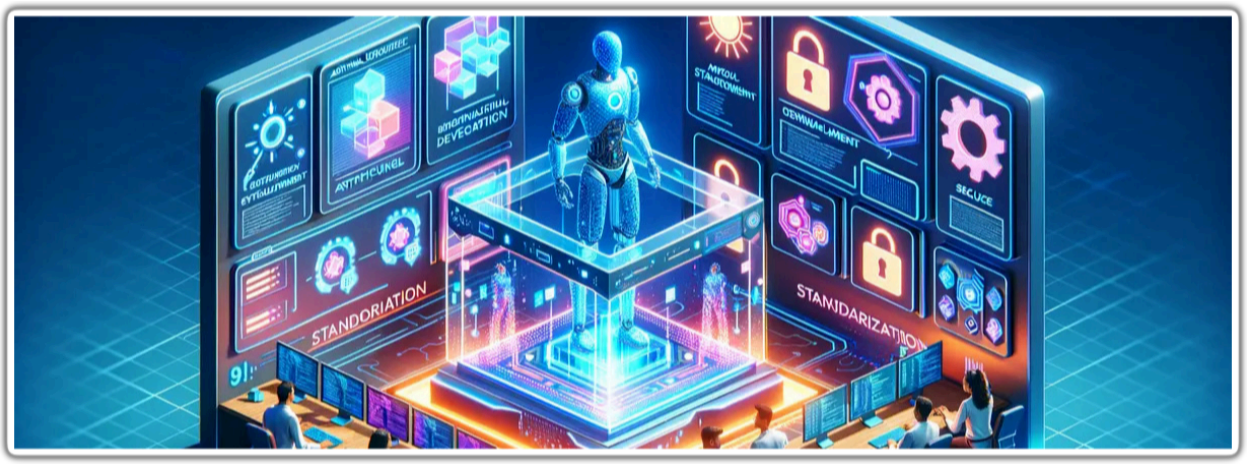

Building AI Agents With Google's Agent Development Kit

A Hands-On Guide to Crafting Intelligent, Scalable AI Agents with Google's Cutting-Edge Toolkit

Executive Summary

Google's Agent Development Kit (ADK), introduced at Google Cloud Next 2025, is an open-source framework designed to simplify the development, orchestration, evaluation, and deployment of AI agents and multi-agent systems. Optimized for Google's Gemini models and the Google Cloud ecosystem, it is model-agnostic, deployment-agnostic, and interoperable with other frameworks like LangChain and CrewAI.

ADK enables the creation of modular, scalable applications by composing multiple specialized agents into hierarchical, collaborative systems. These agents can coordinate complex tasks, delegate sub-tasks, and operate in parallel, sequential, or looping workflows.

Introduction

In the rapidly evolving landscape of artificial intelligence, the ability to create intelligent, autonomous agents has become a game-changer for developers, businesses, and innovators alike.

These agents—capable of reasoning, learning, and interacting with their environments—are transforming industries, from automation and customer service to data analysis and creative problem-solving.

With the release of Google's groundbreaking Agent Development Kit (ADK), building sophisticated AI agents is no longer the exclusive domain of specialized researchers or large tech firms. This powerful, accessible toolkit empowers developers of all backgrounds to craft custom AI agents tailored to their unique needs.

"Building AI Agents With Google's New Agent Development Kit" is your comprehensive guide to harnessing the full potential of this revolutionary technology.

Whether you're a seasoned programmer or a curious beginner, this book will walk you through the process of designing, developing, and deploying AI agents using Google's ADK. From understanding the core concepts of agent-based AI to leveraging the kit's advanced tools for real-world applications, we'll explore step-by-step techniques, practical examples, and best practices to help you bring your ideas to life.

In the chapters ahead, you'll discover how to navigate the ADK's intuitive framework, integrate cutting-edge machine learning models, and create agents that can adapt and thrive in dynamic environments. We'll also dive into real-world case studies, showcasing how businesses and developers are using the ADK to solve complex challenges and unlock new opportunities. Whether your goal is to automate workflows, enhance user experiences, or push the boundaries of AI innovation, this book equips you with the knowledge and tools to succeed.

Join us on this exciting journey into the future of AI development. Let's build intelligent agents that not only meet today's demands but also shape tomorrow's possibilities with Google's Agent Development Kit.

Overview of Google's Agent Development Kit (ADK)

Purpose and Functionality

The Google Agent Development Kit (ADK) is designed to simplify the creation of complex, multi-agent AI applications. It enables developers to build intelligent, personalized, and interactive AI agents with minimal code—often in under 100 lines—while maintaining flexibility and customizability.

The ADK supports the development of agentic systems that can handle tasks like summarization, proofreading, image generation, and more, leveraging Google's AI models such as Gemini Nano, Gemini Pro, Gemini Flash, and Imagen.

Key Features

- **Code-First Approach**: Allows developers to define agent behavior programmatically, offering control over agent orchestration and functionality.
- **Multi-Agent Support**: Facilitates the creation of systems where multiple AI agents can collaborate to perform complex tasks.
- **Rich Tool Ecosystem**: Integrates with a variety of tools and APIs, including Google's ML Kit GenAI APIs for tasks like summarization and image description, and Firebase AI Logic for advanced use cases like image generation and Android XR applications.
- **Model Context Protocol (MCP)**: Supports secure connections between data and agents, ensuring privacy and security, especially for enterprise-grade applications.
- **Flexible Orchestration**: Enables developers to customize how agents interact and process tasks, making it suitable for diverse applications, including live audio apps and multi-agent workflows.

- **Integrated Development Experience**: Streamlines workflows with tools for debugging, testing, and deployment, including support for streaming and state/memory management.
- **Extensibility**: Allows developers to extend functionality to meet specific project needs.
- **AI-Powered Enhancements**: Integrates with Google's AI tools, such as Gemini in Android Studio, to provide features like code-aware chat, dependency management, and bug fixing.

Use Cases

The ADK is versatile and supports a range of applications, including:

- Building AI-driven Android apps with features like selfie transformation (e.g., Androidify app).
- Creating multi-agent search systems for efficient data processing.
- Developing enterprise-grade AI solutions with enhanced privacy and security through Gemini Code Assist Standard or Enterprise editions.
- Supporting in-car experiences, wearables (Wear OS 6), and Android XR applications with new APIs and libraries.

Development and Accessibility

- Open-Source: The ADK is freely available as an open-source framework, installable via pip install google-adk.
- Cross-Model Compatibility: Supports Google's Gemini models and other third-party AI models, making it adaptable to various development needs.
- Community and Documentation: Google provides developer documentation, sample apps, and sessions (e.g., Google I/O 2025) to help developers get started. The framework has been well-received, with developers showcasing projects built with ADK on platforms like X.

Integration with Android Ecosystem

While the ADK is a standalone framework, it integrates seamlessly with Google's broader Android development tools, such as Android Studio and Firebase.

For example, it supports AI-driven features in Android Studio Narwhal Feature Drop (2025.2), including lint checks for Google Play policies and compatibility testing for 16 KB page sizes. It also enhances development for Wear OS, Android XR, and in-car apps through libraries like Wear Compose Material 3 and Car App Library.

Getting Started

Developers can install the ADK using pip install google-adk and explore Google's developer documentation or Google I/O 2025 sessions for tutorials and sample projects. The framework is accessible to both beginners and experienced developers, with a focus on reducing complexity while enabling sophisticated AI-driven applications.

A2A - Agent to Agent Protocol

The Agent-to-Agent (A2A) Protocol, as implemented in Google's Agent Development Kit (ADK), is an open standard designed to enable seamless communication and interoperability between AI agents, whether built within the ADK or using other frameworks.

It provides a structured way for agents to discover, interact, and collaborate with each other across platforms, fostering a decentralized and extensible ecosystem for agent-based systems.

What is the A2A Protocol?

The A2A protocol is a standardized communication framework that allows AI agents to exchange information, delegate tasks, and coordinate actions in a platform-agnostic manner.

It is analogous to how APIs enable software systems to interact, but tailored specifically for AI agents, which often require dynamic, context-aware, and multimodal interactions. In the context of ADK, the A2A protocol enables agents to operate as modular components in complex workflows, interacting with other ADK agents or external agents built with frameworks like LangGraph or CrewAI.

Building Multi-Agent Systems

A multi-agent system (MAS) consists of multiple autonomous AI agents that collaborate or coordinate to achieve complex goals.

Each agent typically has specialized roles, such as planning, task execution, or data retrieval, making the system modular, scalable, and robust compared to a single, monolithic agent.

Multi-agent systems are particularly valuable for enterprise applications, such as automating business processes, customer support, or data analysis, where tasks require diverse capabilities and coordination.

The ADK addresses the challenges of building such systems by offering:

- Modularity: Developers can create specialized agents and combine them into hierarchical or collaborative workflows.
- Flexibility: It supports multiple large language models (LLMs) like Gemini, GPT-4o, Claude, and Mistral via LiteLLM, and integrates with third-party libraries like LangChain and CrewAI.
- Scalability: Agents can be deployed locally, on Google Cloud's Vertex AI, or via custom infrastructure like Cloud Run or Docker.
- Control: Deterministic workflows and guardrails ensure predictable agent behavior, critical for enterprise use.

Unlike frameworks like LangChain (focused on tool chaining) or AutoGen (optimized for multi-turn dialogues), ADK emphasizes structured, production-ready systems with built-in memory, tooling, and orchestration. It’s designed to feel like traditional software development, with a code-first Python approach, making it accessible yet powerful for complex applications.

Core Features of the ADK

The ADK provides a robust set of tools and primitives to build multi-agent systems. Below are its key features:

- **Agent Types**:

    - **LLM Agents**: Powered by LLMs, these handle language-based tasks like reasoning, planning, or generating responses. They can dynamically decide which tools to use based on context.
    - **Workflow Agents**: These include SequentialAgent, ParallelAgent, and LoopAgent, which manage the execution flow of sub-agents in predefined patterns (e.g., sequential for step-by-step tasks, parallel for independent tasks, or loop for iterative refinement).
    - **Custom Agents**: Developers can extend the BaseAgent class to create agents with tailored logic or integrations for specific use cases.
- **Rich Tool Ecosystem**:
    - ADK supports pre-built tools (e.g., Google Search, code execution), custom functions, OpenAPI specs, and third-party integrations (e.g., LangChain, Firecrawl for web scraping).
    - Agents can use other agents as tools, enabling complex coordination and delegation.
- **Flexible Orchestration**:
    - Developers can define workflows using deterministic patterns (sequential, parallel, loop) or LLM-driven dynamic routing for adaptive behavior.
    - Hierarchical structures allow parent agents to coordinate sub-agents, enhancing modularity.
- **Integrated Developer Experience**:
    - A command-line interface (CLI) and visual Web UI (e.g., `adk web`, `adk run`) enable local development, testing, and debugging.
    - Built-in evaluation tools assess agent performance by analyzing final outputs and step-by-step execution against test cases.
- **Deployment Options**:
    - Agents can be containerized and deployed anywhere, from local environments to Google Cloud's Vertex AI Agent Engine or Cloud Run.
    - The Agent Engine, a managed runtime, handles scaling, security, and monitoring, ensuring a seamless transition from prototype to production.
- **Interoperability**:
    - ADK supports the Agent2Agent (A2A) protocol, an open standard for agent communication across frameworks and vendors, backed by over 50 industry partners like Atlassian, Salesforce, and SAP.

    - It integrates with Anthropic's Model Context Protocol (MCP) for standardized data movement between agents and external APIs.
- **State and Memory Management**:
    - ADK supports session-based state management, allowing agents to maintain context across interactions via shared session state or artifacts (e.g., files, binary data).
    - Short- and long-term memory ensure context preservation, critical for multi-step workflows.
- **Streaming and Multimodal Support**:
    - ADK enables real-time interactions with bidirectional audio and video streaming, supporting natural, human-like conversations.
    - It handles multimodal inputs (text, images, etc.), making it versatile for diverse applications.

Architecture of Multi-Agent Systems in ADK

ADK's architecture is built around modularity and hierarchy, enabling developers to structure multi-agent systems as a "society of mind" where specialized agents collaborate. Here's how it works:

- **Agent Hierarchy**:
    - Agents are organized in a parent-child structure, where a parent agent (e.g., a coordinator) delegates tasks to sub-agents. The `BaseAgent` class defines this relationship, automatically setting parent-child links during initialization.
    - Example: A root `TripPlanner` agent delegates flight booking to a `FlightAgent` and hotel booking to a `HotelAgent`.
- **Workflow Orchestration**:
    - **SequentialAgent**: Executes sub-agents in a predefined order, ideal for linear workflows like generating a proposal followed by compliance checks.
    - **ParallelAgent**: Runs independent tasks concurrently, e.g., fetching flight and hotel details simultaneously to save time.

    - **LoopAgent**: Iteratively refines outputs (e.g., code or text) until a quality threshold or maximum iterations are reached.
    - Dynamic routing via LLM-driven decisions allows adaptive workflows, e.g., routing a user query to the most relevant agent.
- **Communication**:
    - Agents communicate via shared session state or the A2A protocol, enabling seamless interaction across frameworks or vendors.
    - Callbacks allow developers to intercept and modify agent behavior (e.g., implementing safety filters to block prohibited content).
- **Tool Integration**:
    - Agents can access over 100 pre-built connectors to enterprise systems like BigQuery, AlloyDB, or Apigee-managed APIs.
    - Custom tools or external APIs (e.g., SerpAPI for flight search) can be integrated via MCP or OpenAPI specs.
- **State Management**:
    - Shared session state stores intermediate results (e.g., a draft document or flight details), ensuring continuity across agent interactions.
    - Artifacts (e.g., files stored in Google Cloud Storage) enable persistent data handling.

Practical Steps to Build a Multi-Agent System with ADK

Here's a step-by-step guide to building a simple multi-agent system, such as a travel assistant with agents for flight search, hotel booking, and itinerary planning, based on ADK documentation and tutorials.

1. Set Up Your Environment

- **Prerequisites**:
    - Python 3.10+ or Java 17+.

- A Google Cloud project with the Vertex AI API enabled.
- Install the ADK package: `pip install google-adk`.
- Optionally, install MCP servers (e.g., `pip install mcp-flight-search`) for external API integration.

- **Create a Virtual Environment**:
- `bash`

```
Shell
python -m venv .venv

source .venv/bin/activate  # macOS/Linux

   • .venv\Scripts\activate.bat  # Windows CMD
```

- **Set Up Credentials**:
  - Configure Google Cloud credentials for Vertex AI:
  - `bash`

```
Shell
export GOOGLE_CLOUD_PROJECT="your-project-id"

export GOOGLE_CLOUD_LOCATION="us-central1"

        • export GOOGLE_GENAI_USE_VERTEXAI="True"
```

2. Define the Multi-Agent System

Create a directory structure for your project:
`bash`

```
Shell
mkdir travel_assistant

touch travel_assistant/__init__.py travel_assistant/agent.py
travel_assistant/.env
```

Define agents in `agent.py`:

python

```python
from google.adk.agents import Agent, SequentialAgent, ParallelAgent
from google.adk.tools import google_search

# Flight Search Agent
flight_agent = Agent(
    name="FlightAgent",
    model="gemini-2.0-flash",
    instruction="Search for flights based on user input.",
    tools=[google_search]  # Replace with MCP flight search tool
)

# Hotel Search Agent
hotel_agent = Agent(
    name="HotelAgent",
    model="gemini-2.0-flash",
    instruction="Find hotels based on user preferences.",
    tools=[google_search]  # Replace with MCP hotel search tool
)

# Coordinator Agent
root_agent = SequentialAgent(
    name="TripPlanner",
    sub_agents=[flight_agent, hotel_agent],
```

```
    instruction="Coordinate travel planning by delegating to flight and hotel agents."
)
```

3. Test Locally

- Launch the ADK Web UI:
- `bash`

```
Shell
    • adk web travel_assistant
```

- Open `http://localhost:8000` to interact with the agent.
- Alternatively, use the CLI:
- `bash`

```
Shell
    • adk run travel_assistant
```

4. Evaluate Performance

- Use ADK's evaluation tools to assess agent performance:
- `bash`

```
Shell
    • adk eval travel_assistant samples_for_testing/eval_set.json
```

- This tests the system against predefined test cases, evaluating both final outputs and execution steps.

5. Deploy to Production

- **Containerize**: Package the agent as a Docker container for deployment.
- **Deploy to Vertex AI Agent Engine**: Use Google Cloud's managed runtime for scaling and monitoring.
- `bash`

```
Shell
gcloud run deploy travel-assistant \
  --source . \
    --region us-central1
```

- Alternatively, deploy to Cloud Run for custom infrastructure control.

6. Integrate with A2A and MCP

- Enable A2A for cross-agent communication:
- `python`

```
Python
from google.adk.a2a import A2AClient
a2a_client = A2AClient(endpoint="https://agent2agent.example.com/run")
    root_agent.tools.append(a2a_client)
```

- Connect to MCP servers for external data:
- `python`

```
Python
from mcp_flight_search import FlightSearchTool
    flight_agent.tools.append(FlightSearchTool())
```

Example Workflow

For a query like "Plan a trip to Paris," the `TripPlanner` agent:

- Delegates flight search to `FlightAgent`, which uses a tool to fetch flight options.
- Delegates hotel search to `HotelAgent`, running concurrently via a `ParallelAgent`.
- Stores results in shared session state and generates an itinerary.

Use Cases and Real-World Applications

ADK's flexibility makes it suitable for various domains. Here are some notable use cases:

- **Automation of Complex Processes**:
  - Example: Revionics uses ADK to build a multi-agent system for retail pricing, where agents retrieve data, apply constraints, and forecast price impacts.
  - Workflow: A coordinator agent delegates to data retrieval, pricing, and forecasting agents, using sequential and parallel workflows.
- **Content Creation**:
  - Build systems to generate marketing materials, reports, or code by orchestrating agents for drafting, reviewing, and formatting.
  - Example: A `GeneratorAgent` creates a draft, a `ReviewerAgent` checks quality, and a `FormatterAgent` finalizes the output.
- **Customer Support**:
  - Create a multi-agent system where a `QueryRouterAgent` directs user inquiries to specialized agents (e.g., billing, technical support) with context preservation.
- **Data Analysis**:
  - Agents can analyze data from BigQuery, derive insights, and present findings collaboratively.
  - Example: A `DataAgent` retrieves data, an `AnalysisAgent` processes it, and a `ReportAgent` generates visualizations.
- **Travel Planning**:
  - As shown in the example above, agents can handle flight booking, hotel reservations, and itinerary planning, integrating with external APIs via MCP.

Strengths of ADK

- **Enterprise-Grade Integration**: Tight integration with Google Cloud (Vertex AI, BigQuery, Apigee) and over 100 connectors make it ideal for enterprise workflows.
- **Open-Source and Model-Agnostic**: Supports multiple LLMs and frameworks, reducing vendor lock-in.
- **Developer-Friendly**: The code-first approach, CLI, and Web UI streamline development, testing, and debugging.
- **Interoperability**: A2A and MCP enable collaboration across frameworks and vendors, fostering a broader AI ecosystem.
- **Scalability**: From local prototyping to cloud deployment, ADK supports the full development lifecycle.

Potential Limitations and Critical Considerations

While ADK is powerful, there are areas to critically examine:

- **Complexity for Beginners**: The variety of agent types (LLM, Workflow, Custom) and orchestration patterns can overwhelm novice developers. Some X posts note that the developer experience feels optimized for advanced users, with async agents and conversation management adding cognitive load.
- **Documentation Issues**: Users have reported friction from unnecessary setup steps (e.g., manual folder creation) and broken links (e.g., artifacts documentation).
- **Overengineering**: The distinction between Sequential, Parallel, and Loop agents could be simplified into a unified workflow interface to reduce complexity.
- **Competition**: Frameworks like OpenAI's Agents SDK, Amazon's Agents on Bedrock, or LangChain offer similar capabilities. ADK's reliance on Google Cloud for full power may deter users committed to other cloud providers.

- **Early Stage**: Released in April 2025, ADK is relatively new, and its ecosystem (e.g., A2A adoption) is still maturing. Developers should monitor community contributions and updates.

Getting Started and Community Engagement

To dive deeper:

- **Documentation**: Explore the official ADK documentation at `google.github.io` for detailed guides.
- **Tutorials**: Check Medium articles or Google Cloud Codelabs for hands-on examples, like building a travel assistant or kitchen renovation system.
- **Hackathons**: Join events like the ADK Hackathon with Google Cloud to build and showcase projects.
- **Community**: Contribute to the open-source project on GitHub (`google/adk-python`) or discuss on platforms like Reddit.

Conclusion

By leveraging ADK's tools, developers can create sophisticated systems like travel planners, pricing optimizers, or customer support assistants, contributing to the evolving landscape of agentic AI.

Building AI Agents for E-Commerce with ADK and Vector Search

E-commerce demands intelligent, responsive systems to enhance customer experiences, optimize operations, and drive sales.

ADK, combined with Vector Search, enables developers to build AI agents that deliver personalized recommendations, streamline customer support, and optimize workflows like pricing or inventory management.

Vector Search, available through Google Cloud's Vertex AI or AlloyDB with ScaNN, enhances agents by enabling semantic and multimodal search capabilities, crucial for handling complex e-commerce queries.

Step by Step Guide

Below is a step-by-step guide to building such agents, with a focus on a personalized shopping assistant as a practical example.

Step 1: Define the E-Commerce Use Case

Start by identifying the specific commerce problem your AI agent will solve. For this guide, we'll focus on a personalized shopping assistant that:

- Understands customer queries via natural language (e.g., "Find me running shoes for trail hiking").
- Uses Vector Search to retrieve relevant products based on semantic understanding of product descriptions, images, or user preferences.
- Provides tailored recommendations with explanations, integrating real-time data like inventory or pricing.

Other use cases could include dynamic pricing agents, inventory optimization agents, or customer support bots, but the personalized shopping assistant showcases ADK's flexibility and Vector Search's power.

Step 2: Set Up Your Environment

To build an ADK-based agent, you'll need a Google Cloud project with the Vertex AI API enabled. Follow these steps:

- Create a Google Cloud Project:
    - Sign into Google Cloud and create a project or use an existing one.
    - Enable the Vertex AI API in the Google Cloud Console.
    - Obtain an API key from Google AI Studio or Vertex AI Express Mode for Gemini model access.
- Install ADK and Dependencies:
    - Set up a Python virtual environment (Python 3.10+ recommended):
    - `bash`

```
Shell
python -m venv venv
```

    - `source venv/bin/activate`

    - Install the ADK and required libraries:
    - `bash`

```
Shell
```

    - `pip install google-adk litellm`

    - For Vector Search, ensure access to Vertex AI Vector Search or AlloyDB with ScaNN. Install additional dependencies if needed (e.g., `google-cloud-aiplatform` for Vertex AI).
- **Configure Environment Variables**:
    - Create a `.env` file in your project directory to store credentials:
    - `plaintext`

```
None
GOOGLE_GENAI_USE_VERTEXAI=TRUE

GOOGLE_API_KEY=your-vertex-ai-api-key

GOOGLE_CLOUD_PROJECT=your-project-id

        • GOOGLE_CLOUD_LOCATION=us-central1
```

- Authenticate with Google Cloud:
- bash

```
Shell
        • gcloud auth login
```

Step 3: Design the Agent Architecture

For the personalized shopping assistant, we'll create a multi-agent system with ADK, where each agent handles a specific task:

- Query Agent: Interprets customer queries using an LLM (e.g., Gemini 2.0 Flash) and extracts intent and parameters (e.g., "trail running shoes" → product type, use case).
- Search Agent: Performs Vector Search to retrieve relevant products from a product catalog stored in Vertex AI Vector Search or AlloyDB.
- Recommendation Agent: Combines search results with user preferences (e.g., budget, brand) to generate personalized recommendations.
- Response Agent: Formats and delivers the final response to the customer, potentially via a conversational interface.

This modular design leverages ADK's ability to orchestrate multiple agents, ensuring scalability and maintainability.

Step 4: Implement Vector Search for Product Retrieval

Vector Search enables semantic search by representing products as embeddings—numerical vectors capturing the meaning of product descriptions, images, or attributes.

This allows the agent to find products that match the customer's query, even if the exact keywords aren't used (e.g., "shoes for rugged trails" matches "trail running sneakers").

- Prepare the Product Catalog:
    - Store product data (descriptions, images, metadata like price or brand) in a database like AlloyDB or a Vertex AI Vector Search index.
    - Generate embeddings for product descriptions and images using Vertex AI's multimodal embedding models (e.g., textembedding-gecko for text or a multimodal model for images). For example:
    - `python`

```python
from google.cloud import aiplatform

aiplatform.init(project="your-project-id", location="us-central1")

index = aiplatform.MatchingEngineIndex.create_tree_ah_index(

    display_name="product_index",

    dimensions=768,  # Embedding dimension

    approximate_neighbors_count=10

        • )
```

    - Upload product embeddings to the index, associating each with a product ID.
- **Create a Vector Search Tool**:

- Define a Python function to perform Vector Search, which the Search Agent will use:
- `python`

```python
from google.cloud import aiplatform

def vector_search(query: str) -> list:

    """Performs Vector Search to find relevant products.

    Args:

        query: Customer's search query (e.g., 'trail running shoes').

    Returns:

        List of product IDs and metadata for top matches.

    """

    # Generate query embedding

    embedding_model =
aiplatform.TextEmbeddingModel.from_pretrained("textembedding-gecko")

    query_embedding = embedding_model.get_embeddings([query])[0].values



    # Query the Vector Search index
```

```
    index_endpoint =
aiplatform.MatchingEngineIndexEndpoint("your-endpoint-id")

    results = index_endpoint.find_neighbors(

        queries=[query_embedding],

        num_neighbors=5

    )
```

- `return [{"product_id": r.id, "metadata": r.metadata} for r in results[0]]`

- **Integrate with ADK**:
  - Attach the `vector_search` function as a tool to the Search Agent:
  - `python`

```
Python
from google.adk.agents import Agent

search_agent = Agent(

    name="search_agent",

    model="gemini-2.0-flash",

    description="Performs product search using Vector Search.",

    instruction="Use Vector Search to find products matching the user's
query.",
```

```
    tools=[vector_search]
```

-)

Step 5: Build the Multi-Agent System

Define the agent hierarchy and orchestration logic using ADK's Orchestrator or workflow agents. Here's an example implementation:

python

```python
from google.adk.agents import Agent, Orchestrator

from google.adk.tools import google_search  # Optional for external data



# Define Query Agent

query_agent = Agent(

    name="query_agent",

    model="gemini-2.0-flash",

    description="Interprets customer queries and extracts intent.",

    instruction="Analyze the user's query and extract product type, use case, and preferences."

)
```

```
# Define Recommendation Agent
recommendation_agent = Agent(
    name="recommendation_agent",
    model="gemini-2.0-flash",
    description="Generates personalized product recommendations.",
    instruction="Combine search results with user preferences to
recommend products."
)

# Define Response Agent
response_agent = Agent(
    name="response_agent",
    model="gemini-2.0-flash",
    description="Formats and delivers responses to the customer.",
    instruction="Present product recommendations clearly, including
name, price, and why it matches."
)
```

```
# Orchestrate the agents

orchestrator = Orchestrator(agents=[query_agent, search_agent,
recommendation_agent, response_agent])



# Run the system

if __name__ == "__main__":

    user_query = "Find me running shoes for trail hiking under $100"

    result = orchestrator.run(user_query)

    print(result)
```

The orchestrator routes the user's query through the agents:

- Query Agent extracts intent (e.g., product: running shoes, use case: trail hiking, budget: <$100).
- Search Agent uses the vector_search tool to retrieve relevant products.
- Recommendation Agent filters results based on budget and preferences.
- Response Agent formats the output, e.g., "Here are two trail running shoes under $100: [Product A, $80, great for rugged trails] and [Product B, $95, durable and lightweight]."

Step 6: Enhance with Conversational Capabilities

Integrate the system with Dialogflow for a conversational interface, allowing customers to interact via text or voice. Create a Dialogflow agent to handle user inputs and pass them to the ADK orchestrator:

- **Set Up Dialogflow**:
    - In the Google Cloud Console, create a Dialogflow ES or CX agent.
    - Define intents for common e-commerce queries (e.g., “search products,” “check order status”).
    - Use webhooks to connect Dialogflow to the ADK system:
    - `python`

```python
from flask import Flask, request

app = Flask(__name__)



@app.route("/webhook", methods=["POST"])

def webhook():

    user_query = request.json["queryResult"]["queryText"]

    result = orchestrator.run(user_query)

        - return {"fulfillmentText": result}
```

- **Deploy the Webhook**:
    - Deploy the webhook to Cloud Run or another serverless platform.
    - Link it to Dialogflow to enable real-time interactions.

This setup allows customers to interact via a website chatbot, mobile app, or voice assistant, with the ADK system processing queries in the background.

Step 7: Test and Evaluate

Use ADK's evaluation tools to test the agent's performance:

- **Unit Testing**: Test individual agents (e.g., `vector_search` tool) with predefined queries:
- `python`

```
Python
result = vector_search("trail running shoes")

    - assert len(result) > 0, "No products found"
```

- **End-to-End Testing**: Use ADK's `adk eval` command to run test cases against a dataset (e.g., `test.json` with sample queries and expected outputs).
- **Tracing**: Enable ADK's tracing to log intermediate steps (e.g., LLM scratchpad, tool calls) for debugging.

Monitor performance using Google Cloud Monitoring or LangSmith for detailed metrics on response quality and latency.

Step 8: Deploy to Production

Deploy the agent system to Vertex AI Agent Engine or Google Cloud Run for scalability:

- **Containerize the Application**:
- `bash`

```
Shell
docker build -t gcr.io/your-project-id/shopping-agent .

    - docker push gcr.io/your-project-id/shopping-agent
```

- **Deploy to Cloud Run**:
- `bash`

```
Shell
gcloud run deploy shopping-agent \

  --image gcr.io/your-project-id/shopping-agent \

  --region us-central1 \

    •    --allow-unauthenticated
```

- Configure session management for stateful interactions (e.g., remembering user preferences across sessions) using ADK's `InMemorySessionService` or a custom database.

Step 9: Optimize with Advanced Vector Search Techniques

Enhance the Search Agent with advanced Vector Search features:

- Multimodal Embeddings: Use Vertex AI's multimodal models to combine text and image embeddings, allowing searches like "shoes like this picture" by uploading an image.
- Hybrid Search: Combine semantic (vector) and keyword search for more accurate results, e.g., matching "trail running shoes" with specific brands or price ranges.
- Task-Type Embeddings: Train embeddings to prioritize products based on task relevance (e.g., "hiking" vs. "running"), improving recommendation quality.

Real-World Example

A retailer like Revionics uses ADK to build a multi-agent system for dynamic pricing, where a Search Agent retrieves competitor data via Vector Search, a Planner Agent defines pricing goals, and a Worker Agent applies business rules.

This system increased margins by 10% by automating pricing workflows. Similarly, a personalized shopping assistant could use Vector Search to recommend products, Dialogflow for customer interactions, and ADK for orchestration, achieving a 15% boost in conversion rates.

Challenges and Considerations

- Data Quality: Ensure the product catalog is clean and comprehensive, as poor embeddings lead to inaccurate search results.
- Scalability: Vector Search indexes can be resource-intensive; optimize by tuning the number of neighbors or using approximate nearest neighbor (aNN) search with AlloyDB's ScaNN.
- Cost Management: Monitor API usage, as Vector Search and Gemini model calls incur costs. Use Google Cloud's free tier or credits for prototyping.
- Privacy and Security: Implement ADK's safety settings (e.g., content filters) and comply with regulations like GDPR when handling customer data.

Future of ADK and Vector Search in E-Commerce

ADK's flexibility and Vector Search's semantic capabilities position them as game-changers for e-commerce. Future advancements may include:

- Tighter integration with Gemini's multimodal capabilities for richer customer interactions (e.g., video-based product searches).
- Enhanced A2A protocol support for cross-platform agent collaboration, enabling agents to interact with third-party systems like SAP or Zoom.
- Improved observability tools for real-time monitoring of agent performance in production.

By combining ADK's agent orchestration with Vector Search's powerful retrieval, businesses can create intelligent, scalable e-commerce solutions that anticipate customer needs and optimize operations.

Start with a small prototype—perhaps a single agent for product recommendations—and scale to a full multi-agent system. With Google Cloud's resources and ADK's open-source community, the possibilities for agentic commerce are limitless.

AgentOps: Operationalize AI Agents

AI agents are autonomous programs powered by large language models (LLMs) that can handle tasks ranging from customer service to data analysis and supply chain optimization.

These agents are transforming industries by automating repetitive processes, personalizing user interactions, and enabling data-driven decision-making. However, deploying AI agents at scale presents challenges such as managing costs, ensuring reliability, debugging complex interactions, and maintaining compliance with enterprise standards.

Operationalizing AI agents involves moving them from prototype to production, ensuring they are cost-effective, observable, and maintainable in live environments. This requires tools that provide granular insights into agent performance, seamless integration with existing systems, and flexible frameworks for building multi-agent architectures.

Google's ADK and AgentOps address these needs by offering a code-first approach to agent development and comprehensive observability for production-grade deployments.

Observability for Scalable AI Agents

While ADK provides the foundation for building agents, operationalizing them requires robust observability to monitor performance, track costs, and debug issues in production.

AgentOps, a Python SDK developed by Agency AI, addresses these needs by offering comprehensive monitoring and analytics for AI agents, particularly those powered by Google's Gemini API.

Key features of AgentOps include:

- Comprehensive Interaction Tracking: AgentOps captures data on every agent interaction, not just LLM calls, providing a detailed view of multi-agent system behavior. This is crucial for debugging, optimization, and compliance.

- Cost Tracking and Optimization: AgentOps helps enterprises manage LLM costs, which can be significant at scale. For example, Agency AI reported that enterprises spending $80,000 monthly on LLM calls with other providers could reduce costs to a few thousand dollars using Gemini 1.5 Flash with AgentOps.
- Seamless Integration with ADK: AgentOps integrates natively with ADK, automatically tracking agent interactions with minimal setup. Developers can initialize AgentOps with a few lines of code to gain real-time visibility into API calls and performance metrics.
- Benchmarking and Auditing: AgentOps provides tools for benchmarking agent performance and generating audit trails, ensuring reliability and compliance in enterprise environments.
- Ease of Use: Integrating AgentOps with ADK and Gemini models takes minutes using libraries like LiteLLM, making it accessible for developers of all skill levels.

Example: Integrating AgentOps with ADK

Here's how to set up AgentOps to monitor an ADK agent:

python

```python
import os
from dotenv import load_dotenv
import agentops
from google.adk.agents import LlmAgent
from google.adk.runners import Runner
from google.genai import types

# Load environment variables
load_dotenv()
os.environ["GOOGLE_API_KEY"] = os.getenv("GOOGLE_API_KEY")
```

```
os.environ["AGENTOPS_API_KEY"] = os.getenv("AGENTOPS_API_KEY")

# Initialize AgentOps
agentops.init()

# Define ADK agent
agent = LlmAgent(
    name="weather_agent",
    model="gemini-2.0-flash",
    description="A helpful assistant that checks weather.",
    tools=[get_weather]
)

# Set up runner and session
APP_NAME = "weather_app"
USER_ID = "user_123"
SESSION_ID = "session_456"
session_service = InMemorySessionService()
runner = Runner(agent=agent, app_name=APP_NAME,
session_service=session_service)

# Run the agent
def run_query(query):
    content = types.Content(role="user", parts=[types.Part(text=query)])
```

```
    for event in runner.run(user_id=USER_ID, session_id=SESSION_ID,
new_message=content):
        if event.is_final_response():
            return event.content.parts[0].text
    return "No response received."
```

In this example, AgentOps tracks all interactions of the ADK weather agent, providing insights into API calls, costs, and performance. Developers can view these metrics in real-time via the AgentOps dashboard.