# 5. 知財業務の自動化への道筋

# 5. 自動化への道筋 - 概要

## この章で学ぶこと

**主なトピック:**

- Claude Code、Cursor、Devin 等コーディングアシスタントの活用
- クラウドベンダー別 AI エージェント比較と Vellum AI LLM リーダーボード活用
- MCP（Model Context Protocol）と A2A（Agent-to-Agent）プロトコル

# 5-1. コーディングアシスタントの活用

## Claude Code

- **コード生成**: 特許分析ツールの作成・カスタムスクリプトの開発
- **デバッグ支援**: エラーの自動修正・パフォーマンス最適化
- **最適化**: アルゴリズム改善・メモリ使用量の最適化
- **ドキュメント生成**: コードの自動ドキュメント化・API 仕様書作成

## Cursor

- **IDE 統合**: 開発環境での直接支援・リアルタイムコード補完
- **リアルタイム**: コード作成の即座の支援・エラー予測
- **学習機能**: プロジェクト固有の学習・コードスタイルの適応
- **リファクタリング**: 自動的なコード改善・アーキテクチャ最適化

## Devin

- **自律開発**: 完全自動のソフトウェア開発・要件から実装まで
- **要件理解**: 自然言語からの仕様理解・技術選択の自動化
- **継続改善**: フィードバックによる改善・継続的デプロイメント
- **プロジェクト管理**: タスク分割・進捗管理・品質保証

## 知財業務での活用例

Claude Code を使用した特許分析ツールの自動生成例：

**参照ファイル**: `code-examples/claude-code-patent-analysis-tool.py`

この例では、Claude Code のプロンプトエンジニアリング機能を活用して、特許文献の自動分析ツールを段階的に開発するプロセスを示しています。プロンプトには以下の要素が含まれています：

- **Role**: 専門家としての役割定義
- **Context**: 開発の背景と要件
- **Task**: 具体的な開発手順
- **Output Format**: 出力形式の指定
- **Constraints**: 制約条件の明示

## 5-2. Gemini CLI について

### 特徴

- **コマンドライン統合**: ターミナルからの直接利用・シェルスクリプト統合
- **スクリプト化**: 自動化の容易な実装・バッチ処理の支援
- **ファイル操作**: ローカルファイルの直接処理・一括変換
- **API 連携**: 外部サービスとの統合・データ取得・処理

## インストールと設定

### インストール手順

```
# Claude Code CLI のインストール
npm install -g @google/gemini-cli
```

実際の画面はこちら（Cursor 利用の場合）

※以下のコマンド（process,generate-report など）は、MCP 拡張で実現します。

## 活用例

```
# 特許文献の一括処理
gemini process patents/*.pdf --output analysis/ --format json

# 定期レポートの自動生成
gemini generate-report --template weekly --data patents.json --output reports/

# 技術動向の監視
gemini monitor-trends --keywords "AI,patent" --interval daily --output trends/

# 文書の自動翻訳
gemini translate --input japanese_patents/ --output english_patents/ --source ja --target en
```

## 知財業務での具体的活用

### 特許文献の一括分析

```
# 複数の特許文献を一括で分析
gemini analyze-patents \
  --input-dir ./patent_documents/ \
  --output-dir ./analysis_results/ \
  --analysis-type "technical_trends,competitor_analysis,classification" \
  --format "json,csv,pdf"
```

### 技術動向の自動監視

```
# 特定技術分野の動向を自動監視
gemini setup-monitoring \
  --technology "artificial intelligence" \
  --keywords "machine learning,deep learning,neural networks" \
  --frequency "daily" \
  --notification "email,slack" \
  --output "./monitoring_results/"
```

# 5-3. Claude Code CLI の活用

## Claude Code CLI の基本概念

### Claude Code CLI とは

- **コマンドライン統合**: ターミナルからの直接利用
- **コード生成**: 自然言語からのコード自動生成
- **プロジェクト管理**: 既存プロジェクトの理解と改善
- **デバッグ支援**: エラーの自動修正と最適化

**主要機能**

- **ファイル操作**: 既存ファイルの読み込み・分析・修正
- **コード生成**: 新規ファイルの作成・機能追加
- **プロジェクト分析**: プロジェクト全体の構造理解
- **自動テスト**: テストコードの生成・実行

## インストールと設定

### インストール手順

```
# Claude Code CLI のインストール
npm install -g @anthropic-ai/claude-code-cli

# または
pip install claude-code-cli

# 認証設定
claude-code auth
```

実際の画面はこちら（Cursor 利用の場合）

## 基本設定

```
# 設定ファイルの作成
claude-code init

# プロジェクトの設定
claude-code config --project-path ./my-project
claude-code config --language python
claude-code config --framework flask
```

## 知財業務での活用例

### 特許分析スクリプトの生成

```
# 特許分析スクリプトの生成（Claude API使用例）
claude --prompt "特許文献を分析するPythonスクリプトを作成してください。\
PDFファイルを読み込み、テキストを抽出し、キーワード分析を行い、
結果をCSVファイルに出力する機能が必要です。" > patent_analyzer.py

# 生成されたスクリプトの実行
python patent_analyzer.py --input patents/ --output results/
```

## 既存コードの改善

```
# 既存の特許検索スクリプトを改善（Claude API使用例）
claude --prompt "以下のファイルのエラーハンドリングを追加し、
ログ機能を実装し、パフォーマンスを最適化してください。
" --file patent_search.py > improved_patent_search.py

# 特定の関数を改善
claude --prompt "search_patents関数の検索精度を向上させ、
結果の並び替え機能を追加してください。\
" --file patent_search.py > optimized_patent_search.py
```

## 活用パターン

### プロジェクト全体の分析

```
# プロジェクト全体の構造分析（treeコマンド使用）
tree ./patent-analysis-system > project_structure.txt

# 改善提案の生成（Claude API使用例）
claude --prompt "以下のプロジェクト構造を分析し、パフォーマンス、
セキュリティ、保守性の観点から改善提案をしてください。\
" --file project_structure.txt > analysis_report.md
```

## 自動テストの生成

```
# 既存コードに対するテストの生成（Claude API使用例）
claude --prompt "patent_analyzer.pyのテストコードを
pytestフレームワークで作成してください。\
 --file patent_analyzer.py > test_patent_analyzer.py

# テストの実行
pytest test_patent_analyzer.py
```

## 高度な活用例

### 特許監視システムの構築

```
# 特許監視システムの生成（Claude API使用例）
claude --prompt "特許監視システムを作成してください。以下の機能が必要です：
1. 定期的な特許検索（毎日午前2時）
2. 新規特許の自動検出
3. 重要度評価
4. メール通知機能
5. データベース保存
6. Web ダッシュボード" > patent_monitor_system.py

# システムの実行
python patent_monitor_system.py
```

## API クライアントの生成

```
# 特許庁API クライアントの生成（Claude API使用例）
claude --prompt "特許庁API を使用するPython クライアントを作成してください。
- 特許検索機能
- 特許詳細取得機能
- エラーハンドリング
- レート制限対応
- キャッシュ機能" > patent_api_client.py
```

## 設定とカスタマイズ

### プロジェクト固有の設定

```
# プロジェクト設定ファイルの作成（手動作成例）
cat > pyproject.toml << EOF
[build-system]
requires = ["setuptools", "wheel"]
build-backend = "setuptools.build_meta"

[project]
name = "patent-analysis"
version = "0.1.0"
dependencies = ["fastapi", "langchain", "psycopg2-binary", "pytest"]
EOF

# カスタムテンプレートの設定（Claude API使用例）
claude --prompt "特許分析用のクラステンプレートを作成してください。" > template.py
```

## ワークフローの自動化

```
# 自動化ワークフローの設定（cron使用例）
cat > daily_patent_analysis.sh << EOF
#!/bin/bash
cd /path/to/patent-analysis
python search_patents.py
python analyze_results.py
python generate_report.py
python send_notification.py
EOF

chmod +x daily_patent_analysis.sh

# crontabに追加（毎日午前2時実行）
echo "0 2 * * * /path/to/daily_patent_analysis.sh" | crontab -
```

## 効果測定と改善

### コード品質の測定

```
# コード品質の分析（pylint使用例）
pylint patent_analyzer.py --output-format=json > quality_report.json

# 改善提案の生成（Claude API使用例）
claude --prompt "以下のコード品質レポートを基に、patent_analyzer.pyの改善提案をしてください。\
" --file quality_report.json > improved_patent_analyzer.py
```

## パフォーマンス最適化

```
# コード例: claude-code-cli-commands.sh（続き）
# 詳細は code-examples/claude-code-cli-commands.sh を参照
```

# 5-4. 寝てる間に進めてもらう(サンプル)

## 自動化戦略

- **スケジューリング**: 定期的なタスク実行・時間帯最適化
- **条件分岐**: 状況に応じた処理選択・優先度の動的調整
- **通知機能**: 完了・異常の自動通知・進捗レポート
- **継続学習**: 実行結果からの学習・改善の自動化

## 実装例

コード例: nightly-patent-analysis.py

詳細は code-examples/nightly-patent-analysis.py を参照

## 注意点

- **品質管理**: 自動処理結果の確認・検証システムの構築
- **エラーハンドリング**: 異常時の対応・復旧機能の実装
- **セキュリティ**: 機密情報の保護・アクセス制御の強化
- **監視・ログ**: 実行状況の監視・詳細ログの記録

# 5-5. 自動化システムの構築

## システムアーキテクチャ

```
# コード例: fully-automated-patent-system.py
# 詳細は code-examples/fully-automated-patent-system.py を参照
```

# 5-6. 自動化の効果測定

## 定量的効果

### 効率性指標

- **処理時間**: 自動化前後の処理時間比較
- **処理量**: 単位時間あたりの処理件数
- **コスト**: 人的コスト・運用コストの削減
- **精度**: 自動化による精度向上

**品質指標**

- **エラー率**: 人的エラーの削減
- **一貫性**: 処理結果の標準化
- **完全性**: 処理漏れの防止
- **タイムリー性**: リアルタイム処理の実現

# 5-7. AI エージェントのクラウドベンダー比較

## 主要クラウドベンダーの AI エージェントサービス

出典:生成 AI の AI エージェントを大手 3 社（AWS、Azure、Google Cloud）で徹底比較してみた [7]

### Amazon Bedrock Agents（AWS）

- **構成要素**: エージェント、ツール、ナレッジベース、アクション
- **対応モデル**: Claude 3.5 Sonnet、Claude 3 Haiku、Llama 3.1 405B
- **主要機能**:
  - 自然言語での複雑なタスク実行
  - ツールの動的呼び出し
  - ナレッジベースとの統合
  - アクションの実行（Lambda 関数等）
- **料金**: 入力トークン $0.00015/1K、出力トークン $0.0006/1K

## Azure AI Agent Services（Azure）

### 構成要素

- **AI Agent**: 自然言語処理と意思決定
- **Tools**: 外部サービスとの連携
- **Knowledge Base**: 企業データの統合
- **Actions**: 具体的なタスク実行

### 対応モデル

- **GPT-4.1**: 最新の高性能モデル（2025 年 4 月リリース）
- **o1-pro**: 高度な推論能力を持つ最新モデル（2025 年 3 月リリース）
- **GPT-4o**: 高性能なマルチモーダルモデル
- **GPT-4o mini**: コスト効率を重視した軽量モデル
- **GPT-4 Turbo**: バランスの取れた性能

## Azure AI Agent Services（Azure）の機能

### 主要機能

- **マルチエージェントシステム**: 複数エージェントの協調
- **高度なツール連携**: カスタムツールの開発・統合
- **セキュリティ**: エンタープライズレベルのセキュリティ
- **スケーラビリティ**: 大規模システムへの対応

## Vertex AI Agents（Google Cloud）

### 構成要素

- **Agent**: 自律的な意思決定を行う AI エージェント
- **Tools**: 外部サービスとの連携・API 呼び出し
- **Knowledge Base**: 企業データの統合・RAG システム
- **Actions**: 具体的なタスク実行・ワークフロー管理

### 対応モデル

- **Gemini 2.5 Pro**: 最新の高性能モデル
- **Gemini 2.5 Flash**: 高速処理重視

## Vertex AI Agents（Google Cloud）の機能

### 主要機能

- **自律的エージェント**: 複雑なタスクの自動実行
- **マルチターン対話**: 継続的な対話による課題解決
- **高度なツール連携**: カスタムツールの開発・統合
- **セキュリティ**: エンタープライズレベルのセキュリティ
- **スケーラビリティ**: 大規模システムへの対応

## 3 社比較：機能面での違い

| 機能 | AWS Bedrock Agents | Azure AI Agent Services | Vertex AI Agents |
| --- | --- | --- | --- |
| **マルチエージェント** | 対応 | 対応 | 対応 |
| **カスタムツール** | 対応 | 対応 | 対応 |
| **ナレッジベース** | 対応 | 対応 | 対応 |
| **アクション実行** | Lambda 等 | カスタムアクション | カスタムアクション |
| **自律的対話** | 制限的 | 制限的 | マルチターン対話 |
| **セキュリティ** | 高 | 高 | 高 |

## 3 社比較：料金面での違い

### 想定シナリオ：社内ナレッジシステム（月間 100 万トークン）

| ベンダー | モデル | 月間料金 | 特徴 |
|---|---|---|---|
| **AWS** | Claude 4 Opus | $200 | 最高性能 |
| **AWS** | Claude 4 Sonnet | $150 | バランス型 |
| **Azure** | GPT-4.1 | $1,800 | 最新高性能 |
| **Azure** | GPT-4o | $1,500 | マルチモーダル |
| **Google Cloud** | Gemini 2.5 Pro | $2,000 | 最新思考型モデル |
| **Google Cloud** | Gemini 2.5 Flash | $50 | 高速・低コスト |
| **Anthropic** | Claude 4 Opus | $200 | 最高品質 |
| **OpenAI** | GPT-4.1 | $1,800 | 最新汎用性 |

## 選択の指針

### AWS Bedrock Agents が適している場合

- **安定性重視**: 実績のある Claude モデルの活用
- **コスト効率**: 比較的安価な料金体系
- **AWS 環境**: 既存の AWS インフラとの統合
- **シンプルな構成**: 基本的なエージェント機能で十分

### Azure AI Agent Services が適している場合

- **高性能要求**: GPT-5 の最新機能が必要
- **Microsoft 環境**: 既存の Microsoft 製品との統合
- **エンタープライズ**: 大企業向けのセキュリティ・ガバナンス
- **マルチエージェント**: 複雑な協調システムの構築

## 選択の指針（続き）

### Vertex AI Agents が適している場合

- **最新技術**: Gemini 2.5 Pro の最新機能を活用
- **自律的エージェント**: 複雑なタスクの自動実行
- **Google 環境**: 既存の Google Cloud サービスとの統合
- **高速処理**: Gemini 2.5 Flash による高速処理
- **マルチターン対話**: 継続的な問題解決が必要

## 実装時の考慮事項

### 技術的考慮事項

- **既存インフラ**: 現在使用中のクラウド環境
- **統合要件**: 既存システムとの連携
- **スケーラビリティ**: 将来的な拡張性
- **セキュリティ**: データ保護・アクセス制御

### 組織的考慮事項

- **スキルセット**: チームの技術力
- **予算**: 初期投資・運用コスト
- **タイムライン**: 開発・導入スケジュール
- **リスク**: 技術的・組織的リスク

# 5-9. Vellum AI LLM リーダーボードの活用

## Vellum AI リーダーボードの概要

出典: Vellum AI LLM Leaderboard [21]

## 最新の LLM 性能比較

- **更新日**: 2025 年 8 月 7 日
- **対象モデル**: 2024 年 4 月以降にリリースされた最新モデル
- **評価基準**: 非飽和ベンチマーク、独立評価
- **特徴**: 実用的なベンチマークに焦点

## 主要タスク別トップモデル

### 推論能力（GPQA Diamond）

- **Grok 4**: 87.5%
- **GPT-5**: 87.3%
- **Gemini 2.5 Pro**: 86.4%
- **Grok 3 [Beta]**: 84.6%
- **OpenAI o3**: 83.3%

### 高校数学（AIME 2025）

- **GPT-5**: 100%
- **GPT oss 20b**: 98.7%
- **OpenAI o3**: 98.4%
- **GPT oss 120b**: 97.9%

## エージェント・コーディング性能

### エージェントコーディング（SWE Bench）

- **Grok 4**: 75%
- **GPT-5**: 74.9%
- **Claude Opus 4.1**: 74.5%
- **Claude 4 Sonnet**: 72.7%
- **Claude 4 Opus**: 72.5%

### ツール使用（BFCL）

- **Llama 3.1 405b**: 81.1%
- **Llama 3.3 70b**: 77.3%
- **GPT-4o**: 72.08%
- **GPT-4.5**: 69.94%

## 速度・コスト性能

### 最速モデル（トークン/秒）

- **Llama 4 Scout**: 2,600 t/s
- **Llama 3.3 70b**: 2,500 t/s
- **Llama 3.1 70b**: 2,100 t/s
- **Llama 3.1 8b**: 1,800 t/s
- **Llama 3.1 405b**: 969 t/s

### 最低遅延（TTFT）

- **Nova Micro**: 0.3 秒
- **Llama 3.1 8b**: 0.32 秒
- **Llama 4 Scout**: 0.33 秒
- **Gemini 2.0 Flash**: 0.34 秒

## コスト効率

### 最も安価なモデル（100 万トークンあたり）

- **Nova Micro**: $0.04（入力）/ $0.14（出力）
- **Gemma 3 27b**: $0.07（入力）/ $0.07（出力）
- **Gemini 1.5 Flash**: $0.075（入力）/ $0.3（出力）
- **GPT oss 20b**: $0.08（入力）/ $0.35（出力）

### 知財業務での選択指針

- **高精度要求**: GPT-5、Grok 4、Gemini 2.5 Pro
- **コスト効率**: Nova Micro、Gemma 3 27b
- **高速処理**: Llama 4 Scout、Llama 3.3 70b
- **バランス型**: Claude 3.5 Sonnet、GPT-4o

## 知財業務での活用戦略

### 用途別モデル選択

#### 特許分析・技術評価

- **推奨モデル**: GPT-5、Grok 4、Gemini 2.5 Pro
- **理由**: 高い推論能力、複雑な技術文書の理解

#### 大量データ処理

- **推奨モデル**: Llama 4 Scout、Nova Micro
- **理由**: 高速処理、低コスト

### エージェント開発

- **推奨モデル**: Grok 4、GPT-5、Claude Opus 4.1
- **理由**: 高いエージェントコーディング性能

### 日常業務支援

- **推奨モデル**: Claude 3.5 Sonnet、GPT-4o
- **理由**: バランスの取れた性能とコスト

# 5-10. MCP (Model Context Protocol) と A2A (Agent-to-Agent) プロトコル

## MCP と A2A の基本概念

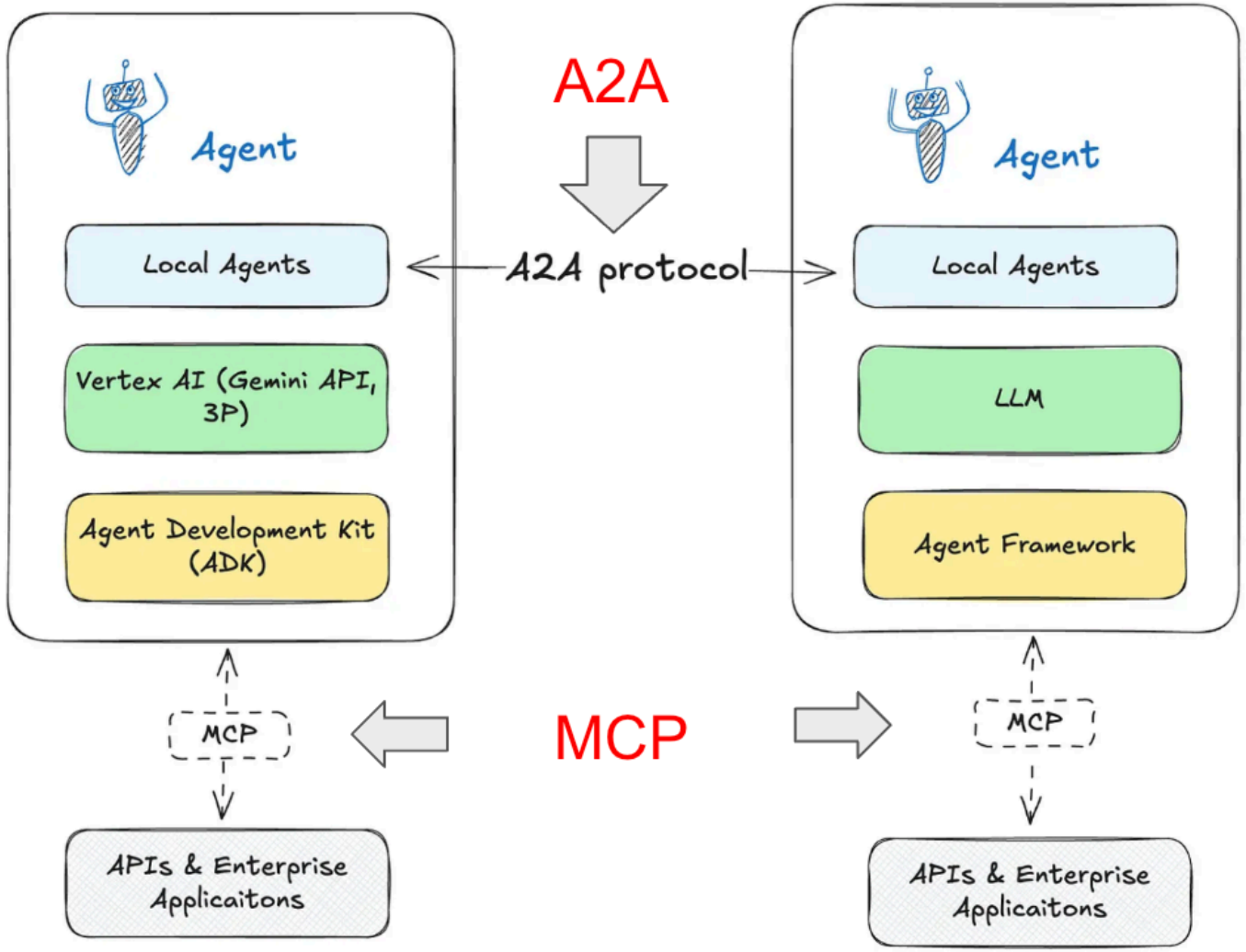
### MCP (Model Context Protocol)

- **標準化プロトコル**: AI モデルとアプリケーション間の標準プロトコル
- **相互運用性**: 異なるシステム間の連携
- **拡張性**: 新機能の追加が容易
- **オープンソース**: オープンな標準規格

## A2A (Agent-to-Agent)

- **エージェント間通信**: AI エージェント間の通信プロトコル
- **協調作業**: 複数エージェントの協調
- **タスク分担**: 効率的なタスク分担
- **結果統合**: 複数結果の統合

# MCP/A2A アーキテクチャ



**アーキテクチャの構成要素（例）**

**左側の Agent システム**

- **Local Agents**: 他の Agent システムとの A2A 通信の起点
- **Vertex AI (Gemini API, 3P)**: Google Cloud の Vertex AI、Gemini API、サードパーティ AI サービス
- **Agent Development Kit (ADK)**: Agent 開発キット

**右側の Agent システム**

- **Local Agents**: 他の Agent システムとの A2A 通信の起点
- **LLM**: 大規模言語モデル
- **Agent Framework**: Agent 構築フレームワーク

## 通信プロトコル

- **A2A Protocol**: Agent 間の直接通信
- **MCP**: 外部 API・エンタープライズアプリケーションとの連携

## 知財業務での活用例

### USPTO Patent MCP Server

FlowHunt USPTO Patent MCP [32] ：米国特許商標庁（USPTO）の特許データにアクセスするための MCP サーバー

**主要機能:**

- **特許検索**: USPTO Public Search API を使用した特許・出願の検索
- **全文取得**: 特許文書の完全なテキスト取得（クレーム、明細書等）
- **PDF ダウンロード**: USPTO ソースからの直接 PDF 取得
- **包括的メタデータ**: 特許の書誌情報、譲渡、訴訟データの取得
- **Claude Desktop 統合**: 安全なローカル分析とレポート生成

**利用可能なツール:**

- `ppubs_search_patents`: 特許検索
- `ppubs_search_applications`: 出願検索
- `ppubs_get_full_document`: 完全文書取得
- `get_app_metadata`: メタデータ取得
- `get_app_assignment`: 譲渡履歴取得
- `get_app_litigation`: 訴訟データ取得

**知財業務での活用:**

- 先行技術調査の自動化
- 競合分析の効率化
- 特許ポートフォリオ管理
- 技術動向の監視

### Google Patents MCP Server

Google Patents MCP Server [33] ：Google Patents の情報を検索するための MCP サーバー。SerpApi Google Patents API をバックエンドとして使用

**主要機能:**

- **Google Patents 検索**: SerpApi を使用した Google Patents 検索
- **高度な検索フィルター**: 発明者、譲受人、国、言語、ステータス等での絞り込み
- **日付フィルター**: 出願日、公開日、優先日での期間指定
- **ソート機能**: 関連性、新着順、古い順での並び替え
- **npx 実行**: ローカルインストール不要での直接実行

**利用可能なツール:**

- `search_patents` : Google Patents 検索（主要ツール）

**検索パラメータ:**

- `q` : 検索クエリ（必須）
- `page` : ページ番号
- `num` : 1 ページあたりの結果数（10-100）
- `sort` : ソート方法（relevance, new, old）
- `before/after` : 日付フィルター
- `inventor` : 発明者名フィルター
- `assignee` : 譲受人名フィルター
- `country` : 国コードフィルター
- `language` : 言語フィルター

**知財業務での活用:**

- Google Patents での包括的な特許検索
- 多言語特許文献の検索
- 時系列での技術動向分析
- 特定企業・発明者による特許調査

**知財業務での活用:**

(手前味噌で住みませんが)
Playwright MCP を使って特許調査
https://www.enlighton.co.jp/post/playwright-mcpを使って特許調査 [31]

- MCP サーバを使った事例・エコシステムは知財分野でもどんどん増えるはず。
- 欲しい機能を持った MCP に、**言葉で**話しかけるだけで用事が済む
- A2A でもっと便利に（1 人目の Agent に話しかければ全部済む）

## 知財業務での活用例(作ってみる)

### 分散特許調査システム

```
# コード例: distributed-patent-research.py
# 詳細は code-examples/distributed-patent-research.py を参照
```

### 特許分析エージェントの実装例

```
# コード例: distributed-patent-research.py（続き）
# 詳細は code-examples/distributed-patent-research.py を参照
```

# MCP/A2A の導入戦略

## 段階的導入アプローチ

### Phase 1: 基盤構築（1-2 ヶ月）

1. MCP クライアントの設定
   - プロトコルの理解
   - 基本設定の実装
   - テスト環境の構築

2. 単一エージェントの開発
   - 基本的なエージェント機能
   - MCP との連携
   - 品質保証の実装

## Phase 2: マルチエージェント化（2-3 ヶ月）

1. A2A プロトコルの実装
   - エージェント間通信の設定
   - タスク分配ロジックの実装
   - 結果統合機能の開発

2. 協調作業の実装
   - 複数エージェントの協調
   - 競合解決の仕組み
   - 品質管理の強化

## Phase 3: 本格運用（3-6 ヶ月）

1. 大規模システムへの展開
   - スケーラビリティの確保
   - パフォーマンスの最適化
   - 監視・ログ機能の強化

2. 継続的改善
   - 効果測定の実施
   - 新機能の追加
   - ベストプラクティスの確立

## 効果測定と改善

```
# コード例: mcp-a2a-metrics.py
# 詳細は code-examples/mcp-a2a-metrics.py を参照
```

## 将来展望と課題 技術的展望

### 短期（1 年以内）

- **プロトコル標準化**: MCP/A2A の標準化の進展
- **ツールエコシステム**: 開発ツールの充実
- **パフォーマンス向上**: 通信効率の改善

### 中期（1-3 年）

- **高度な協調**: より複雑な協調作業の実現
- **自律性の向上**: エージェントの自律性強化
- **セキュリティ強化**: セキュアな通信の実現

## 将来展望と課題 技術的展望

### 課題と対策

### 技術的課題

- **プロトコル標準化**: 業界標準の確立
- **相互運用性**: 異なるシステム間の連携
- **セキュリティ**: セキュアな通信の確保
- **スケーラビリティ**: 大規模システムへの対応

# Agent Design Catalogue—どんな AI エージェントアーキテクチャがあるか