute the command pip install google-adk litellm fastapi uvicorn httpx pydantic openai streamlit in your terminal. This command downloads and installs the ADK framework along with other useful libraries. The litellm library facilitates the use of various large language models beyond just Google's offerings, such as OpenAI's models, showcasing ADK's flexibility.[1] FastAPI and uvicorn are used for building and running the API endpoints for your agents, which is essential for A2A communication.[1] Httpx is an asynchronous HTTP client that can be used by agents acting as A2A clients to make requests to other agents.[1] Pydantic is used for data validation and schema definition, ensuring that the data exchanged between agents is structured correctly.[1] Openai is the Python library for interacting with OpenAI's models if you choose to use them, and streamlit is a library for creating interactive web UIs, which can be useful for testing and demonstrating your agentic applications.[1]

After installing the necessary libraries, you will need to configure authentication for accessing the language models that your agents will use. ADK offers the flexibility to work with different model providers, including Google's Gemini and third-party models via LiteLLm.[1] The authentication process will depend on the specific model provider you choose. For instance, if you are using OpenAI models, you will typically need to set up an API key and configure it in your environment, possibly by setting an environment variable like OPENAI_API_KEY.[1] If you are using Google's Gemini models through Vertex AI, you will need a Google Cloud project and will need to set up and authenticate using the Google Cloud CLI.[25] This often involves setting environment variables such as GOOGLE_CLOUD_PROJECT and GOOGLE_CLOUD_LOCATION, and potentially setting GOOGLE_GENAI_USE_VERTEXAI to True.[25] The ADK documentation provides detailed instructions for configuring authentication for various scenarios.[25]

Finally, it is important to organize your project files in a logical structure. While the specific structure can vary depending on the complexity of your application, a basic structure often involves separate files for the main agent logic (agent.py), any data schemas (schemas.py), utility functions for A2A clients and servers (a2a_client.py, a2a_server.py), and potentially a dedicated directory (e.g., .well-known) to host the agent.json file (Agent Card) that advertises the agent's capabilities.[1] This well-defined project structure promotes better code organization, maintainability, and collaboration, making it easier to manage the different components of your agentic application.[1]

4. Building a Real-Time Agentic Application: A Practical Implementation

To illustrate the practical application of the Agent2Agent Protocol with the Google ADK, let's consider a scenario involving a real-time news summarization system.[7] In this system, a user submits a query, and the system, leveraging multiple specialized agents, retrieves relevant news articles and then summarizes them, streaming the summaries back to the user as they become available. This scenario highlights the collaborative potential of A2A and the real-time communication capabilities of ADK.

The system will consist of three main agents: the **NewsRetrieverAgent**, the **NewsSummarizerAgent**, and the **HostAgent**.[1] The NewsRetrieverAgent will be responsible for taking a user query and fetching relevant news articles using a tool or API (this agent's implementation details are outside the scope of A2A but are necessary for the overall scenario). The NewsSummarizerAgent will act as an A2A server, responsible for taking a list of news articles and generating concise summaries for each. The HostAgent will function as an A2A client, receiving the user query, orchestrating the interaction between the NewsRetrieverAgent and the NewsSummarizerAgent, and managing the streaming of summarized results back to the user.

Let's focus on the implementation of the **NewsSummarizerAgent** using Google ADK. This agent will act as an A2A server, exposing an endpoint to receive news articles and return their summaries. We can define this agent using the Agent class from google.adk.agents, potentially using a LiteLlm model for the summarization task.[3] The core logic of the agent will reside in its execute method, which will take the incoming news article(s) and use the language model to generate summaries. To make this agent accessible via A2A, we will need to implement a /run endpoint using FastAPI. We can utilize the create_app utility, as demonstrated in examples [1], to wrap our ADK agent with a FastAPI application. This endpoint will receive A2A requests, typically containing the news article content as a TextPart within a Message.[6] The agent will then process the article, generate a summary (also as a TextPart within a Message or an Artifact), and return it in the A2A response.[6] To enable real-time communication, the NewsSummarizerAgent can support Server-Sent Events (SSE) for streaming the summaries back to the client as they are generated.[6] This would involve using the tasks/sendSubscribe method on the client side and implementing the server-side logic to push updates as they become available. Finally, we need to create and deploy an agent.json file (Agent Card) for the NewsSummarizerAgent. This file will advertise the agent's capability to summarize news articles, along with its endpoint URL and any necessary authentication details, allowing the HostAgent to discover and interact with it.[6]

The **HostAgent** will act as an A2A client. Its role is to receive a user's news query, send it to the NewsRetrieverAgent (implementation not detailed here), and then take the retrieved news articles and send them to the NewsSummarizerAgent for summarization. The HostAgent can be implemented using Google ADK or even a standard HTTP client like httpx.[1] The first step for the HostAgent is agent discovery. It will need to fetch and parse the Agent Card of the NewsSummarizerAgent to learn its capabilities and endpoint.[6] Once the HostAgent has retrieved the news articles, it will formulate an A2A request to the NewsSummarizerAgent's /run endpoint. This request will include the news articles, likely as TextPart objects within a Message.[6] If the NewsSummarizerAgent supports streaming, the HostAgent will use the tasks/sendSubscribe method to initiate the summarization task and will then need to handle the incoming Server-Sent Events (SSE) containing the summaries.[6] The

HostAgent will then be responsible for taking these summaries and streaming them back to the end user in real-time.

The following table summarizes the key components and configurations of the A2A Server (NewsSummarizerAgent) and the A2A Client (HostAgent) in this example:

Feature	**NewsSummarizerAgent (Server)**	**HostAgent (Client)**
Name	news_summarizer_agent	host_news_agent
Description	Summarizes news articles.	Orchestrates news retrieval and summarization.
Endpoint	http://localhost:8001/run (example)	N/A (Acts as a client initiating requests)
Agent Card URL	http://localhost:8001/.well-known/agent.json (example)	N/A
Capabilities	Summarization of text (advertised in Agent Card)	Orchestration, discovery of summarization capabilities
Input Message	TextPart containing the news article content	User query (initially), potentially TextPart with retrieved articles
Output Message	TextPart containing the summary of the article	Aggregated summaries (streamed back to user)
Real-time Communication	Supports streaming of summaries via SSE	Handles SSE events from summarizer

This example illustrates how the Agent2Agent Protocol, in conjunction with the Google Agent Development Kit, can be used to build a real-time collaborative application where specialized agents communicate and work together to fulfill a user's request, providing a dynamic and interactive experience.

5. Ensuring Robustness: Code Correctness and Error Handling

Building robust agentic applications with the Agent2Agent Protocol and Google ADK requires careful attention to code correctness and the implementation of comprehensive error handling mechanisms. One crucial aspect of ensuring code correctness is the validation of input and output data. Libraries like Pydantic, which are commonly used in ADK projects [1], allow developers to define data schemas that can be used to automatically validate the structure and types of data being exchanged between agents.[1] By ensuring that the data conforms to a predefined structure, the likelihood of errors arising from malformed or unexpected input is significantly reduced. This is particularly important in distributed systems where agents might be developed independently and might have different expectations about data formats.

Implementing robust error handling is equally critical for building reliable agentic applications. Both client and server agents should be designed to anticipate and gracefully handle potential failures, such as network issues, unexpected responses from other agents, or errors within their own processing logic.[22] The A2A protocol itself defines task states like "failed" to indicate when an unrecoverable error has occurred.[22] Developers should leverage these states and implement appropriate error handling within their agent code to catch exceptions, log errors, and potentially attempt recovery or notify the initiating agent about the failure. For instance, if the NewsSummarizerAgent encounters an issue while trying to summarize an article, it should set the task status to "failed" and include an informative error message in the response to the HostAgent. The HostAgent, in turn, should be programmed to handle such failure responses, perhaps by logging the error or trying to re-submit the task.

When dealing with real-time streaming and multiple concurrent tasks, as is common in agentic applications, managing asynchronous operations and potential concurrency issues becomes vital. Python's async and await keywords, which are often used in ADK projects [1], provide a powerful way to handle asynchronous operations efficiently without blocking the main execution thread. This is crucial for maintaining the responsiveness of agents that need to interact with multiple other agents or handle streaming data. Developers should carefully design their asynchronous workflows to avoid race conditions and ensure that shared resources are accessed in a thread-safe manner if concurrency is involved.

Finally, security is a paramount concern, especially when deploying agentic applications in enterprise environments. Considerations for secure communication and data exchange between agents should be integrated into the design from the outset. This includes using HTTPS for all communication to encrypt data in transit, implementing appropriate authentication mechanisms to verify the identity of agents interacting with each other (as hinted in the Agent Card specification [6]), and potentially encrypting sensitive data at rest. By addressing these aspects of code correctness and error handling, developers can build robust and reliable agentic applications using the Agent2Agent Protocol and Google ADK.

6. Testing and Validation: Ensuring Seamless Agent Interactions

Thorough testing and validation are essential to ensure that agentic applications built with the Agent2Agent Protocol and Google ADK function correctly and provide a seamless interaction experience. The ADK provides several built-in developer tools that can be leveraged for local testing and debugging. One such tool is the adk web command, which launches an interactive, browser-based development UI.[1] This UI allows developers to interact with their agents in a chat-like interface, inspect the agent's state, and step through the execution flow, making it easier to identify and resolve issues during development.[1]

In addition to using the developer UI, writing unit and integration tests is crucial for validating the functionalities of individual agent components and the communication flows between them.[3] Unit tests should focus on verifying the behavior of individual functions or methods within an agent, while integration tests should examine how different agents interact with each other over the A2A protocol. For our news summarization example, unit tests could verify that the NewsSummarizerAgent correctly generates summaries for given articles, and integration tests could ensure that the HostAgent can successfully discover the NewsSummarizerAgent, send it articles, and receive the summarized results. The ADK provides features and utilities that can aid in writing and running these tests.[3]

When dealing with real-time interactions, such as the streaming of summaries in our example, techniques for simulating these scenarios and monitoring the message exchanges can be invaluable. Developers might use tools like network sniffers to observe the HTTP traffic between agents, ensuring that the A2A protocol is being followed correctly and that the messages and SSE events are being exchanged as expected. Additionally, leveraging ADK's logging capabilities to record the details of agent interactions, including the content of messages and the timing of events, can help in identifying and resolving issues related to timing, data serialization, and protocol compliance. By employing a combination of developer tools, automated testing, and monitoring techniques, developers can ensure the correctness and reliability of their agentic applications built with the Agent2Agent Protocol and Google ADK.

7. Best Practices for Designing and Implementing A2A

Applications with Google ADK

Designing and implementing effective agentic applications using the Agent2Agent Protocol and Google ADK involves adhering to certain best practices that can enhance the efficiency, maintainability, and scalability of the system. One fundamental practice is to define clear and concise Agent Cards for each agent.[6] The Agent Card serves as the primary means for an agent to advertise its capabilities to other agents. Therefore, it should accurately and comprehensively describe what the agent can do, its endpoint URL, and any requirements for interaction, such as authentication methods or expected data formats. A well-defined Agent Card makes it easier for other agents to discover and interact with the agent effectively, fostering a more dynamic and efficient multi-agent ecosystem.

Efficient task management and orchestration are also crucial in multi-agent systems.[1] Developers should carefully design the workflows for complex tasks, breaking them down into smaller, manageable sub-tasks that can be delegated to specialized agents. The orchestration logic, which determines how these sub-tasks are assigned and how the results are aggregated, should be well-defined to ensure that the overall goal is achieved efficiently. This might involve using sequential, parallel, or even more complex hierarchical structures of agents, as supported by ADK.[1]

When exchanging messages and artifacts between agents, it is important to choose the appropriate part types for different data modalities.[6] The A2A protocol supports TextPart for plain text, FilePart for binary data, and DataPart for structured JSON. Selecting the most semantically appropriate and efficient part type for the data being exchanged ensures seamless interoperability and avoids unnecessary data conversions.

Finally, when designing and implementing A2A-based applications with Google ADK, developers should consider the long-term aspects of scalability, maintainability, and deployment.[1] ADK offers flexibility in terms of deployment options, including running agents locally, containerizing them for deployment on platforms like Cloud Run, or scaling them using Vertex AI Agent Engine.[1] Designing the application with scalability in mind from the beginning, and following good software engineering practices to ensure maintainability, will contribute to the robustness and longevity of the agentic system.

8. Conclusion: The Future of Collaborative AI with Agent2Agent and ADK

In conclusion, the Agent2Agent Protocol, in conjunction with the Google Agent Development Kit, represents a significant step forward in enabling seamless collaboration and interoperability between AI agents.[6] By providing an open, vendor-neutral standard for communication, A2A addresses the growing need for diverse AI systems to work together effectively, regardless of their underlying frameworks or vendors. The Google ADK offers a robust and flexible platform for building these interoperable agents, providing developers with the tools and structures necessary to create sophisticated multi-agent applications.

The real-time news summarization example illustrates the potential of this technology to create dynamic and interactive experiences where specialized agents collaborate to fulfill complex user requests. The principles and practices discussed in this report, ranging from understanding the core concepts of A2A to setting up the development environment, implementing agents, ensuring robustness through error handling and validation, and adhering to best design practices, provide a comprehensive guide for developers looking to leverage these powerful tools.

Looking ahead, the field of agent interoperability is expected to continue to evolve. Future trends and potential enhancements in the A2A protocol, such as more formalized authorization schemes, dynamic user experience negotiation within ongoing tasks, and improved streaming performance [11], as well as advancements in agent discovery and task lifecycle management [6], promise to further enhance the capabilities and usability of multi-agent systems. As AI continues to permeate various aspects of our lives and work, the ability for AI agents to collaborate seamlessly will be crucial in unlocking new levels of automation, efficiency, and problem-solving capabilities. The Agent2Agent Protocol and the Google Agent Development Kit are at the forefront of this exciting future, paving the way for a new era of collaborative artificial intelligence.

Works cited

1. Agent Development Kit (ADK): A Guide With Demo Project | DataCamp, accessed on April 15, 2025, https://www.datacamp.com/tutorial/agent-development-kit-adk
2. Build and manage multi-system agents with Vertex AI | Google Cloud Blog, accessed on April 15, 2025, https://cloud.google.com/blog/products/ai-machine-learning/build-and-manage-multi-system-agents-with-vertex-ai
3. Agent Development Kit: Making it easy to build multi-agent ..., accessed on April 15, 2025, https://developers.googleblog.com/en/agent-development-kit-easy-to-build-mul

ti-agent-applications/

4. Agent Development Kit - Google, accessed on April 15, 2025, https://google.github.io/adk-docs/
5. google/adk-python: An open-source, code-first Python toolkit for building, evaluating, and deploying sophisticated AI agents with flexibility and control. - GitHub, accessed on April 15, 2025, https://github.com/google/adk-python
6. google/A2A: An open protocol enabling communication ... - GitHub, accessed on April 15, 2025, https://github.com/google/A2A
7. Announcing the Agent2Agent Protocol (A2A) - Google for Developers Blog, accessed on April 15, 2025, https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/
8. Home - Google, accessed on April 15, 2025, https://google.github.io/A2A/
9. Google Introduces Agent-to-Agent Protocol to Enhance AI Interoperability - Binance, accessed on April 15, 2025, https://www.binance.com/en/square/post/04-14-2025-google-introduces-agent-to-agent-protocol-to-enhance-ai-interoperability-22898731392850
10. Google Launches Agent2Agent Protocol to Unify AI Agents Communication - Stan Ventures, accessed on April 15, 2025, https://www.stanventures.com/news/google-launches-agent2agent-protocol-to-unify-ai-agents-communication-2421/
11. What is The Agent2Agent Protocol (A2A) and Why You Must Learn It Now - Hugging Face, accessed on April 15, 2025, https://huggingface.co/blog/lynn-mikami/agent2agent
12. Google's Agent2Agent (A2A) protocol: A new standard for AI agent collaboration - Wandb, accessed on April 15, 2025, https://wandb.ai/onlineinference/mcp/reports/Google-s-Agent2Agent-A2A-protocol-A-new-standard-for-AI-agent-collaboration--VmlldzoxMjIxMTk1OQ
13. Google's Agent2Agent Protocol (A2A) - Blockchain Council, accessed on April 15, 2025, https://www.blockchain-council.org/ai/googles-agent2agent-protocol/
14. Using Google's Agent Development Kit and Agent2Agent - Wandb, accessed on April 15, 2025, https://wandb.ai/gladiator/Google-Agent2Agent/reports/Tutorial-Using-Google-s-Agent2Agent-A2A-protocol--VmlldzoxMjIyODEwOA
15. Agent2Agent + (MCP to Tool) in Multi-Agent AI - YouTube, accessed on April 15, 2025, https://www.youtube.com/watch?v=DAQ6msUVOp0
16. A2A and MCP: Start of the AI Agent Protocol Wars? - Koyeb, accessed on April 15, 2025, https://www.koyeb.com/blog/a2a-and-mcp-start-of-the-ai-agent-protocol-wars
17. Google just Launched Agent2Agent, an Open Protocol for AI agents to Work Directly with Each Other - Maginative, accessed on April 15, 2025, https://www.maginative.com/article/google-just-launched-agent2agent-an-open-protocol-for-ai-agents-to-work-directly-with-each-other/
18. What Is the A2A (Agent2Agent) Protocol and How It Works - Descope, accessed on April 15, 2025, https://www.descope.com/learn/post/a2a
19. Using Google's Agent Development Kit and Agent2Agent - Wandb, accessed on

April 15, 2025, https://wandb.ai/gladiator/Google-Agent2Agent/reports/Using-Google-s-Agent-Development-Kit-and-Agent2Agent--VmlldzoxMjIyODEwOA

20. Vertex AI Agent Builder | Google Cloud, accessed on April 15, 2025, https://cloud.google.com/products/agent-builder
21. Google Dropped "A2A": An Open Protocol for Different AI Agents to Finally Play Nice Together? : r/LocalLLaMA - Reddit, accessed on April 15, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1jvuitv/google_dropped_a2a_an_open_protocol_for_different/
22. How the Agent2Agent Protocol (A2A) Actually Works: A Technical ..., accessed on April 15, 2025, https://www.blott.studio/blog/post/how-the-agent2agent-protocol-a2a-actually-works-a-technical-breakdown
23. Just did a deep dive into Google's Agent Development Kit (ADK). Here are some thoughts, nitpicks, and things I loved (unbiased) - Reddit, accessed on April 15, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1jvsvzj/just_did_a_deep_dive_into_googles_agent/
24. How to Build Your First Agent with Google Agent Development Kit (ADK) - YouTube, accessed on April 15, 2025, https://www.youtube.com/watch?v=rLSj47zkTa8
25. Quickstart: Build an agent with the Agent Development Kit | Generative AI on Vertex AI, accessed on April 15, 2025, https://cloud.google.com/vertex-ai/generative-ai/docs/agent-development-kit/quickstart
26. Quickstart - Agent Development Kit - Google, accessed on April 15, 2025, https://google.github.io/adk-docs/get-started/quickstart/
27. Google Agent Development Kit is HERE! Learn It All in One Video - YouTube, accessed on April 15, 2025, https://www.youtube.com/watch?v=TONBnWCsoTU
28. Build Your First AI Agent With Google's new Agent Development Kit! - YouTube, accessed on April 15, 2025, https://www.youtube.com/watch?v=b8Pojbq4Qgk
29. Develop an Agent Development Kit agent | Generative AI on Vertex AI - Google Cloud, accessed on April 15, 2025, https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/develop/adk