Model Context Protocol (MCP) と Agent Development Kit (ADK): AIエージェント開発の標準化と効率化

1. はじめに：AIエージェント開発の新たな潮流

近年、大規模言語モデル（LLM）の進化に伴い、単なるテキスト生成にとどまらず、外部ツールやデータソースと連携して複雑なタスクを実行する「AIエージェント」の開発が活発化しています。しかし、LLMと外部リソース間の連携方法が標準化されていないため、開発者はアプリケーションごとに個別の連携ロジックを実装する必要があり、開発効率や相互運用性の面で課題がありました。

このような背景の中、AIエージェント開発をより標準化し、効率化するための技術として、**Model Context Protocol (MCP)** と **Agent Development Kit (ADK)** が注目されています。MCPはLLMと外部リソース間の通信プロトコルを標準化し、ADKは高度なAIエージェントを構築するための柔軟な開発フレームワークを提供します。

本レポートでは、提供された情報源に基づき、MCPとADKの概要、主要な機能、技術的特徴、そして両者の連携について、初心者にも理解できるよう分かりやすく解説します。

2. Model Context Protocol (MCP): AIと世界をつなぐ標準インターフェース

2.1. MCPの目的：AIアプリケーションのための「USB-Cポート」

Model Context Protocol (MCP) の主な目的は、アプリケーションが大規模言語モデル (LLM) に対して、外部のデータソースやツールといった「コンテキスト」を提供するための標準化された方法を確立することです [1]。これは、様々なデバイスを共通のコネクタで接続できるUSB-Cポートのように、多様なAIモデル、データソース、ツールを標準的な方法で接続するためのインターフェースと考えることができます [1]。

従来、LLMをアプリケーションに組み込む際には、利用するデータソースやAPIごとに個別の連携コードを実装する必要がありました。これは「M x N 問題」とも呼ばれ、M個のアプリケーションとN個のツールが存在する場合、最大で M * N 通りの連携方法を開発・維持する必要が生じます [4]。MCPは、ツール開発者がN個のMCPサーバーを、アプリケーション開発者がM個のMCPクライアントを実装するだけで済むようにし、この問題を解決することを目指します [4]。

MCPを利用することで、開発者は以下のようなメリットを得られます [1]。

- LLMが直接利用できる、構築済みの連携機能（MCPサーバー）の活用。
- 特定のLLMプロバイダーやベンダーに縛られず、柔軟に切り替えられる可能性。
- インフラストラクチャ内でデータを保護するためのセキュリティベストプラクティスの適用促

進。

結果として、MCPはLLM、データソース、ツール間の相互運用性を高め、AIエージェントや複雑なワークフローの開発を容易にし、データセキュリティを向上させることを目指しています[1]。

2.2. MCPのコアコンセプト：ホスト、クライアント、サーバー

MCPはクライアントサーバー型のアーキテクチャを採用しており、主に以下の3つの参加要素で構成されます[1]。

- **MCP ホスト (Host):** ユーザーが直接操作するアプリケーションであり、MCPを介してデータやツールにアクセスしたいプログラムです。例えば、AIチャットインターフェース（Claude Desktopなど）、AI統合開発環境（IDE、Cursorなど）、カスタムAIエージェントなどが該当します[1]。ホストアプリケーションは、内部にMCPクライアントを含みます[2]。
- **MCP クライアント (Client):** ホストアプリケーション内に存在し、特定のMCPサーバーとの1対1の接続を維持・管理するコンポーネントです[1]。ホストからの要求を受け取り、MCPプロトコルに従ってサーバーと通信します[2]。
- **MCP サーバー (Server):** 特定のデータソースやツールへのアクセス機能を提供する軽量なプログラムです。標準化されたMCPを介して、その機能（ツール、リソース、プロンプトなど）をクライアントに公開します[1]。サーバーは、ローカルのファイルシステムやデータベース[1]、あるいはリモートのAPIサービス[1]など、様々なリソースへのブリッジとして機能します。

このアーキテクチャにより、ホストアプリケーションはMCPクライアントを通じて様々なMCPサーバーと接続し、標準化された方法で多様な機能を利用できるようになります。

2.3. プロトコルの構造と主要機能

MCPは、開発ツールと言語サーバー間の通信を標準化する (https://microsoft.github.io/language-server-protocol/) から着想を得ており、AIアプリケーションのエコシステムにおいてコンテキストとツールを統合する方法を標準化することを目指しています[5]。

ベースプロトコル:

- メッセージ形式: 通信には **JSON-RPC 2.0** フォーマットを使用します[5]。これにより、構造化されたリクエストとレスポンスの交換が可能になります。具体的なメッセージ構造はTypeScriptスキーマで定義されています[5]。
- 接続: ステートフルな接続を維持し、セッション中の状態管理を可能にします[5]。
- 機能ネゴシエーション: 接続確立時に、クライアントとサーバー間でサポートする機能（後述）について情報を交換し、合意します[4]。

主要機能 **(Features):**

MCPサーバーはクライアント（ひいてはLLMやユーザー）に対して、主に以下の機能を提供できます [4]。

- リソース **(Resources):** ユーザーやAIモデルが利用するためのコンテキスト情報やデータです。ファイルの内容、データベースのレコード、プロジェクトのコードなどが該当します [4]。アプリケーション（ホスト）が制御し、LLMに背景情報として提供されます [4]。
  - 例: IDE内のMCPサーバーが、現在開いているファイルの内容やプロジェクト構造を「リソース」として提供し、LLMがコード補完やリファクタリングを行う際の文脈として利用する [5]。CRM連携サーバーが顧客情報を「リソース」として提供し、AIアシスタントが顧客対応を支援する [5]。
- プロンプト **(Prompts):** ユーザー向けの定型的なメッセージや、特定のタスクを実行するためのテンプレート化されたワークフローです [4]。ユーザーが直接起動することが想定されています [4]。
  - 例: ドキュメント要約用のプロンプト。ユーザーが特定のドキュメントリソースを選択し、「要約」プロンプトを実行すると、LLMがそのリソースを要約する [5]。特定の形式（例: メール、レポート）でテキストを生成するプロンプト [5]。
- ツール **(Tools):** LLM自身が判断して実行できる具体的なアクションや関数です [4]。LLMが自律的に外部システムと対話するために使用されます [4]。
  - 例: データベース検索ツール。LLMが必要な情報を得るために、「顧客DB検索」ツールを自然言語で呼び出す [5]。コード実行ツール、メール送信ツール、外部API連携ツールなど [5]。

一方、クライアントはサーバーに対して以下の機能を提供できます [5]。

- サンプリング **(Sampling):** サーバー主導のエージェント的な振る舞いや、再帰的なLLMインタラクションを可能にする機能です [5]。

これらの機能に加えて、設定管理、進捗追跡、キャンセル処理、エラー報告、ロギングといったユーティリティ機能もプロトコルに含まれています [5]。

表**1: MCP**の主要機能概要

機能	提供者	説明	主な利用者	利用シナリオ例
リソース	サーバー	LLMやユーザーが利用するコンテキスト情報やデー	LLM/ユーザー	コード分析時のファイル内容提供、顧客情報参照

		タ[5]		[5]
プロンプト	サーバー	ユーザー向けの定型メッセージやワークフローテンプレート[5]	ユーザー	ドキュメント要約の実行、定型レポート生成[5]
ツール	サーバー	LLMが実行できる関数やアクション[5]	LLM	データベース検索、外部API呼び出し、コード実行[5]
サンプリング	クライアント	サーバー主導のエージェント的振る舞いや再帰的LLMインタラクション[5]	サーバー	より複雑な自律的エージェントワークフロー (詳細は仕様参照)

2.4. セキュリティと信頼性

MCPは外部システムとの連携を可能にするため、セキュリティと信頼性が重要な考慮事項となります。仕様では、以下の原則が強調されています[5]。

- ユーザーの同意と制御
- データプライバシー
- ツールの安全性
- LLMサンプリングの制御

実装者は、これらの原則を念頭に置き、適切な認証・認可メカニズムやサンドボックス環境などを導入することが推奨されます[5]。

2.5. MCPエコシステム：SDKと実装例

MCPの普及を促進するため、プロトコルの仕様だけでなく、開発を支援するSDKや実装例が提供されています。

- 公式**SDK:** GitHubリポジトリでは、TypeScript, Python, Java, Kotlin, C# といった主要なプログラミング言語向けのSDKが提供されており、クライアントおよびサーバーの開発を容易にします[4]。
- サーバー実装例**:** ファイルシステム操作、Git/GitHub連携、Slack、Google Maps、各種データベース（PostgreSQL, SQLite）連携など、多様な公式・コミュニティ開発のサーバー実装例が公開されており、すぐに試したり、カスタムサーバー開発の参考にしたりできます[8]。
- クライアント実装例**:** Claude Desktop, Cursor IDE, Continue.dev (IDE拡張), Genkit

(Firebase), ADK, LangGraph, OpenAI Agents SDK など、多くのアプリケーションやフレームワークがMCPクライアントとして対応しており、MCPサーバーを利用できる環境が広がっています [4]。

これらのリソースにより、開発者はMCPの導入と活用をスムーズに進めることができます。

3. Agent Development Kit (ADK): 高度なエージェント構築のためのフレームワーク

3.1. ADKの紹介：目的とGoogleエコシステムへの注力

Agent Development Kit (ADK) は、Googleが提供するオープンソースの**Python**ツールキットであり、AIエージェントおよびマルチエージェントシステムの構築、評価、デプロイを目的としています [10]。

ADKの設計における主な目標は、エージェントの振る舞いに対して柔軟性、モジュール性、そしてきめ細かな制御を開発者に提供することです [10]。特に、設定ファイル主導ではなく、コードファーストのアプローチを採用している点が特徴です [10]。

ADKはGoogleのGeminiモデルやVertex AI、その他のGoogle Cloudツールとの緊密な統合を重視して設計されていますが [12]、LiteLLMの統合によりAnthropicのClaude、Metaの LLaMA、Mistral AIなど、他の多くのLLMも利用可能であり、特定のベンダーにロックインされない柔軟性も備えています [11]。ADKはGoogle内部の製品（Agentspace、Google Customer Engagement Suiteなど）でも利用されている実績のあるフレームワークです [11]。

3.2. ADK開発の中核となる柱

ADKは、高度なエージェント開発を支援するために、いくつかの重要な機能を提供しています。

- コードファースト哲学と柔軟性:
  エージェントのロジック、ツールの定義、ワークフローの編成（オーケストレーション）が、設定ファイルではなくPythonコードで直接記述されます 10。これにより、複雑な条件分岐やカスタムロジックの実装が容易になり、Pythonのエコシステム（テストフレームワーク、バージョン管理システムなど）との親和性が高まります 10。

  ```python
  # 簡単なLLMエージェントの例 [13]
  from google_adk.agents.llm_agent import LlmAgent
  from google_adk.llm.models import Gemini

  llm_agent = LlmAgent(
      model=Gemini(model_name="gemini-pro"),
      prompt="与えられた質問に答えてください。\n質問: {question}"
  ```

```
)
response = llm_agent.run(question="今日の天気は？")
print(response)
```

- 高度なオーケストレーション戦略:
  ADKは、エージェントの実行フローを制御するための複数の方法を提供します 13。
  - 予測可能なパイプライン: SequentialAgent, ParallelAgent, LoopAgent といったワークフローエージェントを使用して、ステップが決まっているタスクを実行するパイプラインを定義できます [13]。例えば、質問応答エージェントの後に要約エージェントを実行する逐次処理パイプラインを構築できます [13]。
  - 適応的なルーティング: LlmAgent は、LLMの推論能力を活用して、状況に応じて次に実行すべきステップや使用するツールを動的に決定できます [13]。
  - 開発者の視点: 一部の開発者からは、これらの異なるエージェントタイプ（LlmAgent, Sequential, Parallel, Loop, Custom）はやや過剰設計であり、統一されたワークフロー概念の方がシンプルかもしれないという意見もあります [14]。
- マルチエージェントシステムの設計:
  ADKは、複数の専門化されたエージェントを階層構造で構成し、連携させることで、複雑なアプリケーションを構築することを得意としています 10。これにより、モジュール性、スケーラビリティが向上し、タスクの委任や専門知識の組み合わせが可能になります 11。AgentTool という機能を使えば、あるエージェントを別のエージェントのツールとして利用することもできます 14。例えば、ルートエージェントが挨拶タスクを「挨拶エージェント」に、別れの挨拶を「別れエージェント」に委任するような構成が可能です 17。
- 豊富なツールエコシステムの活用:
  エージェントに様々な能力を与えるためのツール連携機能が充実しています 18。
  - 組み込みツール: 検索やコード実行などの事前構築済みツール [11]。
  - カスタム関数: Python関数をツールとして定義可能。LLMがツールを正しく理解・利用するためには、関数のdocstring（説明文）を明確に記述することが重要です [17]。
  - サードパーティライブラリ: LangChain, LlamaIndex, CrewAIなどの既存ライブラリとの統合 [11]。
  - 他のエージェント: LangGraphやCrewAIで構築されたエージェントをツールとして利用 [11]。
  - **MCPツール: Model Context Protocol (MCP)** サーバーによって公開されているツールを MCPToolset を介して統合可能 [18]。これにより、ADKエージェントは広範なMCPエコシステムのツールを利用できます。
  - **OpenAPI**仕様: OpenAPI仕様からツールを生成 [10]。
  - **Google Cloud**ツール**/API:** Google Cloudの各種サービスとの連携 [13]。
- 効率的なデプロイ経路:
  開発したエージェントを様々な環境にデプロイするためのオプションが用意されています 10。

    - ローカル環境での実行。
    - Dockerコンテナ化によるポータビリティの確保 [13]。
    - Cloud Runのようなサーバーレスプラットフォームへのデプロイ [10]。
    - Vertex AI Agent Engine を利用した、スケーラブルなマネージド環境へのデプロイ（Google Cloudへの最適化パス）[10]。
- 統合された評価機能:
  エージェントのパフォーマンスを体系的に評価するための機能が組み込まれています 10。
    - 最終的な応答の品質評価。
    - 事前定義されたテストケース（例: evaluation.test.json）に対する、ステップごとの実行軌跡の評価 (AgentEvaluator.evaluate() を使用) [11]。
    - 開発者の視点: 評価機能の使い勝手（Developer Experience）については、改善の余地があるとのフィードバックも見られます [14]。
- 責任あるAIの構築支援:
  ADKは、責任あるAIのパターンとベストプラクティスをエージェント設計に組み込むためのガイダンスを提供します 13。これには、入力検証や安全チェックのためのガードレールコールバック（例: before_model_callback, before_tool_callback）が含まれます 14。
- ユニークな機能:
  テキストベースの対話を超えて、より自然なインタラクションを実現するための双方向の音声・映像ストリーミング機能 11 や、セッション間でファイルやバイナリデータを保持するためのアーティファクト管理機能 14 も提供されています。

表**2: ADK**の主要機能概要

機能	説明	利点
コードファースト	エージェントロジック、ツール、オーケストレーションをPythonコードで定義 [10]	高い柔軟性、テスト容易性、バージョン管理の容易さ [10]
オーケストレーション	予測可能なパイプライン（Sequential等）と適応的ルーティング（LlmAgent）を提供 [13]	構造化されたタスク実行と動的な振る舞いの両方を実現
マルチエージェント	複数の専門エージェントを階層的に構成可能 [10]	モジュール性、スケーラビリティ、複雑なタスクの分担・連携 [11]
ツールエコシステム	組み込みツール、カスタム関数、	エージェントに広範な機能と能力

	サードパーティライブラリ、MCPツール、他のエージェント等、多様なツール連携をサポート [18]	を付与
デプロイ	ローカル、コンテナ（Docker）、Cloud Run、Vertex AI Agent Engine等、多様なデプロイ先をサポート [10]	開発から本番環境へのスムーズな移行とスケーラビリティ
評価	最終応答品質とステップごとの実行軌跡を評価する機能を内蔵 [10]	エージェントのパフォーマンスを体系的に測定・改善
責任あるAI	責任あるAIパターンとガードレール機能を提供 [13]	安全で信頼性の高いエージェントの構築を支援
ストリーミング	双方向の音声・映像ストリーミングに対応 [11]	より自然でリッチなユーザーインタラクションを実現

3.3. ADKの対象開発者と応用可能性

ADKは、シンプルなAIエージェントから、きめ細かな制御が必要な高度なマルチエージェントシステムまで、幅広い開発ニーズに対応します [12]。特に、GeminiモデルやGoogle AIツールを活用したい開発者、コードによる直接的な制御を好む開発者、複雑なワークフローやエージェント連携を実装したい開発者にとって強力な選択肢となります [12]。初心者でもシンプルなエージェントから始めやすく、熟練開発者が必要とする高度な機能も提供されています [13]。

応用可能性としては、複雑な業務プロセスの自動化、特定のドメインに特化したインテリジェントアシスタント、データ収集・分析・レポート生成を行うエージェント群、マルチモーダルな対話アプリケーション、マルチエージェントシステムの協調動作に関する研究プラットフォームなどが考えられます。

4. シナジーの実践：ADKとMCPの連携

ADKの柔軟なエージェント構築能力と、MCPの標準化された外部連携能力を組み合わせることで、より強力で汎用性の高いAIアプリケーションを効率的に開発できます。

4.1. 連携の橋渡し：MCPToolsetによるMCPツールの利用

ADKは、MCPサーバーによって公開されるツールを直接サポートしています [11]。この連携を実現する中心的なメカニズムが、ADK内に用意されている MCPToolset クラスです [18]。

MCPToolset は以下の役割を担います [3]。

1. 指定された接続情報に基づき、ターゲットのMCPサーバーに接続します。
2. 接続後、MCPサーバーに対して利用可能なツールのリストを問い合わせます（内部的にMCPの list_tools メソッドを呼び出します）。
3. 取得したツールのリストを、ADKエージェントが他のツールと同様に扱える形式に変換して提供します。
4. ADKエージェント（内部のLLM）がMCPツールを使用すると判断した場合、MCPToolsetはその要求を受け取り、MCPサーバーに対してツールの実行を指示します（内部的にMCPの call_tool メソッドを呼び出します）。
5. MCPサーバーからの実行結果を受け取り、ADKエージェントに返します。

この連携のために、ADKは mcp ライブラリに依存しています [20]。

4.2. 接続メカニズム：Stdio と HTTP/SSE

MCPToolset は、MCP仕様で定義されている2つの主要な接続方法をサポートしており、状況に応じて使い分けることができます [3]。

- **Stdio (Standard Input/Output):**
  - 概要: MCPサーバーがADKエージェントと同じマシン上でローカルなサブプロセスとして実行される場合に利用されます [3]。プロセス間の標準入出力を介して通信します。
  - **ADK**での設定: MCPToolset.from_server() に StdioServerParameters を渡し、サーバープロセスを起動するためのコマンド (command) と引数 (args) を指定します [18]。必要に応じて環境変数 (env) も渡せます [18]。
  - 利点: ローカルツール（例：ファイルシステムアクセス、ローカルスクリプト実行）の連携がシンプル。ネットワーク遅延がない。
  - 欠点: サーバーをローカルで実行する必要がある。
- **HTTP over SSE (Server-Sent Events):**
  - 概要: MCPサーバーがリモートで実行されており、URLを通じてアクセスする場合に利用されます [3]。HTTP接続を確立し、サーバーからクライアントへイベントをプッシュするSSE技術を用いて通信します。
  - **ADK**での設定: MCPToolset.from_server() に SseServerParams を渡し、サーバーのURL (url) と、必要に応じて認証情報などを含むヘッダー (headers) を指定します [3]。
  - 利点: リモートにあるツールやAPIにアクセス可能。サーバーはどこにでもホストできる。
  - 欠点: ネットワーク接続が必要。ネットワーク遅延が発生する可能性がある。認証などのセットアップが必要になる場合がある。

表3: MCP接続方法の比較

方法	説明	主な用途	利点	欠点
Stdio	ローカルプロセスとして起動したサーバーと標準入出力で通信 [3]	ローカルツール	シンプルな設定、低遅延	サーバーをローカル実行する必要あり
HTTP/SSE	リモートサーバーとHTTP/SSEで通信 [3]	リモートツール	場所を選ばないサーバー、多様なAPI連携が可能	ネットワーク依存、遅延可能性、認証設定が必要な場合あり

4.3. 実装ガイド：ADKエージェントとMCPサーバーの接続（コード例）

以下に、ADKエージェントをローカルのファイルシステムMCPサーバーに接続する具体的なPythonコード例を示します（[18] を基に構成）。

Python

```
#./adk_agent_samples/mcp_agent/agent.py (抜粋・解説付き)
import asyncio
from google.adk.agents.llm_agent import LlmAgent
from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService
from google.adk.artifacts.in_memory_artifact_service import InMemoryArtifactService
# MCPToolsetと接続パラメータクラスをインポート
from google.adk.tools.mcp_tool.mcp_toolset import MCPToolset, StdioServerParameters
from google.genai import types # ユーザー入力用

# MCPサーバーからツールを非同期で取得する関数
async def get_tools_async():
    print("ファイルシステム MCP サーバーへの接続を試みます...")
    # MCPToolset.from_server を使用して接続とツール取得
    # StdioServerParameters でローカルプロセス起動情報を指定
    tools, exit_stack = await MCPToolset.from_server(
        connection_params=StdioServerParameters(
            command='npx', # MCPサーバー起動コマンド
            args=[ # コマンド引数
                "-y",
```

```
                "@modelcontextprotocol/server-filesystem", # ファイルシステムサーバーのnpmパッケージ
                "/path/to/your/folder" # ★アクセスを許可するフォルダパスを指定
            ],
        )
    )
    print("MCP Toolset が正常に作成されました。")
    # 取得したツールリストと、接続終了処理用のexit_stackを返す
    return tools, exit_stack

# MCPツールを持つADKエージェントを非同期で作成する関数
async def get_agent_async():
    # 上記関数でツールを取得
    tools, exit_stack = await get_tools_async()
    print(f"MCP サーバーから {len(tools)} 個のツールを取得しました。")
    # LlmAgentを作成し、toolsパラメータに取得したMCPツールを渡す
    root_agent = LlmAgent(
        model='gemini-1.5-flash', # 使用するLLMモデル
        name='filesystem_assistant',
        instruction='利用可能なツールを使用して、ユーザーがローカルファイルシステムと対話するのを支援します。',
        tools=tools, # ★ここでMCPツールをエージェントに組み込む
    )
    return root_agent, exit_stack

# メインの非同期処理
async def async_main():
    session_service = InMemorySessionService()
    artifacts_service = InMemoryArtifactService()
    session = session_service.create_session(
        state={}, app_name='mcp_filesystem_app', user_id='user_fs'
    )
    # ユーザーからのクエリ（例）
    query = "指定したフォルダ内のファイルをリストしてください" # ★実際のフォルダ名に合わせて調整
    content = types.Content(role='user', parts=[types.Part(text=query)])

    # エージェントを取得
    root_agent, exit_stack = await get_agent_async()

    # Runnerを初期化してエージェントを実行
```

```
    runner = Runner(
        app_name='mcp_filesystem_app',
        agent=root_agent,
        artifact_service=artifacts_service,
        session_service=session_service,
    )
    print("エージェントを実行中...")
    events_async = runner.run_async(
        session_id=session.id, user_id=session.user_id, new_message=content
    )
    async for event in events_async:
        print(f"受信したイベント: {event}")

    # ★重要: 処理終了後、exit_stackを使ってMCPサーバー接続をクリーンアップ
    print("MCP サーバー接続を閉じます...")
    await exit_stack.aclose()
    print("クリーンアップ完了。")

if __name__ == '__main__':
    try:
        # 非同期メイン関数を実行
        asyncio.run(async_main())
    except Exception as e:
        print(f"エラーが発生しました: {e}")
```

コードのポイント:

- MCPToolset.from_server() を async with で使用するか、返された exit_stack を使って await exit_stack.aclose() を呼び出し、接続を適切に閉じる必要があります [18]。これにより、サーバープロセスが正しく終了します。
- StdioServerParameters の args 内のパス /path/to/your/folder は、実際にアクセスしたいローカルフォルダのパスに置き換える必要があります [18]。
- 取得した tools オブジェクトを LlmAgent の tools 引数に渡すだけで、エージェントはMCPサーバーが提供するツール（この例ではファイルリスト取得、ファイル読み書きなど）を利用できるようになります [18]。
- 必要に応じて、MCPサーバーから取得するツールをフィルタリングすることも可能です [21]。

4.4. 連携アプローチの利点

ADKとMCPを連携させることには、以下のような大きな利点があります。

- **既存ツールの活用:** ADKエージェントは、MCPエコシステムで利用可能な、増え続ける多様なMCPサーバー（ファイルシステム、Git、データベース、各種API連携など）に容易にアクセスできます [11]。ツールごとに個別の連携コードを書く必要がなくなります。
- **標準化:** 外部機能の連携方法がMCPという標準プロトコルに準拠するため、エージェントの設計が整理され、一貫性が保たれます [19]。
- **モジュール性:** エージェントのコアロジック（ADKで実装）と、外部ツールの具体的な実装（MCPサーバーで実装）を明確に分離できます。これにより、各コンポーネントの独立性が高まり、開発・保守が容易になります。特定のMCPサーバーは、ADKだけでなく、MCPに対応した他のフレームワーク（LangGraph, OpenAI SDKなど）からも再利用できます [4]。
- **柔軟性:** ADKエージェント内で、ADKネイティブのツール、カスタムPython関数、そしてMCP経由のツールを自由に混在させて利用できます [11]。
- **関心の分離:** この組み合わせは、効果的な関心の分離を実現します。ADKはエージェントの「思考」（推論、計画、オーケストレーション、状態管理）を担当し、MCPはエージェントの「行動」や「知覚」（外部ツール実行、データアクセス）のための標準化されたインターフェースを提供します。ADKはツールの内部実装を知る必要がなく、MCPサーバーはエージェントの内部ロジックを知る必要がありません。この疎結合なアーキテクチャは、堅牢で保守性の高いエージェントアプリケーションの構築に貢献します。

5. 実践的な応用と利用開始

5.1. ユースケース例：GitHub連携エージェント

ADKとMCPの連携による具体的なユースケースとして、GitHubリポジトリと対話するエージェントを考えてみましょう。

1. **セットアップ:**
   - GitHub操作機能を提供するMCPサーバー（例: @modelcontextprotocol/server-github [8]）をセットアップし、必要に応じて認証情報（GitHubトークンなど）を設定します。
   - ADKでエージェントを定義し、MCPToolset を使用してこのGitHub MCPサーバーに接続します。取得したツール（リポジトリ情報の取得、コミット履歴の表示、Issueの作成など）をエージェントに組み込みます。
2. **ユーザーの指示:** ユーザーがエージェントに「最新のコミットメッセージを要約して」と指示します。
3. **エージェントの処理:**
   - ADKエージェント内のLLMがユーザーの指示を解釈し、「最新コミット取得」ツール（MCP経由）を使用する必要があると判断します。
   - ADKの MCPToolset が、GitHub MCPサーバーに対して「最新コミット取得」ツールの実行を要求します。

    - GitHub MCPサーバーは、内部でGitHub APIを呼び出すなどして最新コミット情報を取得し、結果を MCPToolset に返します。
    - MCPToolset は結果をADKエージェントに渡します。
    - ADKエージェント内のLLMが、受け取ったコミット情報を基に要約を生成します。
4. 応答: エージェントがユーザーに要約結果を提示します。

このように、ADKがエージェントの思考と応答生成を担当し、MCPが標準化された方法でGitHubという外部サービスとの連携を仲介します。同様の仕組みで、ファイルシステム操作 [18]、データベース問い合わせ [8]、Ankiカード作成 [23] など、様々なタスクを実行するエージェントを構築できます。

5.2. ADKプロジェクトの開始：クイックガイド

ADKを使った開発を始めるための基本的な手順は以下の通りです [17]。

1. 環境設定:
    - Python 3.10以上がインストールされていることを確認します。
    - 仮想環境を作成して有効化します (python -m venv.venv, source.venv/bin/activate など) [24]。
    - Google Cloudサービス（Vertex AIなど）を利用する場合は、Google Cloudプロジェクトを設定し、認証情報（Application Default Credentials）をセットアップします [24]。
2. ADKのインストール:
    - 仮想環境内で pip install google-adk を実行します [24]。MCPツールを利用する場合は pip install mcp も必要になることがあります [18]。
3. プロジェクト構造の作成:
    - エージェント用のディレクトリを作成します（例: my_agent/）。
    - 必要なファイルを作成します: __init__.py, agent.py（エージェント定義用）, .env（APIキーなど環境変数用）[24]。一部の開発者からは、この手動でのフォルダ作成ステップは不要ではないかとの意見もあります [14]。
4. エージェントとツールの定義:
    - agent.py 内に、LlmAgent や他のADKクラスを使ってエージェントのロジックを記述します [13]。
    - 必要に応じて、Python関数としてカスタムツールを定義します。docstringを明確に記述することが重要です [17]。
    - MCPツールを利用する場合は、MCPToolset を使ってサーバーに接続し、ツールを取得するコードを追加します [18]。

5.3. ADKエージェントの実行と対話

作成したADKエージェントは、ADK CLIツールを使って簡単に実行し、対話することができます [14]。

- **adk run <agent_module>:** ターミナル上で直接エージェントと対話します。
- **adk web:** ローカルマシン上で開発用のWeb UIを起動します（通常 http://localhost:8000）[24]。ブラウザを通じてエージェントを選択し、チャット形式で対話しながらテストやデバッグを行うことができます。
- **adk api_server <agent_module>:** エージェントをAPIサーバーとして起動し、他のアプリケーションからHTTPリクエスト経由で利用できるようにします。

これらのコマンドにより、開発サイクルを通じてエージェントの動作を容易に確認・検証できます。

6. 結論：標準化と効率化によるエージェント開発の未来

Model Context Protocol (MCP) と Agent Development Kit (ADK) は、それぞれがAIエージェント開発における重要な課題に取り組む技術です。

- **MCP** は、LLMと外部ツール・データソース間の連携における標準化レイヤーを提供します [1]。これにより、異なるシステム間の相互運用性が向上し、開発者は再利用可能な連携コンポーネント（MCPサーバー）のエコシステムを活用できるようになります。これは、AIアプリケーション開発における「M x N 問題」を解決し、イノベーションを加速させる可能性を秘めています。
- **ADK** は、高度なAIエージェントやマルチエージェントシステムを構築するための、強力かつ柔軟なコードファーストの**Python**フレームワークを提供します [10]。特にGoogle Cloudエコシステムとの親和性が高い設計でありながら、多様なモデルやツールに対応するオープン性も備えています。きめ細かな制御、豊富なオーケストレーション機能、マルチエージェント設計、評価、デプロイ支援により、開発者は複雑な要件に対応するエージェントを効率的に開発できます。

そして、これら二つの技術のシナジーは非常に強力です。ADKの高度なエージェント構築・管理能力と、MCPの標準化されたツール連携能力を組み合わせることで、開発者はより強力で、汎用性が高く、保守性に優れたAIアプリケーションを効率的に構築できます [11]。ADKがエージェントの「頭脳」と構造を提供し、MCPが標準化された「手足」や「感覚器」として機能する、という役割分担により、堅牢なシステム設計が可能になります。

MCPとADK、そして関連するAgent2Agentプロトコル [19] のような技術の発展は、AIエージェントがより自律的に、そして相互に連携しながら複雑なタスクを実行する未来を示唆しています。これらのオープンな技術と活発なコミュニティ [4] は、AI開発の新たな標準を形成し、次世代のインテリジェントアプリケーションの実現を加速させるでしょう。

引用文献

1. Model Context Protocol: Introduction, 4月 12, 2025にアクセス、 https://modelcontextprotocol.io/introduction

2. MCP Server in Python — Everything I Wish I'd Known on Day One | DigitalOcean, 4月 12, 2025にアクセス、https://www.digitalocean.com/community/tutorials/mcp-server-python
3. Model context protocol (MCP) - OpenAI Agents SDK, 4月 12, 2025にアクセス、https://openai.github.io/openai-agents-python/mcp/
4. Model Context Protocol (MCP) an overview - Philschmid, 4月 12, 2025にアクセス、https://www.philschmid.de/mcp-introduction
5. Specification - Model Context Protocol, 4月 12, 2025にアクセス、https://modelcontextprotocol.io/specification/2025-03-26
6. Building AI Agents with Model Context Protocol: From Specification to Implementation, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=oSGVQIZxi7s
7. Model Context Protocol · GitHub, 4月 12, 2025にアクセス、https://github.com/modelcontextprotocol
8. Example Servers - Model Context Protocol, 4月 12, 2025にアクセス、https://modelcontextprotocol.io/examples
9. Example Clients - Model Context Protocol, 4月 12, 2025にアクセス、https://modelcontextprotocol.io/clients
10. google/adk-python: An open-source, code-first Python ... - GitHub, 4月 12, 2025にアクセス、https://github.com/google/adk-python
11. Agent Development Kit: Making it easy to build multi-agent applications, 4月 12, 2025にアクセス、https://developers.googleblog.com/en/agent-development-kit-easy-to-build-multi-agent-applications/
12. google/adk-docs: An open-source, code-first Python toolkit for building, evaluating, and deploying sophisticated AI agents with flexibility and control. - GitHub, 4月 12, 2025にアクセス、https://github.com/google/adk-docs
13. Agent Development Kit - Google, 4月 12, 2025にアクセス、https://google.github.io/adk-docs/
14. Just did a deep dive into Google's Agent Development Kit (ADK). Here are some thoughts, nitpicks, and things I loved (unbiased) - Reddit, 4月 12, 2025にアクセス、https://www.reddit.com/r/LocalLLaMA/comments/1jvsvzj/just_did_a_deep_dive_into_googles_agent/
15. Just did a deep dive into Google's Agent Development Kit (ADK). Here are some thoughts, nitpicks, and things I loved (unbiased) - Reddit, 4月 12, 2025にアクセス、https://www.reddit.com/r/AI_Agents/comments/1jvsu4l/just_did_a_deep_dive_into_googles_agent/
16. Google CAN'T STOP COOKING: FREE AI AGENTS SDK JUST DROPPED - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=tIdS6e5rD8A
17. Build Your First Intelligent Agent Team: A Progressive Weather Bot with ADK - Google, 4月 12, 2025にアクセス、https://google.github.io/adk-docs/get-started/tutorial/
18. MCP tools - Agent Development Kit - Google, 4月 12, 2025にアクセス、https://google.github.io/adk-docs/tools/mcp-tools/
19. Build and manage multi-system agents with Vertex AI | Google Cloud Blog, 4月 12,

2025にアクセス、 https://cloud.google.com/blog/products/ai-machine-learning/build-and-manage-multi-system-agents-with-vertex-ai
20. pyproject.toml - google/adk-python - GitHub, 4月 12, 2025にアクセス、 https://github.com/google/adk-python/blob/main/pyproject.toml
21. The MCP Integration EVERYONE is Sleeping On (MCP + Custom AI Agents) - YouTube, 4月 12, 2025にアクセス、 https://www.youtube.com/watch?v=soC4n-nKWF8
22. The Model Context Protocol (MCP) Explained (and one cool code example.) - YouTube, 4月 12, 2025にアクセス、 https://www.youtube.com/watch?v=5ZWeCKY5WZE
23. Thinking if MCP server could work with anki connect? thx - Add-ons, 4月 12, 2025にアクセス、 https://forums.ankiweb.net/t/thinking-if-mcp-server-could-work-with-anki-connect-thx/57436
24. Quickstart: Build an agent with the Agent Development Kit | Generative AI on Vertex AI, 4月 12, 2025にアクセス、 https://cloud.google.com/vertex-ai/generative-ai/docs/agent-development-kit/quickstart
25. Build Multi-Model Agent in 10 Minutes - Google's New Agent Kit Is INSANE! - YouTube, 4月 12, 2025にアクセス、 https://www.youtube.com/watch?v=SjZG-QKrw5o
26. Google Agent2Agent Protocol (A2A) - Complements Anthropic's MCP - YouTube, 4月 12, 2025にアクセス、https://www.youtube.com/watch?v=JllSpEG8_VQ