ChatGPT

AIエージェントにおけるモデル・コンテキスト・プロトコル（MCP）調査レポート

MCPの概要（設計目的と解決する課題）

モデル・コンテキスト・プロトコル（**MCP**）は、**AIアプリケーションと外部システムを接続するためのオープン標準プロトコル**です [1]。MCPを使用すると、ChatGPTやClaudeのようなAIモデルが**ファイルシステム、データベース、検索エンジン、電卓**などのツールやデータソースに統一的な方法でアクセスできます [1]。例えるならば、**MCPはAIアプリにおける「USB-Cポート」**のような役割を果たします [2]。USB-Cが様々な機器を標準化されたポートで接続できるように、MCPもAIモデルを様々なデータ源やツールに標準インターフェースで接続します [2]。

このプロトコルが生まれた背景には、**AIモデルが必要な情報に直接アクセスできないという課題**があります。高度なLLM（大規模言語モデル）でも、組織内のデータや最新情報には容易にアクセスできず**「データの孤島」に閉じ込められている**状況でした [3]。新しいデータソースごとに個別のコネクタ実装が必要となり、統合やスケールが困難になるという問題がありました [3]。MCPはこの問題に対処するために設計され、**断片化したカスタム統合を排し、一つの共通プロトコルでAIとデータソースをつなぐ**ことを目指しています [4]。これによりAIアシスタントは最新かつ関連性の高い情報にアクセスし、より適切な応答を生成できるようになります [5] [6]。また**安全で双方向**の接続を標榜しており、AIからツールを呼び出すだけでなく、必要に応じてツール側からAI（ユーザ）に確認を求めるような対話も可能です [6]。

要約すると、MCPは**AIエージェントがコンテキスト（背景情報）を獲得するための標準化された経路**を提供します。MCP自体は**意思決定やプランニングを行うわけではなく** [7]、あくまでモデルに必要なデータやツールへのアクセスを「用意する」役割です。例えば医療AIがチャットの前に患者の電子カルテを取得する場合、MCPは**「対話前にこの患者の情報を取得しておいて」とモデルに共有する**イメージです [6]。こうした**適切な文脈情報の提供**によって、モデルは質問に対しより正確で具体的な回答ができるようになります [6]。開発者にとっても、各種サービス毎のAPI接続やデータフォーマット処理を都度実装する必要がなくなり、**開発負荷と連携の複雑さを削減**できます [8]。

MCPの技術仕様（構造・通信方式・データフォーマット）

アーキテクチャ:　MCPは**クライアント-サーバー型アーキテクチャ**を採用しています。具体的には、AIアプリケーション側が**「MCPホスト」**として振る舞い、外部システムのデータや機能を提供する**「MCPサーバー」**へ接続します [9]。ホスト上では接続先のサーバーごとに**MCPクライアント**をインスタンス化し、各サーバーとの専用通信チャネルを維持します [10]。例えば**Visual Studio Code**はMCPホストとなり、社内エラー追跡サービス用のMCPサーバー（例：Sentry MCPサーバー）やローカルファイルシステム用のMCPサーバーにそれぞれクライアント経由で接続します [11]。このように、一つのAIアプリ（ホスト）が複数のMCPサーバーから並行してコンテキストを取得できる柔軟な構造になっています。

レイヤード設計: MCPは**データ層**と**トランスポート層**の二層から構成されています [12]。**データ層**では**JSON-RPC 2.0**に基づくメッセージ形式と対話の意味論が定義されています [13]。ここには**接続の初期化・終了（ライフサイクル管理）**、**コア機能（後述のツールやリソースなどのプリミティブ）**、**クライアント側機能（モデルへの問い合わせ要求など）**、**通知やエラーハンドリング**といった通信パターンが含まれます [14] [15]。一

方、**トランスポート層**はクライアントとサーバー間の低レベル通信路や認証方式を扱います [16]。MCPは環境に応じて**2種類のトランスポート**をサポートしています [17]:

- **標準入出力（STDIO）トランスポート:** ローカル環境でのプロセス間通信に用いられ、モデル（クライアント）とサーバーが同一マシン上で標準入出力ストリームを介してメッセージを交換します [17]。ネットワークのオーバーヘッドがなく**高速・軽量**な同期通信が可能で、ローカルファイルシステムやデータベース等の連携に適しています [18] [19]。
- **HTTP（ストリーミング対応）トランスポート:** ネットワーク越しのリモート通信向けに、**HTTP POST**を使ったリクエスト送信と、**サーバー送信イベント（SSE）**によるリアルタイム応答ストリーミングを組み合わせます [17] [20]。標準的なHTTP認証（ベアラートークンやAPIキー、OAuthによるトークン取得など）を利用でき、マルチクライアント・非同期通信にも適します [21]。SSEにより、**長時間処理の進捗通知**や**部分的なストリーム結果の逐次受信**、リクエストキャンセル等も可能です [22] [23]。

両トランスポート方式においてメッセージ内容そのものは共通で、**JSON-RPC　2.0形式のリクエスト/レスポンス/通知**がやりとりされます [17] [24]。JSON-RPCに則った**「method（メソッド名）」「params（引数）」「id（識別子）」**等のフィールドを持つ**構造化JSONメッセージ**により、クライアント・サーバー間の呼び出し要求や結果返却、エラー、通知が表現されます [25] [24]。メッセージの基本タイプは**「リクエスト」**（要求とID付き）、**「レスポンス（結果）」**（対応するリクエストIDに対する正常応答）、**「エラー」**（失敗時のコードとメッセージ）、**「通知」**（片方向の通達で応答不要）の4種です [26] [27]。MCPでは**双方向通信**が可能であり、通常はクライアントからサーバーへのリクエストが多いものの、サーバー側がクライアントへ通知や特定メソッド呼び出し（後述の**サンプリング要求**など）を行うこともできます [28]。

プリミティブ（基本機能要素）: MCPのデータ層で定義される重要な概念が**「プリミティブ」**です [29]。プリミティブとは、**クライアントとサーバーが互いに提供し合う機能の分類**であり、これによってAIエージェントが共有できるコンテキスト情報や実行可能なアクションの種類が決まります [14]。特にMCPサーバーは以下の**3種類のコア・プリミティブ**を公開できます [30]:

- **ツール (Tools):** AIアプリケーション側から**呼び出せる関数や操作**です（例: ファイル操作、APIコール、データベースクエリなど） [30]。サーバーは `tools/list` によって提供可能なアクション一覧を公開し、クライアントはそこから選んで `tools/call` で実行を依頼します [31]。
- **リソース (Resources): コンテキストデータを提供する情報源**です（例: テキストファイルの内容、DBのレコード、APIレスポンスなど） [32]。クライアントは `resources/list` で利用可能なデータ項目を確認し、`resources/read` 等で具体的なデータ取得を行います [31]。
- **プロンプト (Prompts): LLMとの対話に用いるテンプレート**やサンプルです（例: システムプロンプト、Few-shot例） [33]。`prompts/list` や `prompts/get` によりテンプレート内容を取得し、これをモデルへの指示として活用できます [31]。

各プリミティブにはこのように**発見（list）・取得（get/read）・実行（call）**のメソッド群が定義されており、クライアントはまずサーバーに問い合わせて利用可能な機能を**動的に発見**し、必要に応じて実行します [31]。例えば、あるMCPサーバーが企業内データベースに関するコンテキストを提供する場合、**「データベース検索ツール」**（tools）、**「データベーススキーマ情報」**（resources）、**「DBクエリの例示プロンプト」**（prompts）といった複数のプリミティブをまとめて公開できます [34]。

加えて、MCPでは**クライアント側**（AIアプリ側）がサーバーに提供するプリミティブも定義されています [28]。これにより、サーバー実装者がよりリッチな対話を構築可能です。主なクライアント・プリミティブには次のものがあります [28]:

- **サンプリング (Sampling):** サーバーが**ホスト側のLLMから文章生成**をリクエストする仕組みです [28]。サーバー側で独自にLLMを組み込まなくても、`sampling/complete` メソッドでホストの言語モデルに補完（completion）を依頼できます。

- **引き出し (Elicitation):** サーバーが**ユーザーから追加情報を引き出す**ためにクライアントに問い合わせる仕組みです [35] 。例えば「本当に削除してよいか？」と確認する場合に `elicitation/request` を使ってユーザー承認を求める、といった対話が可能です。
- **ログ送信 (Logging):** サーバーがクライアントに**ログメッセージ**（デバッグ情報など）を通知できる機能です [36] 。

このようなプリミティブ設計により、**モデル側（クライアント）とツール側（サーバー）の役割分担**が明確化されています。MCPクライアントはサーバーから提供されるツール類を使ってタスクを実行し、一方サーバーはクライアント（モデル）に必要に応じ**追加の質問**を投げたり**モデルの力を借りて処理**したりできます [28] 。さらに**通知 (Notification)**機能もあり、サーバー側でツールリストが変化した場合などに `tools/list_changed` 等の通知を送信してクライアントにリアルタイム反映させることが可能です [15] 。これらにより、動的なツール更新や長時間ジョブの進捗通知など柔軟な相互通信が実現されています。

通信の流れ: MCPにおけるクライアント-サーバーの接続開始時には**初期化ハンドシェイク**が行われ、**プロトコルのバージョン交渉**（互いに対応するプロトコル日付版の確認）や**機能キャパビリティの宣言**（お互いにサポートするプリミティブや通知対応可否などの表明）、**識別情報交換**（クライアント名・バージョンやサーバー名などの送信）が行われます [37] [38] 。これにより異なるバージョン間の誤通信を防ぎ、使えない機能を呼び出すことも避けられます [38] 。初期化後、クライアントはサーバーに対し**利用可能なツールやリソースの一覧取得**を行い（ `*/list` ）、必要に応じて**ツール実行**や**リソース読込**リクエストを送ります [31] 。サーバーは要求を処理して**結果（またはエラー）**を返し、必要ならば逐次データをSSEでストリーム配信します [20] 。会話中にサーバーが新たなツールを登録したり削除した場合は、上記の**通知**でクライアントに変化を伝達し [15] 、クライアントは再度リスト取得して認識を更新します。終了時はいずれかが **`shutdown`** やコネクションクローズ操作を行い、セッションを安全に切断します [39] 。

MCPに対応する環境・フレームワーク

MCPは**2024年末のAnthropic社による提唱・オープンソース化**以降、急速にエコシステムを拡大しています [40] 。現在、複数のAIエージェント向けフレームワークや主要クラウドサービスがMCPを採用・サポートし始めています。

- **LangChain:** LLMエージェント構築のデファクト標準ライブラリであるLangChainは、2025年に**MCPアダプターの公式サポート**を追加しました [41] 。MCPアダプターはLangChainのエージェントとMCPサーバー上のツール群を橋渡しする**ミドルウェア**であり、開発者は**わずかなコード変更で既存のMCPツールをエージェントに組み込めます** [42] 。これにより**LangChainのメモリ管理やプランニング能力**と、MCPの標準化ツール接続とを組み合わせた高度なエージェント構築が容易になりました [42] [43] 。LangChain MCPアダプターの利点として、**既存エコシステムの活用による迅速なプロトタイピング**、エージェントロジックとツール実装の**関心の分離**、ツール管理の一元化（バージョン管理・監査）などが挙げられています [44] [45] 。現に複数の企業がLangChain MCPアダプターを用いて**生産環境でMCP対応エージェントを稼働**させ始めています [46] 。

- **OpenAgents:** OpenAgentsは近年登場した**オープンソースのエージェント実行プラットフォーム**で、コーディング不要で使えるUIやAPIを備えた**即戦力エージェント集**です [47] [48] 。OpenAgentsは**設計当初からMCPファースト**を掲げ、コアアーキテクチャにMCPのオープン標準を取り入れています [47] 。具体的にはOpenAgentsの組み込みエージェントは**MCPクライアント**として外部のMCPサーバーが提供するツール群を利用でき、逆にOpenAgents自体も**MCPサーバー**としてファイル閲覧・Webスクレイピング等の機能を公開することができます [49] [50] 。例えばOpenAgentsには、コードベースを操作する**「コーダーエージェント」**やデータ分析を行う**「データエージェント」**などがあり、いずれも背後でMCPツールを活用してファイルシステムやGitリポジトリ、データ可視化サービス等にアクセスします [51] 。OpenAgentsはUIを通じた対話型の利用を想定しており、複数エージェントを**ワークフローで連携**させたり、**権限管理付きで社内展開**する用途に適しています [52] [53] 。MCPと

の組み合わせにより、一度MCPで登録したツールを**複数のエージェント間で共有再利用**することや、ユーザ操作のログ・監査を統合管理することも容易になっています [54] [55]。

- **OpenAI Agents（エージェントモード）：** OpenAIは自社のAPIに**「関数呼び出し（Function Calling）」**や**プラグイン**機能を通じてLLMからツールを使う仕組みを提供してきましたが、2025年には**公式のエージェントSDK**（OpenAI Agents SDK）で**MCPをネイティブサポート**しました [56] [57]。OpenAI Agents SDKでは、**OpenAIプラットフォームの関数**と**外部MCPツール**を**同列に登録**でき、エージェントがそれらを組み合わせて利用することが可能です [57]。OpenAIのデベロッパードキュメントにもMCP統合の手順が記載されており、例えば外部のGit操作用MCPサーバーをOpenAI Agentにツールとして渡し、モデル（GPT-4など）がそれを**自律的に呼び出す**ような実装が可能です [58] [59]。OpenAIは自社ホスティングの「Hosted MCPサーバーツール」も提供し、OpenAIのクラウド上でMCPツールを動かしてエージェントに組み込む仕組みも整備しています [59] [60]。これによりOpenAIのAgent機能利用者も、よりオープンなMCPエコシステムにシームレスにアクセスできるようになりました。

- **Microsoft Autogen:** マイクロソフトのAutogenは**複数エージェント対話**や**ツール利用**を容易にするフレームワークですが、こちらも**MCP統合を開始**しています [61]。Autogenはエージェント同士が協調してタスクをこなすシナリオを重視しており、MCP対応によって各エージェントが**標準化ツールセット**を共有できるようになります [61]。さらにAutogen自体が**MCPサーバー実装**として機能し、提供する機能を他エージェントから利用させることも検討されています [61]。マイクロソフトはこの分野でGoogleなどとも共に**標準策定**を進めており、後述のAgent-to-Agent (A2A)プロトコルなど複数標準を包括的にサポートする方針です [62] [63]。

- **主要クラウドAIプラットフォーム:** AWSおよびGoogle、IBM、Anthropicなども**自社サービスへのMCP対応を表明・実装**しています。AWSは自社のエージェント開発SDK「**Strands Agents**」でMCPをサポートし、Amazon Bedrock経由のモデルにもMCPツールを接続可能にしています [64] [65]。GoogleもVertex AIやDuet AI（クラウドAIツール群）において**MCP準拠のツール登録**を実験的に進めており、Azure OpenAIサービス/StudioでもMCPを使ったツール連携の簡素化が検討されています [66]。またAnthropicのClaudeシリーズでは**Claude Desktop**アプリでローカルMCPサーバーへの接続をサポートし、社内データへの安全なアクセスを可能にしています [67] [68]。さらにAnthropicはClaude 3.5 Sonnetモデルを用いた**MCPサーバー実装の自動構築**もアピールしており、組織が自前のデータセットをMCPサーバー化する支援も行っています [69]。IBMも自社の**watsonx Orchestrate**などエンタープライズAIソリューションにおいてMCPを活用し、企業内の各種ツール/APIとの連携基盤として位置付けています [70] [71]。

このように、**MCPは事実上の業界標準**として広まりつつあり、**LangChainのようなライブラリによる自作エージェント開発**から、**OpenAgentsのような完成品エージェントの利用**、さらに**主要クラウドのAI統合機能**まで、幅広い環境で採用が進んでいます [72] [73]。早期からAnthropicのClaudeやBlock社、Apollo社といった企業が実運用に組み込んでおり [74]、現在ではZed（エディタ）、Replit、Codeium、Sourcegraph等の開発者向けツールにもMCPが組み込まれて**AIコーディング支援**に活用されています [75]。こうしたエコシステムの広がりにより、「一度MCPに対応したツールは他の多くのAIエージェントでも使える」という**再利用性と相互運用性の高さ**が実現しつつあります [57] [73]。

MCPの利点と設計上の工夫

MCPを採用することにより、AIエージェント開発や運用には次のような**利点**がもたらされます。

- **開発効率とスケーラビリティ向上:** MCPは**ツール統合の複雑さを大幅に低減**します [8]。開発者は各サービスごとに個別のAPI呼び出しロジックやデータ変換コードを書く必要がなく、MCPサーバーさえ用意すれば**統一された手順でツール/データを接続可能**です [8]。結果として、新たなデータソース

を追加する際の工数が減り、AIシステム全体の**拡張性**が高まります [4] [76]。現場では**「コードの再発明」**を防ぎ、チーム間でも標準化されたやり方でツールを共有できるようになります [77]。

- **モデルの能力強化と最新情報の反映:** MCPによりモデルは**リアルタイムの外部知識**を容易に得られるため、応答の正確性・有用性が向上します [6]。例えば従来、LLM単体では訓練データ以降の新情報は知らないため限界がありましたが、MCP経由で最新の社内ドキュメントやニュースデータベースにアクセスすることで、**常に最新状況を踏まえた回答**が可能です [6]。また複数のデータソースから**統合的なコンテキスト**を与えることで、モデルは断片的知識に頼ることなく包括的な推論や計画ができます [7]。エンドユーザーから見れば、MCP対応のAIアプリは**自分のデータや使用ツールに直接働きかけてくれる頼もしいアシスタント**となるでしょう [78]。

- **オープン標準による相互運用性:** MCPはベンダーロックインのないオープンな仕様のため、**異なるプラットフォーム間でツールやエージェントの互換性**があります [79]。例えば社内で開発したMCPツール群を、LangChainベースのエージェントとOpenAIのエージェント双方から利用するといったことが容易です [80]。これにより「特定のフレームワークでしか動かないツール」問題が解消し、**エコシステム全体で再利用可能なコンポーネント**を作る文化が促進されます [80]。AWSやMicrosoft、Googleなど主要企業がMCP標準の委員会に参画し、互換実装を進めているのも相互運用性の観点から大きなプラスです [62] [66]。

- **保守性・ガバナンスの向上:** MCPでは**スキーマ定義**に基づく厳格なメッセージ交換と**バージョン管理、認証・認可**が組み込まれており、システム変更時の影響を抑制できます [21] [37]。ツールにはバージョンやタグ（dev/prod等）を付けて管理でき、変更時には通知機能でエージェント側に即時反映されます [81] [82]。またOAuth等による**統一的な認証**でセキュアな接続を確保しつつ、MCPクライアントごとに**利用ツールのフィルタリングやレート制限**を適用する仕組みも整えられます [83] [84]。ログも各ツール呼び出し単位で構造化記録（タイムスタンプやセッションID付き）できるため、**監査やデバッグ**もしやすくなります [85] [45]。このようにエンタープライズシステムで必要とされる**ガバナンスと可観測性**を考慮した設計上の工夫が随所に施されています。

- **高度な拡張性（マルチエージェント/長時間タスク対応など）:** MCPは元々ツール統合用に設計されたプロトコルですが、その**汎用的なメッセージ機構**により**エージェント同士の連携**にも応用が可能です [86]。実際、MCPコミュニティでは**複数のAIエージェントがMCP経由で互いに通信・協調**するための拡張が進められており、AWSや有志によって**エージェント間対話向けの追加機能**が議論・実装されています [86] [87]。また**タスクプリミティブ**（実験的機能）によって、長時間かかる処理をバックグラウンド実行して途中経過や完了を通知する仕組みも提案されています [88]。これらにより、単一エージェントがツールを使うだけでなく、**複数のエージェントが役割分担しつつ共同作業を進める**ような高度なシナリオにもMCPがベースとして貢献できる柔軟性があります [89] [90]。

以上のように、MCPは**AIエージェントの能力向上と開発・運用の効率化を両立する設計**となっています。その根底には「AIエージェントが現実世界の文脈を容易に取り込み、環境変化に適応し続けるにはオープンな標準基盤が不可欠である」との認識があります [91] [92]。MCPは今後もコミュニティ主導で改良が重ねられ、ツール統合にとどまらずエージェント協調や観測性の標準化といった新たな領域にも発展していくと期待されています [93] [94]。

類似プロトコルとの比較

MCPと類似する目的を持つ**他のプロトコルや手法**として、以下のようなものが存在します。

- **Agent-to-Agent プロトコル（A2A）:** 2025年にGoogleが提唱した**Agent2Agent（A2A）**は、**複数エージェント間の直接通信**に焦点を当てた新しいオープンプロトコルです [62]。MCPがエージェントと外部ツール/データの連携標準なのに対し、A2Aは**エージェント同士がメッセージ交換し協調動作する**た

めの標準です [62]。AWSをはじめ業界はMCPとA2Aの**両方を重要なオープン標準**と位置付け、どちらにもコミットする姿勢を示しています [63]。今後、MCPで培った機能（ストリーミング通信や機能ネゴシエーション等）を活かしつつA2Aとの**相互補完**が図られる見通しです。要するに、**MCP＝エージェント対ツールの通信標準、A2A＝エージェント対エージェント通信標準**という関係であり、どちらもオープンな相互運用性を目指す点で共通しています。

- **OpenAIの関数呼び出し・プラグイン機構:** OpenAIはChatGPTプラグインなど**モデルに外部機能を付与する独自仕組み**を提供してきました。これはモデル側から見れば**関数仕様（JSONス chem a）を与えて出力を制約**したり、HTTP経由で外部APIを呼ぶものです。しかし**OpenAIの関数呼び出しは特定モデルに依存**し、一度の応答内で単発的にAPIを呼ぶ設計で、汎用的なセッション管理やマルチステップ計画は想定されていません [95]。これに対しMCPは**モデル非依存のプロトコル層**であり、セッションの**継続的な状態管理**（Context維持）や**エージェントのメモリ共有**、ツールスキーマの厳密検証など**包括的で拡張性の高い**設計になっています [95] [96]。関数呼び出しは手軽な一方でスコープが限定的ですが、MCPは**複数ツールの組み合わせや長時間対話、複数エージェント介在**といった高度なシナリオに耐え得る柔軟さがあります [95] [96]。加えてMCPはどのLLMにも実装可能であり、例えばAnthropicモデルとOpenAIモデルの双方が同じMCPサーバー群を使うこともできます（関数呼び出しはモデルごとに専用設計です）。以上から、**「簡易だがクローズド」な関数呼び出し**と**「高度だがオープン」なMCP**という対比ができます。

- **エージェント構築フレームワーク（LangChain等）による独自統合:** MCP登場以前、**LangChain**や**AutoGPT**のようなフレームワークでは各ツール接続が個別に組み込まれていました。例えばLangChainでは**Python関数ツール**や**検索APIラッパー**を内部クラスとして用意し、エージェント実行中にこれらを呼び出すアーキテクチャでした。しかしこのやり方は**フレームワーク間でツール使い回しができない**、新ツール追加時に**コード変更が必要**、運用時に**権限管理やログ収集を各所で実装**する必要があるなどの課題がありました [97]。MCPはそうした問題を解決する**標準レイヤー**として、LangChain等の上位フレームワークから下位のツール実装を疎結合に切り離しました [42]。実際LangChainは前述の通りMCPアダプターを提供し、AutoGPTやBabyAGIのような自律エージェントもMCPを使う実験が始まっています（IBMによるチュートリアルなどが公開されています）。総じて、**従来のカスタム統合アプローチ**と比べ、MCPは**再利用性・保守性・標準化**の面で優れていると言えます。

- **その他の関連標準:** エージェント分野では他にも、**OpenTelemetry**によるエージェント動作の観測標準化（OTELのエージェント対応）や、複数LLM間のやりとりプロトコルなど検討が進んでいます [65]。これらは直接MCPと競合するものではありませんが、将来的にMCPと連携して**より包括的なオープン標準スタック**を形成する可能性があります。例えばMCPでツールやエージェントを接続しつつ、OpenTelemetryでそのやりとりを計測・可視化する、といった構成です。また各種ベンダー独自実装（例: IBM Watsonx OrchestrateやAzure Logic AppsのLLM連携機構など）もありますが、潮流としては**MCP等の標準を取り入れて相互運用性を確保**する方向に向かっています [66]。長期的には、これら標準プロトコル同士の比較よりも**補完関係**が強まっていくと考えられます。

以上、MCPの概要から技術仕様、実際の利用環境、利点、そして他の取り組みとの比較について概観しました。MCPはまだ発展途上の規格ではありますが、**「AIエージェントのための共通インターフェース」**という着想は業界横断的な支持を集めており [72]、今後ますますエージェント技術の基盤として重要度を増すでしょう。各種フレームワークやクラウドサービスでMCPが利用可能になりつつある現在、**標準化されたコンテキスト共有**がAI活用の幅を広げ、生産性やユーザ体験の向上に寄与することが期待されています [79]。今まさに、AIエージェントを扱う開発者にとって**MCPを理解し活用する価値は非常に大きい**と言えるでしょう。

参考文献・出典: 本レポートの記載内容はAnthropic社の発表資料 [4] [67]、Model Context Protocol公式ドキュメント [1] [14]、Medium等の技術解説記事 [2] [6]、IBMおよびAWSによる分析記事 [20] [76] など信頼できる情報源に基づいています。各所に挙げた【△†Lxx-Lyy】のリンクより詳細情報をご確認いただけます。

1 8 78 What is the Model Context Protocol (MCP)? - Model Context Protocol
https://modelcontextprotocol.io/docs/getting-started/intro

2 6 7 AI Agents vs. Model Context Protocol (MCP): Choosing the Best Approach | by YUSUFF ADENIYI GIWA | Medium
https://medium.com/@adeniyi221/ai-agents-vs-model-context-protocol-mcp-choosing-the-best-approach-a72723446eba

3 4 5 40 67 68 69 74 75 Introducing the Model Context Protocol \ Anthropic
https://www.anthropic.com/news/model-context-protocol

9 10 11 12 13 14 15 16 17 21 28 29 30 31 32 33 34 35 36 37 38 39 88 Architecture overview - Model Context Protocol
https://modelcontextprotocol.io/docs/learn/architecture

18 19 20 24 25 70 71 76 91 92 93 97 What is Model Context Protocol (MCP)? | IBM
https://www.ibm.com/think/topics/model-context-protocol

22 23 62 63 64 65 86 87 89 90 94 Open Protocols for Agent Interoperability Part 1: Inter-Agent Communication on MCP | AWS Open Source Blog
https://aws.amazon.com/blogs/opensource/open-protocols-for-agent-interoperability-part-1-inter-agent-communication-on-mcp/

26 27 41 LangChain MCP Adapter: A step-by-step guide to build MCP Agents - Composio
https://composio.dev/blog/langchain-mcp-adapter-a-step-by-step-guide-to-build-mcp-agents

42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 61 66 72 73 77 79 80 81 82 83 84 85 95 96 Integrating MCP with Popular Frameworks: LangChain & OpenAgents
https://www.getknit.dev/blog/integrating-mcp-with-popular-frameworks-langchain-openagents

58 59 60 Model Context Protocol (MCP) | OpenAI Agents SDK
https://openai.github.io/openai-agents-js/guides/mcp/