# Google Agent Development Kit (ADK): A Technical Overview

## I. Executive Summary:

The Google Agent Development Kit (ADK) is presented as an open-source Python toolkit engineered to facilitate the construction, evaluation, and deployment of sophisticated artificial intelligence agents with a high degree of flexibility and control.[1] This toolkit is designed with a code-first philosophy, emphasizing direct programmatic definition of agent behavior, orchestration, and utilization of tools.[1] The ADK demonstrates a strong emphasis on integration with Google Cloud services, particularly its Gemini family of large language models.[1] The architecture encompasses several key modules, including those for managing agents, executing code, leveraging external tools, integrating language models, planning agent actions, handling memory, and collecting telemetry data.

## II. Introduction to the Google Agent Development Kit (ADK):

### What is ADK?

The Agent Development Kit (ADK) is a versatile and modular framework intended for the development and deployment of AI agents.[3] A fundamental characteristic of the ADK is its "code-first" nature, which empowers developers to define the operational logic, coordination mechanisms, and tool interactions of their agents directly through Python code.[1] This design choice suggests a development paradigm where developers comfortable with Python can exert significant control over the intricacies of their AI agents. The repeated emphasis on "code-first" in the documentation and the provision of Python-based examples underscore a design philosophy that prioritizes programmatic control. Developers who prefer to articulate logic through code will likely find this approach intuitive and powerful.

### Target Audience and Key Benefits:

The ADK is specifically tailored for developers who require a high level of control and adaptability in building advanced AI agents.[1] Utilizing this toolkit offers several advantages, including enhanced debugging capabilities, robust version control, and the ability to deploy agents across diverse environments, ranging from local machines to cloud infrastructure.[1] Furthermore, the ADK facilitates the creation of modular and scalable applications through the composition of multiple specialized agents organized in flexible hierarchies.[1] The explicit mention of "multi-agent systems" as a core capability across multiple sources [1] indicates that the ADK is architected to support complex applications that necessitate the collaboration of several distinct

agents. This could be a significant advantage for developers tackling intricate problems that can be decomposed into the responsibilities of interacting agents.

**Integration with Google Cloud and Gemini:**

A notable aspect of the ADK is its close integration with Google Cloud services, particularly the Gemini family of language models.[1] While the ADK is optimized for this ecosystem, it also provides support for other language models through the Vertex AI Model Garden and the LiteLLM library.[4] The inclusion of LiteLLM, which offers a standardized interface to over a hundred different language models [6], provides developers with the flexibility to experiment with or utilize models from various providers, including Anthropic, Meta, and Mistral AI.[4] This suggests that developers are not restricted to using only Google's language models and can select the most appropriate model for their specific application requirements.

**III. In-depth Analysis of ADK Modules:**

**A. Agents Module:**

This module serves as the foundation for constructing agents within the ADK framework, housing the essential classes required for their creation.[7] The BaseAgent class acts as the fundamental blueprint upon which all other agent types are built.[7]

**Agent Types:**

- **LlmAgent (and Agent):** These agent types are powered by Large Language Models (LLMs), which serve as their core mechanism for understanding natural language, performing reasoning, formulating plans, and deciding which tools to employ.[7] They are particularly well-suited for tasks that are inherently language-centric and require flexible decision-making.[7] The initial query provided an example of creating a basic LlmAgent instance, demonstrating the fundamental step in utilizing this agent type:

  ```
  Python
  from google.adk.agents import LlmAgent, RunConfig

  # Example of creating a simple LLM agent
  llm_agent = LlmAgent(llm=None) # You would typically provide an LLM here
  run_config = RunConfig()
  # llm_agent.run(..., config=run_config) # Example of how you might run the agent
  ```

  The LlmAgent appears to be the primary agent type for applications involving natural language processing, leveraging the capabilities of LLMs for core

functionalities such as reasoning and planning, which are central to many agent applications.

- **Workflow Agents:** These specialized agents are designed to control the execution flow of other agents in a predefined and deterministic manner, without relying on an LLM for managing this flow.[7] This makes them ideal for structured processes where the sequence of actions is known and needs to be predictable.
  - **SequentialAgent:** Executes a series of steps in a specific, linear order.[7]
  - **ParallelAgent:** Enables the concurrent execution of multiple steps.[7]
  - **LoopAgent:** Facilitates the repeated execution of a set of steps.[7]

  Workflow agents offer a method for orchestrating tasks with a high degree of predictability, which is valuable in scenarios where the sequence of operations must follow a specific order. This contrasts with the more dynamic and less deterministic nature of the LlmAgent, where the LLM plays a significant role in determining the next steps.
- **Custom Agents:** Developers can create highly tailored agents by directly extending the BaseAgent class. This allows for the implementation of unique operational logic, specific control flows, or specialized integrations that are not covered by the standard agent types.[7] This capability provides maximum adaptability for developers who need to implement very specific behaviors or integrate with systems that are not readily supported by the built-in agent types.

**Multi-Agent Systems:** The true potential of the ADK is often realized when different types of agents are combined to work together.[2] Complex applications can benefit from multi-agent architectures where LlmAgent instances handle intelligent, language-based tasks, workflow agents manage the overall process flow using established patterns, and custom agents provide specialized functionalities or rules required for unique integrations.[7]

**Table 1: Comparison of Agent Types:**

| Feature | LLM Agent (LlmAgent) | Workflow Agent | Custom Agent (BaseAgent subclass) |
| :---------- |...source |

The agents module, therefore, provides a comprehensive toolkit for constructing various types of agents, catering to a wide spectrum of application requirements, from those needing the advanced reasoning of LLMs to those requiring structured and predictable execution flows.

**B. Code Executors Module:**

This module is responsible for enabling agents to execute code as part of their workflow [from the initial query]. It includes several types of code executors, each

with its own characteristics and use cases.

**Executor Types:**

- **BaseCodeExecutor:** This likely serves as an abstract base class, defining the common interface and functionalities for all code executors within the ADK.
- **ContainerCodeExecutor:** This executor operates by running code within isolated containers [from the initial query]. This approach offers enhanced security and ensures a consistent execution environment by sandboxing the code and managing its dependencies. The isolation provided by containers is particularly important when dealing with untrusted code or operations that require a high degree of reproducibility.
- **UnsafeLocalCodeExecutor:** This executor allows code to be executed directly on the local system [from the initial query]. While this can be convenient for development and testing purposes, it presents potential security risks due to the lack of isolation. Therefore, its use should be approached with caution, especially in production environments where untrusted inputs might be involved. The absence of an isolated environment means that any vulnerabilities in the executed code could potentially affect the host system.
- **VertexAiCodeExecutor:** The presence of this executor suggests an integration with Google Cloud's Vertex AI platform for managed code execution. This could offer benefits such as scalability, security, and integration with other Vertex AI services.

The initial query provided an example of using the UnsafeLocalCodeExecutor:

Python

```
from google.adk.code_executors import UnsafeLocalCodeExecutor

# Example of creating a local code executor
code_executor = UnsafeLocalCodeExecutor()
# result = code_executor.execute("print('Hello from code executor')") # Example of executing code
```

In summary, the code_executors module offers a range of options for integrating code execution into agent workflows, balancing factors like flexibility, security, and integration with cloud services. The choice of which executor to use depends on the specific requirements of the application, particularly concerning the security and

isolation needs of the code being executed. It is worth noting that the official documentation for built-in code execution appears to be inaccessible in the provided material [8], which limits the depth of analysis possible based solely on these sources.

**C. Tools Module:**

The tools module provides reusable units of functionality that agents can leverage to interact with the external world.[9] These tools extend the core capabilities of agents beyond mere language processing and reasoning, enabling them to perform specific actions by executing developer-defined logic.[9] They function as modular components that can be easily integrated into an agent's repertoire.

**Key Tools:**

- **APIHubToolset (and potentially OpenAPI tools):** This toolset facilitates integration with external APIs that are defined using the OpenAPI specification.[9] By adhering to this widely adopted standard for describing APIs, the ADK allows agents to interact with a broad range of services, thereby significantly expanding their potential functionalities. This integration enables agents to access and utilize capabilities offered by various online services, such as databases, booking systems, and other specialized platforms.
- **FunctionTool:** This fundamental tool type allows developers to wrap standard Python functions or methods and expose them to agents as callable tools.[9] This provides a direct and flexible way to extend an agent's capabilities with custom Python code, tailored to the specific needs of the application. The example from [9] illustrates the creation of a FunctionTool to retrieve weather information, demonstrating how custom logic can be integrated into an agent's workflow. This approach offers considerable flexibility, as any desired functionality that can be implemented in Python can be made accessible to an agent through this mechanism.
- **LongRunningFunctionTool:** The name of this tool suggests support for functions that may take an extended period to execute, possibly involving asynchronous operations or processes that require significant computation time.
- **GoogleSearchTool:** This built-in tool enables agents to perform searches on Google and retrieve relevant snippets from the search results.[9] This capability allows agents to access and incorporate real-time information from the internet into their responses, which is crucial for tasks that require up-to-date knowledge or information that is not readily available within the agent's internal knowledge base. The example in [9] shows how to create and use the GoogleSearchTool to answer questions requiring web searches.

- **VertexAiSearchTool:** Similar to the VertexAiCodeExecutor, this tool suggests an integration with the Vertex AI platform for search functionalities, potentially allowing agents to leverage Vertex AI's search capabilities over specific datasets or knowledge bases.

In essence, the tools module equips developers with a diverse set of instruments that allow their agents to interact effectively with the world beyond their internal processing. By combining custom-defined functions, access to external APIs, and the ability to search the internet, agents built with the ADK can perform a wide array of tasks and provide comprehensive assistance.

**D. Events Module:**

The events module defines the structure for events that occur throughout an agent's lifecycle.[11] These events serve as the fundamental units of information flow within the ADK, representing every significant occurrence from the initial user input to the final response and all the intermediate steps.[11]

The Event class represents a specific occurrence within the agent's execution, capturing details such as user messages, agent replies, tool requests, tool results, state changes, and control signals.[11] The EventActions class likely defines actions that can be associated with an event, such as updating the agent's state or transferring control to another agent. Events play a crucial role in agent communication, triggering subsequent actions, monitoring the agent's behavior, and managing the overall flow of execution.[11] The initial query provided an example of creating a simple event:

Python

```
from google.adk.events import Event, EventActions

# Example of creating a simple event
my_event = Event(name="task_completed", data={"result": "success"})
# You would typically have a system that listens for and reacts to events
```

The event-driven architecture inherent in the ADK facilitates a decoupled communication model between different components of the system.[11] This means that one component can produce an event, and other interested components can react to it without having direct dependencies on the producer. This design promotes

modularity and scalability, as components can be developed and modified independently.

The is_final_response() method, associated with the Event class, is particularly important for identifying events that are suitable for displaying as the agent's complete output for a given turn.[11] This helps to filter out intermediate steps, such as tool calls or internal state updates, ensuring that only the final, user-facing message is presented.

In summary, the events module provides a robust and flexible mechanism for managing the intricate lifecycle of an agent and enabling seamless communication between its various components through a well-defined event-driven paradigm. It is important to note that the official documentation for the events module appears to be inaccessible in the provided research material.[13]

**E. Examples Module:**

The examples module likely serves as a repository of practical demonstrations and utilities designed to assist users in getting started with the ADK.[4] It is plausible that this module contains pre-built agent implementations, example configurations for tools, and sample data intended for illustrative purposes [from the initial query].

The primary goal of the examples module is likely to lower the barrier to entry for new users by providing concrete illustrations of how to utilize the various components of the ADK. By offering ready-to-use code snippets and configurations, this module can significantly accelerate the onboarding process and help developers quickly grasp the fundamental concepts and best practices for building their own agents. While the official documentation for this module is not available in the provided material [14], the presence of such a module is a common practice in software development kits to facilitate learning and adoption. Snippet [28], which demonstrates the use of function calls within an agent in the context of Vertex AI, could be indicative of the types of examples one might find within this module.

**F. Flows Module:**

The flows module likely contains predefined workflows or patterns that dictate the behavior of agents, especially those powered by Language Models.[3] These flows could represent common interaction sequences or multi-step reasoning processes that agents often need to perform [from the initial query]. The specific mention of llm_flows suggests that this module includes workflows that are specifically tailored

for agents that leverage the capabilities of LLMs.

Such a module could offer significant benefits by providing reusable templates for common agent interaction patterns. This would save developers considerable time and effort that would otherwise be required to define these patterns from scratch for each new agent or application.[15] By offering pre-designed workflows for tasks like information retrieval, summarization, or multi-turn conversations, the flows module could promote consistency and efficiency in agent development. While detailed information about this module is absent from the provided research material [3], the concept of predefined workflows is common in frameworks aimed at streamlining the development of complex applications. Snippets [15] and [16], though from different contexts (LLMFlows and PromptFlow), illustrate the general idea of defining and managing workflows for applications involving language models.

**G. Memory Module:**

The memory module is crucial for managing the state and knowledge of agents, providing components for both short-term and long-term information storage.[3] This allows agents to maintain context across interactions and retain information over time, enhancing their ability to provide relevant and coherent responses.

The module includes several implementations of memory services:

- **BaseMemoryService:** This likely serves as the abstract base class, defining the interface for all memory service implementations within the ADK.
- **InMemoryMemoryService:** This implementation provides a mechanism for storing and retrieving data in the agent's memory for the duration of the current interaction [from the initial query]. The initial query provided an example of using this service:

  Python
  ```
  from google.adk.memory import InMemoryMemoryService

  # Example of creating an in-memory memory service
  memory_service = InMemoryMemoryService()
  # memory_service.store("user_preferences", {"theme": "dark"})
  # preferences = memory_service.retrieve("user_preferences")
  ```

  The InMemoryMemoryService is well-suited for managing short-term conversational context within a single session due to its simplicity and speed. However, it is important to note that data stored in this manner is not persistent and will be lost once the session ends or the application terminates.

- **VertexAiRagMemoryService:** This implementation suggests an integration with Google Cloud's Vertex AI for providing memory services based on Retrieval-Augmented Generation (RAG) [from the initial query]. RAG involves retrieving relevant information from an external knowledge source and incorporating it into the language model's input to improve the quality and factual accuracy of its responses. This service likely allows agents to access and utilize external knowledge bases hosted on Vertex AI, enabling them to provide more informed and context-aware responses over extended periods.

In summary, the memory module offers different strategies for managing an agent's state and knowledge, catering to both immediate interaction needs and the requirement for long-term information retention. The choice of memory service depends on the specific application requirements regarding statefulness, persistence, and access to external knowledge sources. It is worth noting that the official documentation for the memory module is inaccessible in the provided material.[17]

**H. Models Module:**

The models module defines the interfaces and provides implementations for various Language Models that can power the intelligence of ADK agents.[6]

A key aspect of this module is its likely strong integration with Google's Gemini family of language models, as indicated by the presence of the Gemini class.[2] The initial query provided a potential example of creating a Gemini model instance:

Python

```
from google.adk.models import Gemini

# Example of (potentially) creating a Gemini model instance
# Note: This might require further configuration like API keys
# gemini_model = Gemini(model_name="gemini-pro")
# response = gemini_model.generate_content(prompt="Tell me a joke.")
```

The ADK internally uses the google-genai library for interacting with Gemini models.[6] There are two primary backends for utilizing Gemini through this library: Google AI Studio, which is best suited for rapid prototyping and development and typically requires an API key, and Vertex AI, which is recommended for production applications

and leverages Google Cloud's infrastructure, utilizing Application Default Credentials (ADC) for authentication.[6] The documentation also mentions specific Gemini models such as gemini-2.0-flash-exp and gemini-pro [2], each with its own characteristics and capabilities.

Beyond Gemini, the module likely includes foundational classes like BaseLlm, LLMRegistry, LlmRequest, and LlmResponse, which provide the necessary abstractions for interacting with different types of language models.[6] Furthermore, the ADK supports integration with a wide range of other language models from various providers through the LiteLLM library.[4] This is achieved by instantiating a LiteLlm wrapper class and passing it to the model parameter of the LlmAgent.[6]

In essence, the models module offers a flexible and comprehensive approach to integrating language models into ADK agents. The strong support for Google's Gemini models, coupled with the extensibility provided by LiteLLM, allows developers to select the most appropriate language model for their specific application needs and constraints. This central role of the models module underscores its importance in powering the intelligence behind ADK agents. It is important to note that the official documentation for the models module is inaccessible in the provided research material.[21]

**I. Telemetry Module:**

The telemetry module provides the necessary tools for collecting and reporting data regarding the execution of agents built with the ADK.[22] This data is invaluable for monitoring the performance of agents, debugging potential issues, and gaining insights into how the agents are being utilized.

The module likely includes various classes and functions, such as trace, trace_call_llm, and trace_tool_call, which are designed to facilitate the tracing of an agent's behavior during runtime [from the initial query]. This allows developers to follow the sequence of actions taken by an agent, including interactions with language models and the use of tools.

The ADK leverages the OpenTelemetry standard for collecting telemetry data, as evidenced by the dependency on "opentelemetry" in the ADK's pyproject.toml file [23] and the mention of OpenTelemetry support in the Vertex AI Agent Engine.[24] OpenTelemetry is a widely adopted standard that provides a unified way to collect and export telemetry data, such as traces, metrics, and logs. By adhering to this standard, the ADK ensures interoperability with a wide range of monitoring and

tracing tools, which is crucial for managing and observing agents deployed in production environments where observability is paramount.

The ability to collect and analyze telemetry data is essential for ensuring the reliability and maintainability of ADK agents in production. It allows developers to identify performance bottlenecks, diagnose errors, and understand usage patterns, enabling them to optimize their agents and provide a better user experience. The initial query provided an example of how telemetry might be used with a @trace_tool_call decorator:

Python

```
# Example of how you might use telemetry (specific usage depends on the implementation)
# from google.adk.telemetry import trace_tool_call
# @trace_tool_call
# def my_tool_function(...):
#     ...
```

In conclusion, the telemetry module plays a vital role in the lifecycle of ADK agents, providing the mechanisms necessary for monitoring their operation and ensuring their stability and effectiveness in real-world applications. It is worth noting that the official documentation for the telemetry module is inaccessible in the provided research material.[25]

**J. Planners Module:**

The planners module houses the components responsible for determining the sequence of actions that an agent should undertake to achieve a given objective [from the initial query]. This involves more than just reacting to immediate inputs; it requires the agent to formulate a plan of action to reach a desired outcome.

The module likely offers various planning strategies, ranging from simpler rule-based approaches to more sophisticated methods that might involve reasoning about the best course of action [from the initial query].

Key classes within this module include:

- **BasePlanner:** This likely serves as the abstract base class, defining the common interface for all planner implementations in the ADK.

- **BuiltInPlanner:** The name suggests a default or standard planning strategy that is provided as part of the ADK. This could offer a baseline level of planning capability for agents that do not require more specialized approaches.
- **PlanReActPlanner:** This class likely implements the ReAct (Reasoning and Acting) framework, a popular and effective technique for enabling agents to perform complex tasks by interleaving reasoning steps with actions.[26] The ReAct framework has been shown to improve an agent's ability to handle tasks that require both logical thinking and interaction with the environment. The initial query provided an example of creating a PlanReActPlanner:

```
Python
from google.adk.planners import PlanReActPlanner

# Example of creating a PlanReActPlanner
planner = PlanReActPlanner(llm=None, tools=) # You would typically provide an LLM and available tools
# plan = planner.plan(goal="Search the web and then write a summary.")
```

  The inclusion of the PlanReActPlanner indicates that the ADK supports advanced planning strategies that allow agents to engage in sophisticated reasoning and decision-making processes before and during the execution of their tasks. This capability can lead to more intelligent and effective agent behaviors, particularly in complex scenarios. It is important to note that the official documentation for the planners module is inaccessible in the provided research material.[27] Snippets [29], and [31], which mention "Planner" in different contexts (Google Ads, API reference), do not provide direct information about the ADK's planners module.

**IV. Conclusion:**

The Google Agent Development Kit (ADK) emerges as a robust and versatile framework for building sophisticated AI agents. Its emphasis on code-first development provides developers with a high degree of control and flexibility in defining agent behavior and integrating various functionalities. The modular architecture, encompassing components for agent management, code execution, tool utilization, language model integration, memory handling, planning, and telemetry, offers a comprehensive toolkit for addressing a wide range of AI application requirements.

The tight integration with Google Cloud services, particularly the Gemini family of language models, positions the ADK as a powerful platform for leveraging cutting-edge AI capabilities. Furthermore, the support for other language models

through Vertex AI Model Garden and LiteLLM enhances the toolkit's adaptability and allows developers to choose the most suitable model for their specific needs.

The ADK's focus on multi-agent systems suggests its suitability for tackling complex problems that can be effectively addressed through the collaboration of specialized agents. Features like the PlanReActPlanner indicate support for advanced planning strategies, enabling agents to reason and act in a more sophisticated manner. The inclusion of a telemetry module, built upon the OpenTelemetry standard, underscores the importance of observability and maintainability for production deployments.

While the provided research material indicates that some areas of the official documentation are currently inaccessible, the overall structure and functionality of the Google Agent Development Kit suggest a powerful and adaptable toolkit for developers looking to build advanced AI agent applications with a strong emphasis on programmatic control and integration with a rich ecosystem of tools and services.

**Works cited**


1. google/adk-docs: An open-source, code-first Python toolkit for building, evaluating, and deploying sophisticated AI agents with flexibility and control. - GitHub, accessed on April 10, 2025, https://github.com/google/adk-docs
2. 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 10, 2025, https://github.com/google/adk-python
3. Agent Development Kit - Google, accessed on April 10, 2025, https://google.github.io/adk-docs/
4. Agent Development Kit: Making it easy to build multi-agent applications, accessed on April 10, 2025, https://developers.googleblog.com/en/agent-development-kit-easy-to-build-multi-agent-applications/
5. Google Releases Agent Development Kit (ADK): An Open-Source AI Framework Integrated with Gemini to Build, Manage, Evaluate and Deploy Multi Agents - MarkTechPost, accessed on April 10, 2025, https://www.marktechpost.com/2025/04/09/google-releases-agent-development-kit-adk-an-open-source-ai-framework-integrated-with-gemini-to-build-manage-evaluate-and-deploy-multi-agents/
6. Models - Agent Development Kit - Google, accessed on April 10, 2025, https://google.github.io/adk-docs/agents/models/
7. Agents - Agent Development Kit - Google, accessed on April 10, 2025, https://google.github.io/adk-docs/agents/
8. accessed on January 1, 1970, https://google.github.io/adk-docs/tools/built-in-code-execution/
9. Tools - Agent Development Kit - Google, accessed on April 10, 2025,

https://google.github.io/adk-docs/tools/
10. Quickstart: Build an agent with the Agent Development Kit ..., accessed on April 10, 2025, https://cloud.google.com/vertex-ai/generative-ai/docs/agent-development-kit/quickstart
11. Events: - Agent Development Kit - Google, accessed on April 10, 2025, https://google.github.io/adk-docs/events/
12. Aidemy: Building Multi-Agent Systems with LangGraph, EDA, and Generative AI on Google Cloud, accessed on April 10, 2025, https://codelabs.developers.google.com/aidemy-multi-agent/instructions
13. accessed on January 1, 1970, https://google.github.io/adk-docs/runtime/events/
14. accessed on January 1, 1970, https://google.github.io/adk-docs/get-started/sample-agents/
15. llmflows/docs/user_guide/Creating LLM Flows.md at main - GitHub, accessed on April 10, 2025, https://github.com/stoyan-stoyanov/llmflows/blob/main/docs/user_guide/Creating%20LLM%20Flows.md
16. Automate Your LLM Pipeline: Visualizing Pipelines and Testing Prompt Variants - Medium, accessed on April 10, 2025, https://medium.com/usee-ai/automate-your-llm-pipeline-visualizing-pipelines-and-testing-prompt-variants-a3ffb7158fcd
17. accessed on January 1, 1970, https://google.github.io/adk-docs/sessions-memory/memory/
18. Google models | Generative AI, accessed on April 10, 2025, https://cloud.google.com/vertex-ai/generative-ai/docs/learn/models
19. Google Gemini Module - What is Decisions?, accessed on April 10, 2025, https://documentation.decisions.com/docs/ai-googlegemini
20. Gemini API Libraries | Google AI for Developers, accessed on April 10, 2025, https://ai.google.dev/gemini-api/docs/libraries
21. accessed on January 1, 1970, https://google.github.io/adk-docs/models/
22. Announcing the Agent2Agent Protocol (A2A) - Google for Developers Blog, accessed on April 10, 2025, https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/
23. pyproject.toml - google/adk-python - GitHub, accessed on April 10, 2025, https://github.com/google/adk-python/blob/main/pyproject.toml
24. Vertex AI Agent Engine overview - Google Cloud, accessed on April 10, 2025, https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/overview
25. accessed on January 1, 1970, https://google.github.io/adk-docs/runtime/telemetry/
26. Agent Development Kit documentation - Google, accessed on April 10, 2025, https://google.github.io/adk-docs/api-reference/
27. accessed on January 1, 1970, https://google.github.io/adk-docs/planners/
28. Develop an Agent Development Kit agent | Generative AI on Vertex AI - Google Cloud, accessed on April 10, 2025, https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/develop/adk

29. Create and edit a plan with Performance Planner - Google Ads Help, accessed on April 10, 2025, https://support.google.com/google-ads/answer/9229976?hl=en
30. Submodules - Agent Development Kit documentation - Google, accessed on April 10, 2025, https://google.github.io/adk-docs/api-reference/google-adk.html
31. Ad Planner has been discontinued - Google Help, accessed on April 10, 2025, https://support.google.com/adplanner/answer/3310841?hl=en