A Comprehensive Guide to Implementing Model Context Protocol Servers with the Agent Development Kit

I. Introduction

The landscape of artificial intelligence (AI) is rapidly evolving, with AI agents increasingly tasked with complex operations that require interaction with external systems, tools, and data sources. Bridging the gap between AI models and these external capabilities in a standardized, scalable, and efficient manner is a critical challenge. This report provides an in-depth exploration of the Model Context Protocol (MCP) and its integration with Google's Agent Development Kit (ADK), offering a comprehensive guide for creating and connecting MCP servers.

A. Defining the Model Context Protocol (MCP)

The Model Context Protocol (MCP) is an open standard designed to facilitate seamless integration between Large Language Model (LLM) applications and external data sources and tools.[1] Its primary purpose is to provide a standardized way for AI agents to discover, connect to, and utilize these external capabilities (Video Guide: 3:15). MCP addresses the significant problem of redundancy and scalability issues inherent in traditional tool-calling methods, where developers often need to write custom code for every API an agent interacts with.[2] By establishing a common framework, MCP enables AI agents to dynamically discover and use APIs without hardcoded logic, acting as a server that interfaces between an AI agent and various tools.[2]

B. Introducing the Agent Development Kit (ADK)

Google's Agent Development Kit (ADK) is an open-source, code-first Python toolkit designed for building, evaluating, and deploying sophisticated AI agents with flexibility and control.[3] While optimized for Google's Gemini models and ecosystem, ADK is model-agnostic and built for compatibility with other frameworks.[3] In the context of MCP, ADK provides robust mechanisms for AI agents to act as MCP clients, simplifying the process of connecting to and utilizing tools exposed by MCP servers (Video Guide: Overview). The MCPToolSet class within ADK is a primary component for this integration.[4]

C. Purpose and Scope of the Report

This report aims to deliver a comprehensive, expert-level guide on creating or connecting to Model Context Protocol (MCP) servers, with a particular focus on

leveraging Google's Agent Development Kit (ADK). Drawing extensively from a detailed video tutorial and supplemented by broad research into MCP and ADK documentation and community resources, this document will cover:

- Fundamental concepts of MCP and its advantages over traditional tool-calling.
- Step-by-step instructions for connecting ADK agents to remote MCP servers.
- Detailed guidance on building custom local MCP servers and integrating them with ADK agents.
- An overview of key technologies, resources, development tools, and community support channels.
- Advanced considerations such as security, scalability, error handling, and best practices for tool design.

The objective is to equip developers with the necessary knowledge and practical steps to effectively implement MCP solutions within their AI agent architectures.

II. Understanding the Core Concepts: MCP and Traditional Tool Calling

To appreciate the advancements offered by MCP, it is essential to first understand the conventional methods of tool integration for AI agents and their inherent limitations.

A. Traditional Tool Calling Explained

In traditional AI agent development, connecting an agent to an external tool, such as the Notion API, typically involves writing specific code to interact with each function or endpoint the tool's API offers [Video Guide: 4:11-4:16]. For instance, if an agent needs to query a database, insert a block, or create a page in Notion, a developer would manually implement the logic for each of these API calls.

This approach, while direct, suffers from significant redundancy. If multiple AI agents or different projects require access to the same Notion tools, developers often find themselves duplicating substantial amounts of integration code [Video Guide: 4:20, 7:00]. Sharing tools across projects frequently devolves into copying and pasting Python files or modules, an inefficient and error-prone practice that hinders scalability and maintainability [Video Guide: 7:00-7:27]. Each new agent or project might necessitate rewriting or adapting these custom wrappers, leading to a fragmented and difficult-to-manage tool ecosystem.

B. The Model Context Protocol (MCP) Approach

MCP introduces a more structured and standardized paradigm for tool integration, designed to overcome the inefficiencies of traditional methods.

1. Architecture and Operational Flow
   MCP operates on a client-server architecture. The AI agent (or the application hosting it) acts as the MCP client, while the MCP server is an intermediary that wraps around one or more tools, exposing them in a standardized way.2 This server can connect to various external tools and APIs, such as the Notion API, which might itself be wrapped in a dedicated "Notion MCP Server" [Video Guide: 8:00].
   The operational flow typically involves the following steps [2]:
   - **Client Constructs the Request**: The AI agent assembles a payload specifying the method (operation to perform), parameters for the method, and any necessary context (e.g., authentication tokens, trace IDs).
   - **Client Sends the Request**: The payload is sent over a transport layer (like stdio for local servers, or HTTP/WebSocket for remote servers) to the MCP server.
   - **Server Validates and Logs**: The MCP server authenticates the request, validates it against the tool's schema, and logs the request for auditing or tracing.
   - **Server Executes the Request**: The server translates the MCP method into backend operations (e.g., an API call to Notion, a SQL query). It may enforce rate limits or other business logic.
   - **Data Source Responds**: The underlying tool or data source returns raw data to the MCP server.
   - **Server Responds to Client**: The MCP server formats the response according to the MCP specification and sends it back to the AI agent.

   MCP Servers expose capabilities through three primary components [5]:
   - **Tools**: Actions the AI can invoke to perform operations or trigger external workflows (e.g., create_notion_page).
   - **Resources**: Read-only or low-side-effect data endpoints that provide external information (e.g., get_user_profile).
   - **Prompts**: Predefined conversation templates designed to guide or structure interactions for specialized tasks.
2. Standardization and Benefits
   The core strength of MCP lies in its standardization. Instead of bespoke integrations for each tool, MCP provides an open, uniform framework.2 Developers describe their tools using a standardized schema, allowing AI agents to dynamically discover, understand, and use these tools without hardcoded

logic.2

This leads to several key benefits:

- **Scalability and Reusability**: Tools wrapped by an MCP server become highly scalable and easily reusable across different agents and projects. Any agent capable of speaking MCP can connect to an MCP server and access its tools without requiring new integration code.[2]
- **Simplified Development**: Developers can focus on building the core functionality of their AI models and tools, rather than on the boilerplate of API integration. MCP reduces the complexity of creating custom integrations.[2]
- **Interoperability and Consistency**: MCP ensures consistency in how agents interact with external systems, making it easier to scale AI-powered applications. An AI model built with MCP can interact with any compliant MCP server or API.[2]
- **Unified Interface**: MCP provides a single interface for diverse APIs, tools, and data sources, reducing the complexity of managing multiple integrations.[2]
- **Dynamic Tool Discovery**: Models can automatically detect and interact with new tools or services exposed by an MCP server without manual reconfiguration.[5]

C. When to Use MCP (and When Not To)

The video guide provides pragmatic advice on when to adopt MCP [Video Guide: 12:58]:

- **Use MCP if**:
  - You are accessing common APIs (e.g., Notion, Slack, Google Maps), as pre-built MCP servers for these are increasingly available (like the official Notion MCP server [Video Guide: 8:16, 8:30]).
  - You need to share tools across multiple projects or agents, as MCP's reusability offers significant advantages.
- **Don't use MCP (or it might be overkill) if**:
  - You are rapidly experimenting with very custom, one-off APIs locally. In such scenarios, the overhead of setting up an MCP server for a tool that won't be reused might outweigh the benefits, and traditional, direct tool creation could be simpler and faster for that specific, isolated use case.

However, even for local custom tools, if there's a prospect of future reuse or sharing, investing in an MCP wrapper can be beneficial in the long run. The initial complexity of setting up custom tools with MCP, especially involving local server setup and potentially more challenging debugging, is a factor to consider [Video Guide: 8:12,

12:11].

III. Phase 1: Connecting an ADK Agent to a Remote MCP Server

The first practical step in leveraging MCP with ADK often involves connecting an AI agent to an existing MCP server that is hosted remotely or can be run as a distinct process. This allows the agent to immediately access the tools provided by that server.

A. Overview

The primary goal of this phase is to enable an ADK-based AI agent to communicate with an established MCP server, such as the official Notion MCP server or one of the many reference servers available [Video Guide: 13:53]. This connection allows the agent to list and call tools hosted by that server.

B. Finding Remote MCP Servers

Several resources exist for discovering available MCP servers:

1. **ModelContextProtocol/servers GitHub Repository**: This repository is a central hub maintained by the Model Context Protocol project. It lists numerous reference implementations and community-built servers, showcasing how MCP can grant LLMs access to various tools and data sources.[1] Examples include servers for file system operations, Git, Google Drive, Google Maps, Slack, and more.[8]
2. **Smithery.ai**: This platform facilitates the hosting, deployment, and discovery of MCP servers.[9] It allows users to deploy their own MCP servers or connect to existing ones listed on the platform.[9]
3. **Other Sources**:
    - Specific service providers, like Notion, offer their own official MCP servers and provide documentation on how to connect to them.[11]
    - Anthropic maintains a list of third-party remote MCP servers that can be connected to via their API, including servers for Asana and Atlassian.[12]
    - Platforms like Azure API Center can be used to register and discover MCP servers within an organization's API inventory, treating MCP server endpoints as a specific API type.[13]
    - VS Code's Copilot Chat can also connect to MCP servers, and its documentation points to the official MCP server repository as a starting point.[14]

C. Connecting to a Remote Server: The Notion MCP Server Example (as per

Video Guide)

The video guide demonstrates connecting to the Notion MCP server, which is hosted by Make Notion on GitHub and can be run using npx [Video Guide: 19:21, 19:52].

1. **Prerequisites**:
   - **Notion API Key**: A Notion integration token (secret) is required. This is obtained by creating an "internal integration" in your Notion workspace and granting it necessary capabilities (e.g., read, update, insert content) [Video Guide: 19:17, 20:05-20:24].
   - **Sharing with Integration**: The specific Notion pages or databases the agent needs to access must be explicitly shared with this created integration within Notion's interface [Video Guide: 20:25].
2. ADK Agent Code Setup (agent.py):
   The video illustrates using StdioServerParameters to launch the Notion MCP server via an npx command. This command downloads and runs the server package, which then likely handles the actual remote communication with Notion's backend APIs.
   - **Imports**:
     Python
     ```
     from google.adk.tools.mcp_tool.mcp_toolset import MCPToolSet, StdioServerParameters
     ```
     [Video Guide: 10:25]
   - **Environment Variables**: The Notion API key should be securely managed, typically as an environment variable. The video shows setting this up to be passed to the npx command [Video Guide: 19:17-19:39].
     Python
     ```
     # Example:
     NOTION_API_KEY = os.environ.get("NOTION_API_KEY")
     # Headers might also be needed, e.g., for Notion API version
     NOTION_HEADERS = {"Notion-Version": "2022-06-28"}
     ```
   - **MCPToolSet Configuration**: Within the tools list of your ADK LlmAgent, an MCPToolSet instance is added:
     Python
     ```
     tools=, # Command to run Notion MCP server
             env={
                 'NOTION_API_KEY': NOTION_API_KEY,
                 'NOTION_MCP_SERVER_HEADERS': json.dumps(NOTION_HEADERS) # Pass headers as JSON string
             }
     ```

```
    )
  )
]
```

[Video Guide: 10:25, 11:28, 11:31, 19:52, 20:01] Here, StdioServerParameters is used because the ADK agent is starting a local process (npx) that acts as the MCP server (or a client/proxy to it). The env dictionary passes necessary credentials and configurations to this process.

D. Alternative: Connecting to True Remote Servers using SseServerParams

For MCP servers that are already running remotely and expose an HTTP endpoint for Server-Sent Events (SSE) communication, ADK provides SseServerParams. This is distinct from launching a local process with StdioServerParameters.

1. Introduction to SseServerParams:
   SseServerParams is used when the ADK agent needs to connect to an MCP server accessible via a URL, typically using SSE for receiving streaming responses and server-initiated messages.4 Some MCP server implementations might use two distinct endpoints: one for POST requests from the client and another GET endpoint for the SSE connection.15 More recent MCP specifications also include "Streamable HTTP," which can use a single HTTP endpoint.9
2. Configuration:
   When using MCPToolSet.from_server or directly instantiating MCPToolSet with SseServerParams, key fields include 4:
   - url: The URL of the remote MCP server's SSE endpoint.
   - headers: A dictionary for any required HTTP headers, such as Authorization for API keys or bearer tokens.
   - message_url (optional): If the server uses a different URL for POSTing messages than for the SSE stream, this field specifies it.
3. Example Scenario (Conceptual Wikipedia Server):
   Based on an example of an ADK agent connecting to an external Wikipedia MCP server using SSE 15, the setup in agent.py would conceptually look like this:

```
Python
# Conceptual example based on [15]
from google.adk.tools.mcp_tool.mcp_toolset import MCPToolSet, SseServerParams
# Assuming API_KEY is required by the remote server and passed in headers
# WIKIPEDIA_MCP_API_KEY = os.environ.get("WIKIPEDIA_MCP_API_KEY")

# tools, exit_stack = await MCPToolSet.from_server(
#     connection_params=SseServerParams(
#         url="http://remote-wikipedia-mcp-server.com/sse", # SSE endpoint
```

```
#         # message_url="http://remote-wikipedia-mcp-server.com/message", # If different for POST
#         # headers={
#         #     "Authorization": f"Bearer {WIKIPEDIA_MCP_API_KEY}"
#         # }
#     )
# )
# root_agent = LlmAgent(..., tools=tools)
```

This demonstrates how an agent would establish a direct network connection to a pre-existing remote MCP server. The MCPToolSet.from_server method is often used for asynchronous setup.[15]

E. Running the Agent and Interaction

Once the agent.py file is configured:

1. **Execute the ADK agent**: Typically using the command adk web in the terminal from the project's root directory [Video Guide: 22:12]. This starts the ADK development web UI.
2. **Server Launch (for StdioServerParameters)**: The MCPToolSet will automatically launch the MCP server specified in command (e.g., the npx process for Notion) as a subprocess.
3. **Interaction**: Through the ADK Web UI, users can interact with the agent.
    - Asking "what tools do you have access to?" will prompt the agent to query the MCP server (e.g., Notion MCP server) and list the available tools [Video Guide: 22:50].
    - Subsequent natural language requests, like "find my recent databases" or "add a comment to page X," will be processed by the ADK. The ADK, in conjunction with the LLM, maps these requests to the appropriate tool calls and parameters, which are then sent to the MCP server for execution [Video Guide: 23:10-26:08, 24:49]. The results are returned to the agent and then to the user.

IV. Phase 2: Creating and Connecting to a Local MCP Server

While connecting to existing remote MCP servers provides access to established tools, many use cases require creating custom tools, often interacting with local data sources or private APIs. This phase details building a local MCP server using Python and connecting an ADK agent to it.

A. Overview

This section focuses on constructing an MCP server that runs on the local machine

and exposes custom functionalities. The video guide uses a local SQLite database as an example to demonstrate how to wrap custom Python functions into MCP tools [Video Guide: 27:14]. The ADK agent will then connect to this locally running server.

B. Setting up the Local Tool Environment (Example: SQLite Database)

1. **Database Creation**: A local SQLite database (e.g., database.db) is established, containing relevant tables (e.g., users, todos in the video example) [Video Guide: 28:59].
2. **Python Utility Functions**: A set of Python functions is created to perform operations on this database. These functions encapsulate the core logic that will eventually be exposed as tools. Examples include list_db_tables, get_table_schema, query_db_table, insert_data, and delete_data [Video Guide: 28:05]. These functions are standard Python code, not yet MCP-specific.

C. Building the Local MCP Server (server.py)

The server.py script is the heart of the local MCP setup. It defines the server, its tools, and how it communicates.

1. Core Imports:
   Essential modules from the mcp Python SDK are imported. The video uses mcp.server.stdio_server for communication and mcp.server.server for the server logic [Video Guide: 28:33, 29:00]. Other common imports include asyncio and mcp.types.16
   Python
   ```
   import asyncio
   from mcp.server import Server # Or from mcp.server.lowlevel import Server
   import mcp.server.stdio
   from mcp import types as mcp_types
   from google.adk.tools.function_tool import FunctionTool
   from google.adk.tools.mcp_tool.conversion_utils import adk_to_mcp_tool_type
   # Your database utility functions (e.g., from db_utils.py)
   # from. import db_utils
   ```

2. Defining Tools as Python Functions:
   The previously created database utility functions (e.g., list_db_tables, query_db_table) serve as the basis for the tools [Video Guide: 29:49].
3. Converting Python Functions to MCP Tool Schema:
   This is a critical step to make standard Python functions understandable to the MCP framework and, by extension, to the LLM via the ADK agent.

- **Wrap with FunctionTool**: Each Python function intended to be an MCP tool is first wrapped in an ADK FunctionTool object. This ADK class helps describe the function, its parameters, and its return type based on Python type hints and docstrings [Video Guide: 32:00-32:25].

```
Python
# Example:
# adk_list_tables_tool = FunctionTool(func=db_utils.list_db_tables)
# adk_query_table_tool = FunctionTool(func=db_utils.query_db_table)
# ADK_DB_TOOLS = {
#     "list_database_tables": adk_list_tables_tool,
#     "query_database_table": adk_query_table_tool,
# }
```

- **Convert to MCP Schema with adk_to_mcp_tool_type**: The ADK provides a utility function, adk_to_mcp_tool_type, which takes an ADK FunctionTool instance and converts its schema into an MCP-compatible tool schema (mcp_types.Tool).[17] This generated schema includes the tool's name, description, and input/output parameter definitions that the MCP server will advertise.
  - **Addressing inputSchema: None for No-Parameter Functions**: A known issue (or behavior) with adk_to_mcp_tool_type is that if a FunctionTool wraps a Python function with no parameters, the resulting mcp_types.Tool object might have its inputSchema attribute set to None.[18] For MCP compliance and robust client interaction, an empty parameter list should ideally be represented by an empty JSON object schema (e.g., {"type": "object", "properties": {}}). The workaround, if this issue is encountered, is to add a dummy, unused parameter to the Python function definition and its type hints. This forces adk_to_mcp_tool_type to generate a populated inputSchema.[18]

4. **Initializing and Configuring the MCP Server**:
   - **Instantiate Server**: An MCP server instance is created, for example, app = Server("my-db-mcp-server") using mcp.server.Server [Video Guide: 32:03]. Alternatives like FastMCP from the mcp SDK can also be used for simpler setups.[19]
   - **List Tools Handler**: An asynchronous function is defined to list the available tools. This function is decorated with @app.tool_list() (or @server.list_tools() if using the low-level server from mcp.server.lowlevel [20]). Inside this function, iterate through the ADK FunctionTool objects, convert each to an MCP tool schema using adk_to_mcp_tool_type, and return a list of these mcp_types.Tool objects [Video Guide: 33:01-33:52].

```
Python
# @app.tool_list() # Using mcp.server.Server
# async def list_mcp_tools() -> list:
#     mcp_tools_list =
#     for tool_name, adk_tool_instance in ADK_DB_TOOLS.items():
#         if not adk_tool_instance.name: # Ensure name is set for conversion
#             adk_tool_instance.name = tool_name
#         mcp_tool_schema = adk_to_mcp_tool_type(adk_tool_instance)
#         mcp_tools_list.append(mcp_tool_schema)
#     return mcp_tools_list
```

- **Call Tool Handler**: Another asynchronous function, decorated with @app.call_tool() (or @server.call_tool() [20]), is defined to handle incoming tool execution requests. This function receives the tool name and arguments from the MCP client (the ADK agent). It then finds the corresponding ADK FunctionTool (or the original Python function) and executes it with the provided arguments, returning the result in the format expected by MCP (e.g., a list of mcp_types.Content objects) [Video Guide: 34:05-35:07].

```
Python
# @app.call_tool() # Using mcp.server.Server
# async def call_mcp_tool(name: str, arguments: dict) -> list[mcp_types.Content]:
#     if name in ADK_DB_TOOLS:
#         adk_tool_instance = ADK_DB_TOOLS[name]
#         # Assuming synchronous execution for simplicity here,
#         # but actual tool execution might be async
#         result = adk_tool_instance.func(**arguments) # Or adk_tool_instance.call(arguments)
#         return
#     else:
#         raise ValueError(f"Tool {name} not found.")
```

5. Running the MCP Server via Stdio:
   To make the server listen for connections over standard input/output (which is how the ADK agent will connect to it locally using StdioServerParameters):
   - An asynchronous function (e.g., run_mcp_stdio_server) is created. This function uses mcp.server.stdio_server() as an async context manager to get read and write streams. Inside this context, app.run() (or server.run()) is called, passing these streams and initialization options (server name, version, capabilities).[16]
   - This run_mcp_stdio_server function is then called using asyncio.run() within the if __name__ == "__main__": block to start the server when the script is executed directly [Video Guide: 36:39].

D. Connecting the ADK Agent to the Local MCP Server (agent.py)

The ADK agent's agent.py file is configured to connect to this newly created local server.py.

1. **MCPToolSet Configuration**: This is similar to connecting to the Notion server example, but the StdioServerParameters will point to the local Python script.
2. **StdioServerParameters Details**:
   - command: This will be python3 (or python, or the specific path to your Python executable).[4]
   - args: A list containing the path to your local server.py script. This path should ideally be an absolute path, which can be constructed dynamically using os.path.abspath and os.path.join.[4]
   - env (Optional): If your server.py script requires any environment variables, they can be passed here.

```
Python
# In agent.py
# import os
# from google.adk.tools.mcp_tool.mcp_toolset import MCPToolSet, StdioServerParameters

# path_to_local_server_script = os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'path_to_your_server_dir', 'server.py'))
# # Adjust path as per your project structure

# tools=
#         ),
#         # tool_filter=['list_database_tables'] # Optional
#     )
# ]
```

3. **Optional tool_filter**: As before, tool_filter within MCPToolSet can be used to specify a subset of tools from the local MCP server that this particular agent should have access to.[4]

E. Running the System and Interaction

1. **Start Local MCP Server**: In one terminal window, navigate to the directory containing your server.py and run it: python server.py [Video Guide: 36:40]. The server will start and wait for a connection over stdio.
2. **Start ADK Agent**: In another terminal window, navigate to your ADK project directory and run the ADK Web UI: adk web [Video Guide: 39:13].
3. **Interaction**: The ADK agent, upon initialization, will launch the python3 server.py command as a subprocess and establish communication.
   - In the ADK Web UI, you can ask the agent "what tools do you have access

to?". It should list the custom database tools defined in your server.py (e.g., list_database_tables, query_database_table) [Video Guide: 39:38].
    - You can then issue commands like "what tables do I have access to?", "what users are in the users table?", or "add a todo for Bob to go grocery shopping". The ADK agent will use the LLM to interpret these requests, select the appropriate tool from your local MCP server, formulate the parameters, and invoke the tool. The results from your server.py will be relayed back through the agent to the UI [Video Guide: 40:01-41:21].

This two-part process (connecting to remote servers and creating local ones) provides a flexible framework for equipping ADK agents with a wide array of capabilities through the standardized Model Context Protocol.

V. Key Technologies and Resources

Successfully implementing MCP servers and integrating them with ADK agents relies on a set of core technologies, libraries, and community resources. Understanding these components is crucial for effective development.

A. Model Context Protocol (MCP) Deep Dive

1. Specification and Documentation:
   The foundational documents for MCP are its official specification and accompanying documentation. These resources define the protocol's requirements, message formats, and operational guidelines.1 The primary locations are:
    - Specification: spec.modelcontextprotocol.io
    - Documentation and Guides: modelcontextprotocol.io The Model Context Protocol GitHub organization (github.com/modelcontextprotocol) hosts the source for these documents, as well as SDKs and reference server implementations.[1] The protocol itself is an open-source project managed by Anthropic, PBC, and encourages community contributions.[1]
2. Core Components Exposed by MCP Servers:
   MCP servers make capabilities available to clients through three main primitives 5:
    - **Tools**: These are functions or actions that an AI agent can invoke. Tools allow LLMs to perform operations beyond text generation, such as querying a database, sending a message, or interacting with an external API.[5]
    - **Resources**: These represent read-only or low-side-effect data endpoints. They provide external information or context to the LLM without causing significant state changes (e.g., fetching a user's profile or a document's

content).[5]
    - **Prompts**: These are predefined, reusable conversation templates or workflows that can accept dynamic arguments and guide interactions for specific tasks. They can include context from resources and help standardize common LLM interactions, often surfaced as UI elements like slash commands.[5] Clients can discover available prompts via a prompts/list endpoint and retrieve specific prompt structures using prompts/get.[22]
3. Communication:
   MCP uses JSON-RPC 2.0 for its messaging format.5 Communication between clients and servers occurs over defined transport layers. Common transports include:
    - **Stdio (Standard Input/Output)**: Used for local MCP servers that run as a subprocess of the client application. Communication happens via the parent process's stdin/stdout pipes.[24]
    - **Server-Sent Events (SSE) over HTTP**: A common method for remote MCP servers, where the server pushes data to the client over a persistent HTTP connection.[13] This often involves a GET request to an SSE endpoint for receiving messages and potentially a separate POST endpoint for sending requests to the server.[15]
    - **Streamable HTTP**: A newer transport protocol in MCP that allows a server to manage multiple client connections via HTTP POST and GET requests over a single endpoint, optionally using SSE for streaming.[9] Smithery.ai, for example, supports deploying servers using streamable HTTP.[9]
4. Key Features Summarized:
   MCP offers several distinguishing features 5:
    - **Dynamic Tool Discovery**: Models can automatically detect and query available tools from a server.
    - **Context-Aware State Management**: MCP can help maintain context across multiple API calls, enabling more complex workflows.
    - **Built-in Security and Access Control Considerations**: While MCP itself doesn't mandate a specific security model, its architecture allows for the implementation of authentication and access control.[5]
    - **Lightweight JSON-RPC Communication**: Efficient, low-latency messaging.
    - **Interoperability and Extensibility**: Designed to integrate with diverse tools and be adaptable to new technologies.

B. Google's Agent Development Kit (ADK)

1. Overview and Purpose:
   ADK is a flexible, modular, code-first Python toolkit for developing and deploying

AI agents.3 It aims to simplify agent development, making it feel more like traditional software development, and supports architectures from simple tasks to complex multi-agent systems.3

2. Key ADK Components for MCP Integration:
   ADK provides several classes and utilities specifically for working with MCP:
   - **MCPToolSet**: This class is ADK's primary mechanism for integrating tools from an MCP server. When included in an agent's tool list, it handles connection, tool discovery, schema adaptation, and proxying tool calls to the MCP server.[4]
   - **StdioServerParameters**: Used within MCPToolSet to configure connection to an MCP server that runs as a local subprocess and communicates via stdio. Key parameters are command, args, and env.[4]
   - **SseServerParams**: Used within MCPToolSet to configure connection to a remote MCP server that communicates over HTTP using Server-Sent Events. Key parameters include url and headers.[4]
   - **FunctionTool**: An ADK class that wraps a Python function, making it discoverable as a tool. It infers schema from type hints and docstrings, which is essential for then converting to MCP format.[18]
   - **google.adk.tools.mcp_tool.conversion_utils.adk_to_mcp_tool_type**: A utility function that converts an ADK FunctionTool instance (or its schema) into an MCP-compatible tool schema (mcp_types.Tool).[17]
3. ADK Web UI:
   ADK includes a built-in web-based developer UI, typically launched with the adk web command. This UI facilitates testing, debugging, and showcasing ADK agents, including those using MCP tools.3

C. Essential Libraries and SDKs

1. MCP Python SDK:
   The official Python SDK for MCP (mcp package) provides modules for building both MCP servers and clients. Key components include mcp.server.Server (or mcp.server.lowlevel.Server), mcp.server.stdio.stdio_server, mcp.server.fastmcp.FastMCP, and mcp.types for defining MCP structures.1 The SDK handles much of the protocol complexity.
2. MCP SDKs for Other Languages:
   A significant aspect of MCP's design is its ambition for wide adoption. This is evidenced by the availability of official SDKs in multiple popular programming languages, including TypeScript, Java, C#, and Kotlin, in addition to Python.1 This multi-language support is a strategic enabler, as it substantially lowers the barrier to entry for developers from diverse technological backgrounds. Teams can build

MCP servers for their existing tools or integrate MCP clients into their applications using their preferred language stack, which in turn accelerates the growth and richness of the overall MCP ecosystem.

3. Google ADK Python Library:
   The google-adk PyPI package contains all necessary modules for ADK development in Python, including the MCP integration components.3

D. Development and Debugging Tools

1. ADK Web UI:
   As mentioned, adk web provides the primary interface for interacting with and testing ADK agents during development.26
2. MCP Inspector:
   The Model Context Protocol project provides a visual testing tool called MCP Inspector (@modelcontextprotocol/inspector). It can connect to an MCP server (especially stdio-based ones) and allow developers to list tools, call them with parameters, and inspect the raw request/response flow. This is invaluable for debugging MCP server implementations.1 It can be run using npx.27
3. Logging and Print Statements:
   Standard Python logging and print() statements are useful for debugging both ADK agent logic and custom MCP server code. However, for stdio-based MCP servers launched as subprocesses by ADK, stdout might be used for MCP communication, potentially obscuring direct print outputs. Writing logs to a file can be a more reliable debugging method for such servers.27 The video guide also notes that debugging can become more challenging with the added layer of an MCP server [Video Guide: 8:12].

E. Community and Support

1. **Official Documentation**:
   - MCP: modelcontextprotocol.io (and spec.modelcontextprotocol.io) [1]
   - ADK: google.github.io/adk-docs/ [3]
2. GitHub Repositories:
   The GitHub platform is the central nexus for both MCP and ADK development.
   - MCP: github.com/modelcontextprotocol (for SDKs, specification, servers) [1]
   - ADK: github.com/google/adk-python (for Python SDK), github.com/google/adk-web (for Web UI) [3] These repositories are not just for code; they host documentation, samples, and crucially, issue trackers where developers can report bugs, ask technical questions, and engage with the development teams and community.[18] This centralization on GitHub means that familiarity with navigating these resources is essential for any developer

working seriously with these technologies, as it's the primary channel for the latest updates, support, and contribution opportunities.

3. **Developer Communities**:
    - The video guide mentions a Skool community for AI developers with over 7,000 members and weekly coaching calls [Video Guide: 1:38].
    - The Graphlit MCP Server documentation points to a Discord community for support.[29]
    - Online forums like Reddit have dedicated communities (e.g., r/AI_Agents, r/mcp) where developers discuss MCP, ADK, and related topics.[30]

VI. Advanced Considerations and Best Practices

Implementing robust and reliable MCP solutions requires attention to several advanced topics beyond basic setup, including security, scalability, error handling, and thoughtful tool design.

A. Security in MCP

1. Authentication and Authorization:
   MCP servers that expose tools accessing sensitive data or performing critical actions must implement strong authentication and authorization mechanisms. The MCP specification itself does not mandate a specific authentication method but provides the framework where such security measures can be integrated.2
    - **API Keys/Tokens**: As seen in the Notion MCP server example, API keys can be passed via environment variables to the server process or in headers for remote connections.[4] SseServerParams in ADK supports custom headers for this purpose.[4]
    - **OAuth**: For more complex scenarios, especially with third-party services, OAuth flows might be necessary to authorize the MCP server or client to act on behalf of a user.[9] Smithery.ai mentions future OAuth support for HTTP servers.[9]
    - **Host-Mediated Security**: The MCP security model often emphasizes a host-mediated approach, where the client application (host) manages credentials and permissions.[32]
    - **Zero Trust Architecture**: Adopting principles of zero trust, where no communication is inherently trusted and continuous verification is applied, is a best practice for secure machine-to-machine communication.[33]
2. Data Privacy:
   When MCP tools handle personal or confidential data, ensuring data privacy in compliance with regulations (e.g., GDPR) is paramount. This includes secure data

transmission (encryption in transit) and storage (encryption at rest) if the MCP server caches or logs data.33

3. Input Validation and Sanitization:
   For custom MCP tools, rigorous input validation and sanitization are crucial to prevent security vulnerabilities like command injection, path traversal, or other attacks, especially if tools interact with file systems, databases, or execute shell commands.21 All parameters should be validated against their schema.

B. Scalability and Performance

1. **Local vs. Remote Servers**:
   - **Local (Stdio) Servers**: Launched as subprocesses by the ADK agent. Suitable for tools tightly coupled with the agent or accessing local resources. Performance can be good due to direct communication, but the server lifecycle is tied to the agent. Scalability is limited to the host machine's resources. Stdio connections may have limitations, such as timeouts with inactivity if proxied over HTTP by platforms like Smithery.[9]
   - **Remote (SSE/HTTP) Servers**: Can be independently scaled and managed. Better for shared tools accessed by multiple agents or clients. Latency will be a factor due to network communication. Streamable HTTP is designed for better scalability in multi-node deployments.[20]
2. **Server Design**:
   - **Efficient Tool Execution**: Tools should be designed to execute efficiently to minimize response times. Long-running operations might benefit from asynchronous execution within the tool or progress reporting mechanisms if supported by the MCP client.[21]
   - **Connection Management**: For stdio servers, ADK manages the subprocess. For remote servers, robust connection handling, retries, and timeouts are important.
3. Caching:
   MCP clients, including ADK agents, typically call list_tools to discover available tools. For remote servers, this can introduce latency on each run. The MCP Python SDK and some client implementations allow caching the list_tools response to improve performance.24 This is suitable if the tool list is stable. Cache invalidation mechanisms would be needed if tools change frequently.

C. Error Handling and Debugging

1. ADK Agent Errors:
   The ADK agent needs to gracefully handle errors that may occur during MCP interactions, such as:

    - Failure to connect to the MCP server.
    - Timeouts waiting for a response.
    - Errors returned by the MCP server during tool execution.
2. MCP Server Errors:
   MCP servers should report tool execution errors back to the client in a structured manner. The MCP specification suggests that tool errors should be reported within the result object (e.g., by setting an isError: true flag and providing error details in the content) rather than as protocol-level errors. This allows the LLM to be aware of the tool's failure and potentially adapt its strategy or inform the user.21 Internal server errors should not be exposed directly to the client.21
3. Debugging Challenges:
   Debugging interactions between an ADK agent and an MCP server can be complex due to the distributed nature of the system [Video Guide: 8:12].
    - **MCP Inspector**: A valuable tool for observing raw MCP messages between client and server.[1]
    - **Logging**: Comprehensive logging on both the client (ADK agent) and server-side is essential. As noted, stdout logging from stdio-based MCP servers might be problematic; logging to files is a more robust alternative.[27]
    - **Correlation IDs**: Using correlation IDs across requests can help trace interactions through the system.

D. Designing Effective MCP Tools

The utility of an MCP server is largely determined by the quality of the tools it exposes.

1. **Clear Schemas**: Tools must have well-defined JSON Schemas for their input parameters and output. These schemas are critical for the LLM (via the ADK agent) to understand how to use the tool correctly, what inputs are required, and what outputs to expect.[21]
2. **Granularity**: Decide on the appropriate level of granularity for tools.
    - **Atomic Tools**: Perform a single, well-defined operation. Easier to understand and combine.
    - **Composite Tools**: Encapsulate a multi-step workflow. Can be more efficient for common sequences but may be less flexible. The general recommendation is to keep tool operations focused and atomic where possible.[21]
3. **Descriptions and Examples**: Tool definitions should include clear, human-readable descriptions of what the tool does and, ideally, examples of how it should be used. This information is often passed to the LLM to aid in tool selection and parameter formulation.[21]

4. **Idempotency**: Where possible, design tools to be idempotent, meaning that calling them multiple times with the same input produces the same result without unintended side effects.
5. **Side Effects**: Clearly document any side effects a tool might have (e.g., modifying data, sending messages). Tool annotations can help convey this information to clients for appropriate user confirmation flows.[21]

By considering these advanced aspects, developers can build more robust, secure, scalable, and user-friendly MCP integrations.

VII. Table: MCP Server Connection Parameters in ADK

The Agent Development Kit (ADK) provides distinct parameter classes for connecting to MCP servers based on their transport mechanism and how they are hosted. Understanding these is key to correctly configuring an ADK agent as an MCP client.

Parameter Class	Used For	Key Configuration Fields (agent.py)	Typical command (for StdioServerParameters)	Notes
StdioServerParameters	Local MCP Servers (communicating via stdio)	command (e.g., path to executable or script interpreter), args (list of arguments), env (dictionary of environment variables)	python3, npx, path to a compiled executable	The ADK agent starts the MCP server as a subprocess. The command and args must correctly point to and execute the server. Environment variables in env are passed to this subprocess. This is used for servers like the local Python server or the npx-launched Notion server in the video guide.[4]

SseServerParams	Remote MCP Servers (communicating via SSE/HTTP)	url (the SSE endpoint of the remote server), headers (dictionary for HTTP headers, e.g., for authentication), message_url (optional, if POST requests go to a different URL than the SSE stream)	N/A	Connects to an MCP server that is already running remotely and accessible via an HTTP URL. The headers field is crucial for passing API keys or authorization tokens. This is for true client-server network communication. [4]

VIII. Table: Key MCP and ADK Resources

Navigating the MCP and ADK ecosystems requires familiarity with several key resources that provide documentation, code, examples, and tools.

Resource Type	Name / Link	Description	Relevance
MCP Specification	spec.modelcontextprotocol.io [1]	The formal definition of the Model Context Protocol, detailing message formats, transport bindings, and core concepts.	Essential for a deep understanding of the protocol's technical intricacies and for ensuring compliant implementations.
MCP Documentation	modelcontextprotocol.io [1]	Official guides, tutorials, conceptual explanations, and examples for building and using MCP clients and servers.	The primary resource for learning how to develop with MCP, understand its architecture, and find SDK information.
MCP GitHub Org	github.com/modelcontextprotocol [1]	Central repository hosting official SDKs (Python, TypeScript,	Direct access to the latest source code for SDKs and servers,

		Java, C#, Kotlin, etc.), reference server implementations, and the specification itself.	issue tracking for bugs and feature requests, and a place for community contributions.
MCP Reference Servers	github.com/modelcontextprotocol/servers [1]	A collection of reference and community-contributed MCP server implementations (e.g., Filesystem, Google Maps, Git, Slack).	Provides practical, working examples of MCP servers that can be used for learning, testing client integrations, or as a basis for new server development.
ADK Documentation	google.github.io/adk-docs/ [3]	Official documentation for Google's Agent Development Kit, covering all aspects from agent creation to tool integration and deployment.	The comprehensive guide for all ADK features, including detailed explanations of MCPToolSet, StdioServerParameters, SseServerParams, and other MCP-related components.
ADK Python GitHub	github.com/google/adk-python [3]	The official source code repository for the ADK Python SDK, including examples and an issue tracker.	Crucial for Python developers using ADK to access the latest code, report issues, find examples, and potentially contribute.
ADK Web UI GitHub	github.com/google/adk-web [26]	Source code for the ADK's built-in web-based development user interface, which is launched by adk web.	Useful for understanding the architecture of the development tool used for testing and debugging ADK agents.
MCP Inspector	github.com/modelcontextprotocol/inspector or npx @modelcontextproto	A visual testing and debugging tool for MCP servers, allowing inspection of	An invaluable utility for MCP server developers to verify their implementation,

	col/inspector [1]	list_tools, call_tool requests/responses.	debug communication issues, and understand the protocol flow.
Smithery.ai	smithery.ai [9]	A platform for discovering, deploying, hosting, and managing MCP servers.	A resource for finding publicly available, hosted MCP servers or for deploying custom MCP servers without managing infrastructure.
Notion MCP Docs	developers.notion.com/docs/mcp [11]	Notion's specific documentation for developers on how to connect to and use their official MCP server.	A concrete example of a major SaaS application providing an MCP interface, illustrating how to configure clients for a specific, widely-used tool.

IX. Conclusion

The Model Context Protocol, in conjunction with the Agent Development Kit, offers a powerful and standardized approach to extending the capabilities of AI agents. By abstracting the complexities of direct API integrations, MCP promotes **reusability, scalability, and interoperability** in how agents connect to external tools and data sources.[2] The ADK, particularly through its MCPToolSet and associated parameter classes (StdioServerParameters for locally launched processes and SseServerParams for remote HTTP/SSE connections), significantly **simplifies the development of MCP clients** within AI agents.[4]

Successfully implementing this framework requires a clear understanding of the **distinction between these connection methods** and the appropriate ADK configurations for each. For developers building custom MCP servers, meticulous **tool schema definition** (leveraging ADK's FunctionTool and adk_to_mcp_tool_type for Python-based tools), robust **error handling**, and careful **security considerations** are paramount to creating reliable and trustworthy tool extensions.[18] The process involves defining tools, converting them to MCP-compatible schemas, and exposing them via server handlers like @app.tool_list() and @app.call_tool().[20]

The MCP ecosystem is continuously evolving, with a growing number of reference servers, SDKs in various languages (Python, TypeScript, Java, etc.), and supporting tools like the MCP Inspector.[1] This multi-language support is indicative of MCP's broad ambition to become a ubiquitous standard, lowering barriers for developers across different technology stacks and fostering a richer, more diverse tool landscape. The central role of GitHub for both MCP and ADK projects underscores its importance as the primary hub for code, documentation, and community engagement.[1]

Developers are encouraged to explore these resources, experiment with building and connecting MCP servers, and participate in the growing community. By leveraging the structured approach of MCP and the development facilities of ADK, the potential to create sophisticated, versatile, and highly capable AI agents is significantly enhanced, paving the way for more innovative and impactful AI solutions.

Works cited

1. Model Context Protocol · GitHub, accessed May 31, 2025, https://github.com/modelcontextprotocol
2. MCP (Model Context Protocol): Standardizing How LLMs Connect to ..., accessed May 31, 2025, https://www.getambassador.io/blog/model-context-protocol-mcp-connecting-llms-to-apis
3. google/adk-python: An open-source, code-first Python toolkit for building, evaluating, and deploying sophisticated AI agents with flexibility and control. - GitHub, accessed May 31, 2025, https://github.com/google/adk-python
4. MCP tools - Agent Development Kit - Google, accessed May 31, 2025, https://google.github.io/adk-docs/tools/mcp-tools/
5. Model Context Protocol (MCP) - Everything You Need to Know - Zencoder, accessed May 31, 2025, https://zencoder.ai/blog/model-context-protocol
6. The Model Context Protocol (MCP): A guide for AI integration | Generative-AI - Wandb, accessed May 31, 2025, https://wandb.ai/byyoung3/Generative-AI/reports/The-Model-Context-Protocol-MCP-A-guide-for-AI-integration--VmlldzoxMTgzNDgxOQ
7. What is Model Context Protocol (MCP)? How it simplifies AI integrations compared to APIs | AI Agents That Work - Norah Sakal, accessed May 31, 2025, https://norahsakal.com/blog/mcp-vs-api-model-context-protocol-explained/
8. modelcontextprotocol/servers: Model Context Protocol ... - GitHub, accessed May 31, 2025, https://github.com/modelcontextprotocol/servers
9. Deployments - Smithery Documentation, accessed May 31, 2025, https://smithery.ai/docs/build/deployments
10. Smithery MCP Server Deployment, accessed May 31, 2025, https://smithery.ai/server/@cmtkdot/mcp-server-smithery
11. Model Context Protocol (MCP) - Notion API, accessed May 31, 2025,

https://developers.notion.com/docs/mcp
12. Remote MCP servers - Anthropic API, accessed May 31, 2025, https://docs.anthropic.com/en/docs/agents-and-tools/remote-mcp-servers
13. Register and discover remote MCP servers in your API inventory - Learn Microsoft, accessed May 31, 2025, https://learn.microsoft.com/en-us/azure/api-center/register-discover-mcp-server
14. Use MCP servers in VS Code (Preview), accessed May 31, 2025, https://code.visualstudio.com/docs/copilot/chat/mcp-servers
15. Use Google ADK and MCP with an external server | Google Cloud Blog, accessed May 31, 2025, https://cloud.google.com/blog/topics/developers-practitioners/use-google-adk-and-mcp-with-an-external-server
16. 3 steps to build an MCP server from scratch - Merge.dev, accessed May 31, 2025, https://www.merge.dev/blog/how-to-build-mcp-server
17. Building AI Agents with Google ADK, FastAPI, and MCP - DEV ..., accessed May 31, 2025, https://dev.to/timtech4u/building-ai-agents-with-google-adk-fastapi-and-mcp-26h7
18. Bug: inputSchema is None for no-argument FunctionTool after ..., accessed May 31, 2025, https://github.com/google/adk-python/issues/948
19. MCP Server in Python — Everything I Wish I'd Known on Day One | DigitalOcean, accessed May 31, 2025, https://www.digitalocean.com/community/tutorials/mcp-server-python
20. The official Python SDK for Model Context Protocol servers and clients - GitHub, accessed May 31, 2025, https://github.com/modelcontextprotocol/python-sdk
21. Tools - Model Context Protocol, accessed May 31, 2025, https://modelcontextprotocol.io/docs/concepts/tools
22. Prompts - Model Context Protocol, accessed May 31, 2025, https://modelcontextprotocol.io/docs/concepts/prompts
23. Specification - Model Context Protocol, accessed May 31, 2025, https://spec.modelcontextprotocol.io/
24. Model context protocol (MCP) - OpenAI Agents SDK, accessed May 31, 2025, https://openai.github.io/openai-agents-python/mcp/
25. Model Context Protocol (MCP) - Documentation - What is Trae IDE?, accessed May 31, 2025, https://docs.trae.ai/ide/model-context-protocol
26. google/adk-web: Agent Development Kit Web (adk web) is the built-in developer UI that is integrated with Agent Development Kit for easier agent development and debugging. - GitHub, accessed May 31, 2025, https://github.com/google/adk-web
27. How to Debug MCP Server with Anthropic Inspector? - Snyk, accessed May 31, 2025, https://snyk.io/articles/how-to-debug-mcp-server-with-anthropic-inspector/
28. Model Context Protocol (MCP) - Agent Development Kit - Google, accessed May 31, 2025, https://google.github.io/adk-docs/mcp/
29. Model Context Protocol (MCP) Server for Graphlit Platform - GitHub, accessed

May 31, 2025, https://github.com/graphlit/graphlit-mcp-server
30. ADK(agent development kit) with MCP(model context protocol) is it good if we are using only mcp for cloud data storage pulling : r/AI_Agents - Reddit, accessed May 31, 2025, https://www.reddit.com/r/AI_Agents/comments/1ky4agz/adkagent_development_kit_with_mcpmodel_context/
31. What you need to know about the Model Context Protocol (MCP) - Merge.dev, accessed May 31, 2025, https://www.merge.dev/blog/model-context-protocol
32. Model Context Protocol (MCP) vs Function Calling: A Deep Dive into AI Integration Architectures - MarkTechPost, accessed May 31, 2025, https://www.marktechpost.com/2025/04/18/model-context-protocol-mcp-vs-function-calling-a-deep-dive-into-ai-integration-architectures/
33. MCP Security Best Practices: Ultimate Guide for 2025 - BytePlus, accessed May 31, 2025, https://www.byteplus.com/en/topic/541335
34. MCP Servers: The Ultimate Guide to Model Context Protocol - VideoSDK, accessed May 31, 2025, https://www.videosdk.live/developer-hub/ai/mcp-servers
35. MCPToolset.from_server fails on Windows with NotImplementedError from asyncio subprocess in Python 3.13, 3.12. 3.11 · Issue #585 - GitHub, accessed May 31, 2025, https://github.com/google/adk-python/issues/958/linked_closing_reference?reference_location=REPO_ISSUES_INDEX
36. Support Asynchronous Initialization for root_model in ADK Agents #28 - GitHub, accessed May 31, 2025, https://github.com/google/adk-python/issues/28