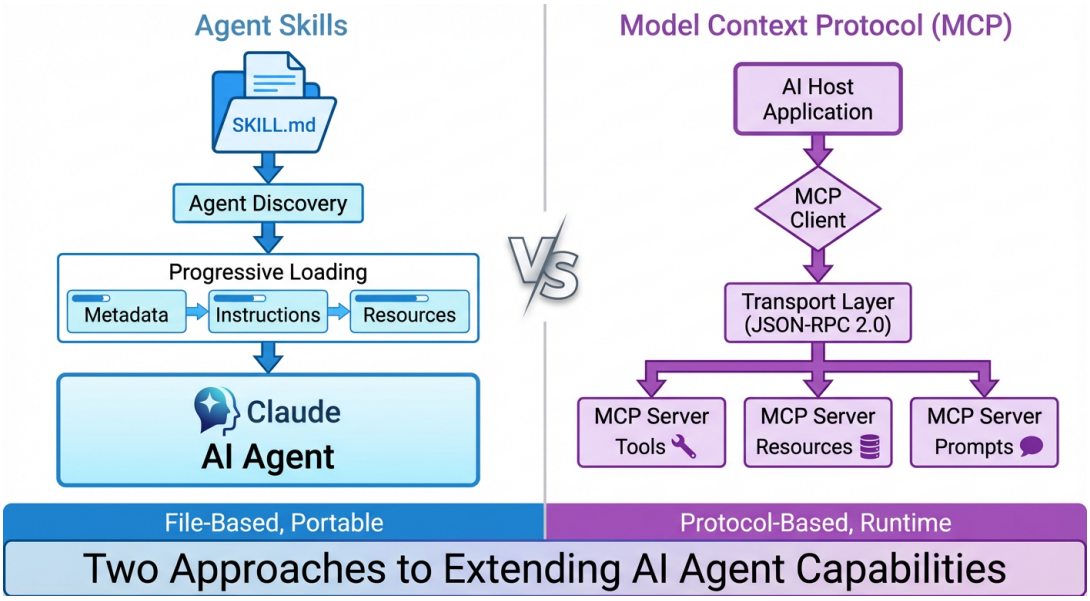
Agent Skills vs Model Context Protocol (MCP)

A Comprehensive Technical Comparison



Two Approaches to Extending AI Agent Capabilities

Author: K-Dense Web
Email: contact@k-dense.ai
Date: January 1, 2026
Document Type: Technical Report

Generated using K-Dense Web (k-dense.ai)

Contents

1 Executive Summary

As artificial intelligence agents become increasingly sophisticated, the need for standardized methods to extend their capabilities has emerged as a critical challenge in the field. Two prominent approaches have arisen to address this need: **Agent Skills** and the **Model Context Protocol (MCP)**. While both aim to enhance AI agent functionality, they represent fundamentally different architectural philosophies with distinct trade-offs.

Agent Skills, developed by Anthropic and released as an open standard, provides a **file-based, portable format** for packaging domain expertise, workflows, and executable code into self-contained skill directories. The approach emphasizes simplicity, portability, and human-readable specifications through SKILL.md files with YAML frontmatter and Markdown instructions.

Model Context Protocol (MCP), also initiated by Anthropic, establishes a **standardized communication protocol** based on JSON-RPC 2.0 for connecting AI applications to external systems, tools, and data sources. MCP functions as "USB-C for AI applications"—providing a universal interface layer between AI hosts and capability-providing servers.

1.1 Key Findings

Dimension	Agent Skills	MCP
Architecture	File-based folders with SKILL.md	Client-Host-Server with JSON-RPC 2.0
Integration	Low (drop files into directory)	Moderate (requires SDK)
State Management	Stateless (file reads)	Stateful (connection lifecycle)
Best For	Knowledge packaging, workflows	External system integration
Security Model	Sandbox execution, allowlisting	OAuth 2.1, transport security

Table 1: Summary comparison of Agent Skills and MCP

Recommendation: These technologies are **complementary rather than competitive**. Organizations should consider Agent Skills for packaging domain expertise and standardized workflows, while leveraging MCP for real-time integration with external systems, databases, and APIs.

2 Introduction

2.1 Background and Motivation

The rapid advancement of large language models (LLMs) has catalyzed a fundamental shift in how we conceptualize AI applications. Modern AI agents are no longer limited to text generation—they can execute code, query databases, interact with APIs, and perform complex multi-step tasks. However, extending agent capabilities has traditionally required custom integrations, leading to fragmented ecosystems and duplicated effort across organizations.

Two open standards have emerged to address this challenge:

1. **Agent Skills**: A simple, open format for packaging domain expertise and workflows into portable, version-controlled skill directories
2. **Model Context Protocol (MCP)**: A standardized protocol for connecting AI applications to external tools, data sources, and systems

Both standards originated from Anthropic's work on AI agents but represent distinctly different approaches to the capability extension problem.

2.2 Purpose of This Report

This technical report provides system architects, developers, and technical decision-makers with:

- A comprehensive understanding of both Agent Skills and MCP architectures
- Detailed comparison across key technical dimensions
- Guidance on selecting the appropriate approach for specific use cases
- Best practices for implementation and integration

2.3 Scope

This analysis covers the latest publicly available specifications as of January 2026:

- **Agent Skills**: Open specification at agentskills.io
- **MCP**: Version 2025-11-25 at modelcontextprotocol.io

3 Technical Overview: Model Context Protocol (MCP)

3.1 Core Architecture

MCP implements a three-layer architecture designed to standardize communication between AI applications and external capability providers.

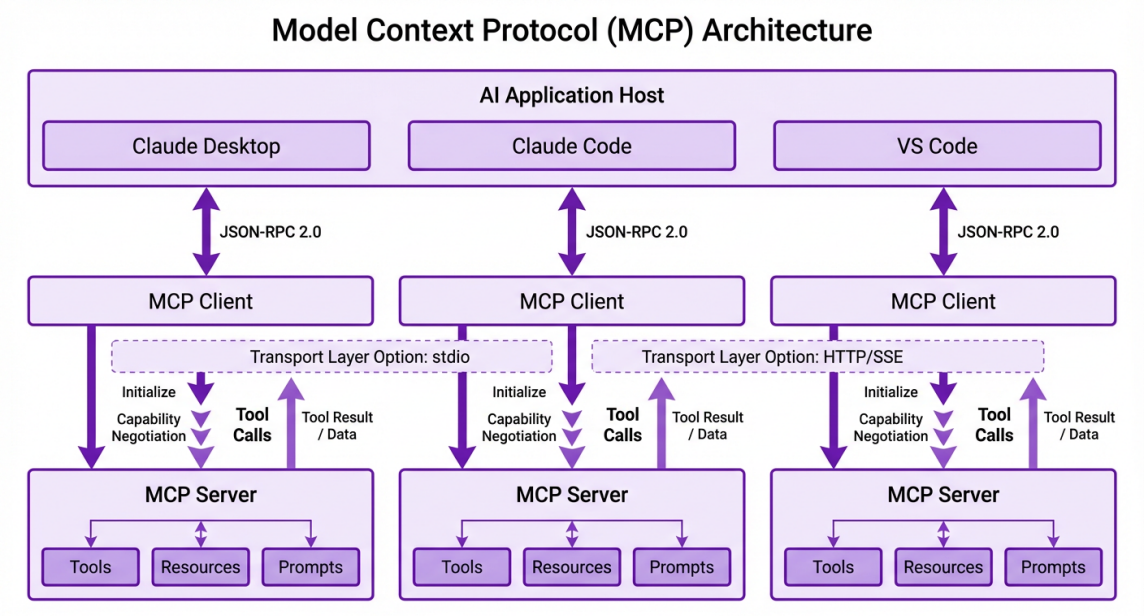


Figure 1: Model Context Protocol Architecture — Three-tier Client-Host-Server model with JSON-RPC 2.0 communication protocol.

3.1.1 Architectural Components

MCP Host: The AI application (e.g., Claude Desktop, Claude Code, VS Code) that coordinates and manages multiple MCP clients. The host:

- Instantiates separate MCP client objects for each server connection
- Aggregates tools from all connected servers into a unified registry
- Routes tool invocations to the appropriate MCP server
- Manages the overall lifecycle of MCP connections

MCP Client: A component that maintains a connection to an MCP server and obtains context for the host. Each client handles:

- Connection establishment and lifecycle management
- Capability negotiation with its paired server
- Message serialization and deserialization
- Notification handling for dynamic capability updates

MCP Server: A program that provides context, tools, and resources to MCP clients. Servers expose:

- **Tools**: Executable functions the AI can invoke
- **Resources**: Data sources providing contextual information
- **Prompts**: Reusable interaction templates

3.2 Communication Protocol

MCP is built on **JSON-RPC 2.0**, a lightweight remote procedure call protocol. The protocol defines three message types:

3.2.1 Request Messages

Requests require a response from the server:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/list",
  "params": {}
}
```

Listing 1: MCP Request Example

3.2.2 Response Messages

Responses are paired with requests:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "tools": [
      {
        "name": "weather_current",
        "description": "Fetch current weather",
        "inputSchema": {
          "type": "object",
          "properties": {
            "location": { "type": "string" }
          },
          "required": ["location"]
        }
      }
    ]
  }
}
```

Listing 2: MCP Response Example

3.2.3 Notification Messages

Notifications are fire-and-forget (no response expected):

```
{
  "jsonrpc": "2.0",
```

```
  "method": "notifications/tools/list_changed"
}
```

Listing 3: MCP Notification Example

3.3 Transport Layer

MCP supports multiple transport mechanisms:

Transport	Use Case	Benefits
stdio	Local process communication	Optimal performance, no network overhead
HTTP/SSE	Remote servers	Standard HTTP authentication support
Custom	Specialized requirements	Flexibility for unique deployments

Table 2: MCP Transport Options

3.4 Connection Lifecycle

MCP connections follow a defined lifecycle:

1. **Initialization**: Client sends `initialize` request with capabilities
2. **Capability Negotiation**: Server responds with its capabilities
3. **Ready Notification**: Client sends `notifications/initialized`
4. **Operation**: Normal request/response message exchange
5. **Termination**: Graceful connection closure

3.5 Core Primitives

3.5.1 Server-Exposed Primitives

Primitive	Purpose	Key Methods
Tools	Executable functions	`tools/list`, `tools/call`
Resources	Data sources	`resources/list`, `resources/read`
Prompts	Interaction templates	`prompts/list`, `prompts/get`

Table 3: MCP Server Primitives

3.5.2 Client-Exposed Primitives

Primitive	Purpose
Sampling	Request LLM completions via `sampling/complete`
Elicitation	Request user input via `elicitation/request`
Logging	Send debug/monitoring messages

Table 4: MCP Client Primitives

3.6 Security Model

MCP implements a multi-layered security approach:

1. **Authorization**: Optional OAuth 2.1 support for sensitive operations
2. **Transport Security**: HTTPS for remote connections
3. **Access Control**: Per-operation permissions
4. **Input Validation**: Schema-based parameter validation
5. **Audit Logging**: Tracking of sensitive operations

3.7 Ecosystem and Adoption

MCP has achieved broad adoption across the AI ecosystem:

- **Official SDKs**: Python, TypeScript/JavaScript
- **Community SDKs**: Rust, Go, and others
- **Pre-built Servers**: File system, database, API integrations
- **Development Tools**: MCP Inspector for debugging and testing

4 Technical Overview: Agent Skills

4.1 Core Architecture

Agent Skills implement a file-based architecture designed for portability, simplicity, and progressive context loading.

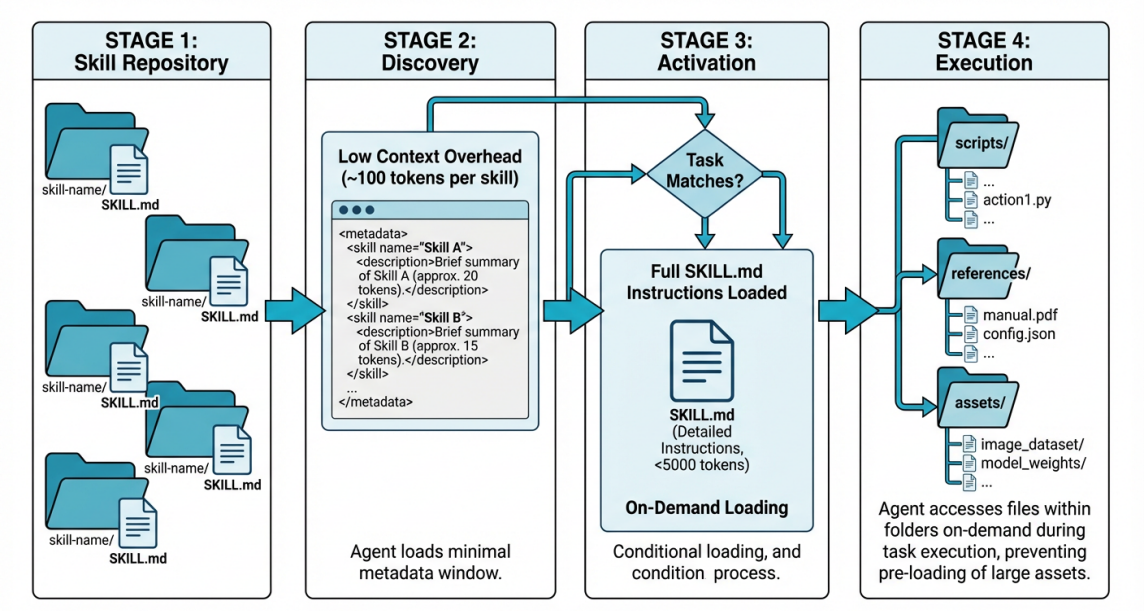


Figure 2: Agent Skills Architecture — Progressive disclosure model with four-stage loading: Repository → Discovery → Activation → Execution.

4.2 Skill Structure

Every skill is a directory containing a `SKILL.md` file with YAML frontmatter and Markdown body:

```
my-skill/
|-- SKILL.md              # Required: Specification file
|-- scripts/              # Optional: Executable code
|   `-- analyze.py
|-- references/           # Optional: Documentation
|   `-- REFERENCE.md
`-- assets/               # Optional: Templates, data
    `-- template.json
```

Listing 4: Agent Skills Directory Structure

4.2.1 SKILL.md Format

```
---
name: data-analysis
description: Provides data analysis capabilities including
  statistical testing, visualization, and dataset exploration
license: MIT
compatibility: Requires pandas, matplotlib
---
```

```
8
9  # Data Analysis Skill
10
11 ## Overview
12 This skill enables comprehensive data analysis workflows...
13
14 ## Usage Instructions
15 1. Load dataset using `scripts/load_data.py`
16 2. Run statistical analysis with `scripts/analyze.py`
17 3. Generate visualizations...
```

Listing 5: Example SKILL.md File

4.2.2 Required Fields

Field	Requirements
name	1-64 characters, lowercase alphanumeric with hyphens
description	1-1024 characters describing functionality

Table 5: Required SKILL.md Fields

4.2.3 Optional Fields

Field	Purpose
license	Licensing terms or reference to bundled license
compatibility	Environment requirements, system packages
metadata	Key-value mapping for custom properties
allowed-tools	Pre-approved tool list (experimental)

Table 6: Optional SKILL.md Fields

4.3 Progressive Disclosure Model

Agent Skills implement a three-stage loading mechanism to optimize context usage:

Stage 1: Metadata (~100 tokens per skill)
Only `name` and `description` loaded at startup for all skills. Enables the agent to determine relevance without loading full instructions.

Stage 2: Instructions (<5,000 tokens)
Full `SKILL.md` body loaded when a task matches the skill's purpose. Provides detailed instructions, examples, and workflows.

Stage 3: Resources (On-demand)
Scripts, references, and assets loaded only when specifically needed during execution.

4.4 Integration Methods

4.4.1 Filesystem-Based Agents

For agents with computer environment access (bash/unix):

```
# Agent reads skill directly
cat /path/to/skills/my-skill/SKILL.md
```

Listing 6: Filesystem-based Skill Loading

This represents the most capable integration option, as agents can directly read and execute skill contents.

4.4.2 Tool-Based Agents

For agents without filesystem access, implement tools that:

1. List available skills
2. Load skill metadata
3. Activate skill instructions
4. Execute bundled scripts

4.5 System Prompt Integration

Skills are surfaced to agents via XML in system prompts:

```
<available_skills>
  <skill>
    <name>data-analysis</name>
    <description>Statistical testing and visualization</description>
    <location>/path/to/skills/data-analysis/SKILL.md</location>
  </skill>
</available_skills>
```

Listing 7: System Prompt Skill Integration

Each skill adds approximately 50-100 tokens to the context.

4.6 Security Considerations

- **Sandboxing**: Execute scripts in isolated environments
- **Allowlisting**: Run scripts only from trusted skill sources
- **User Confirmation**: Request approval before dangerous operations
- **Audit Logging**: Record all script executions

4.7 Ecosystem and Adoption

Agent Skills is supported by major platforms including Claude Code, Claude Desktop, VS Code, OpenCode, Cursor, Amp, Letta, Goose, GitHub, and OpenAI Codex.

5 Comparative Analysis

5.1 Architectural Comparison

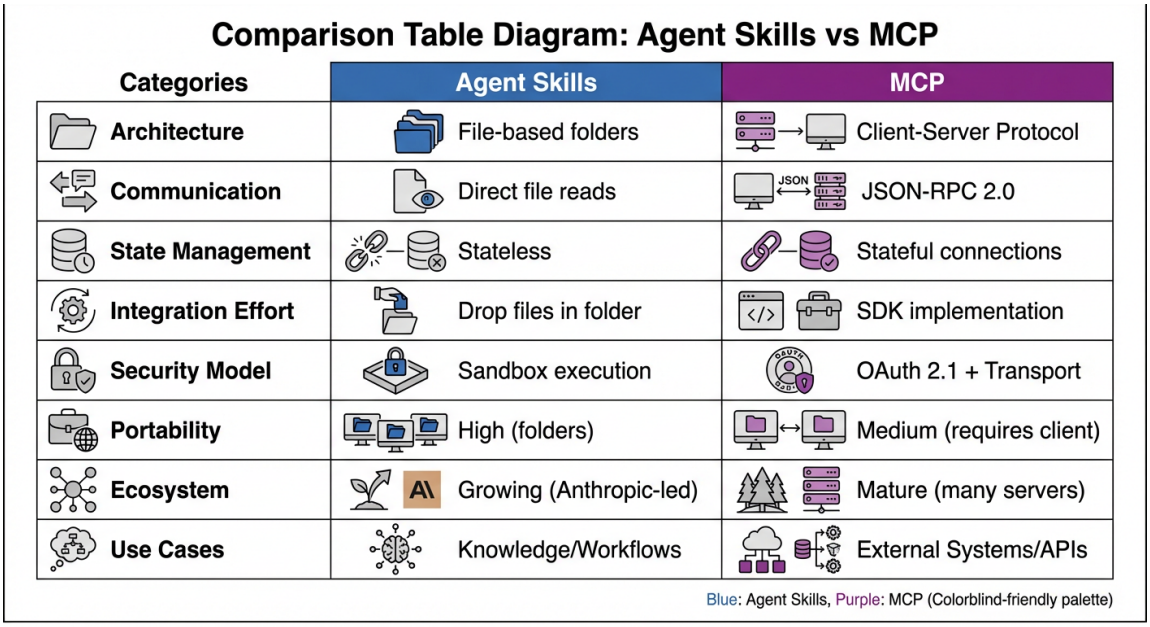


Figure 3: Side-by-side comparison of Agent Skills and MCP across key architectural dimensions.

5.1.1 Fundamental Philosophy

Aspect	Agent Skills	MCP
Paradigm	File-based packaging	Protocol-based communication
Metaphor	"Plug-and-play skill folders"	"USB-C for AI applications"
State	Stateless (each read independent)	Stateful (connection lifecycle)
Complexity	Minimal	Moderate

Table 7: Fundamental Philosophical Differences

5.1.2 Communication Model

Agent Skills: Direct file system access

- Agent reads SKILL.md and associated files
- No runtime protocol overhead
- Works offline
- Simple debugging (inspect files directly)

MCP: JSON-RPC 2.0 protocol

- Bidirectional message exchange
- Capability negotiation at connection time

- Real-time notifications for dynamic updates
- Structured request/response patterns

5.2 Data Flow Comparison

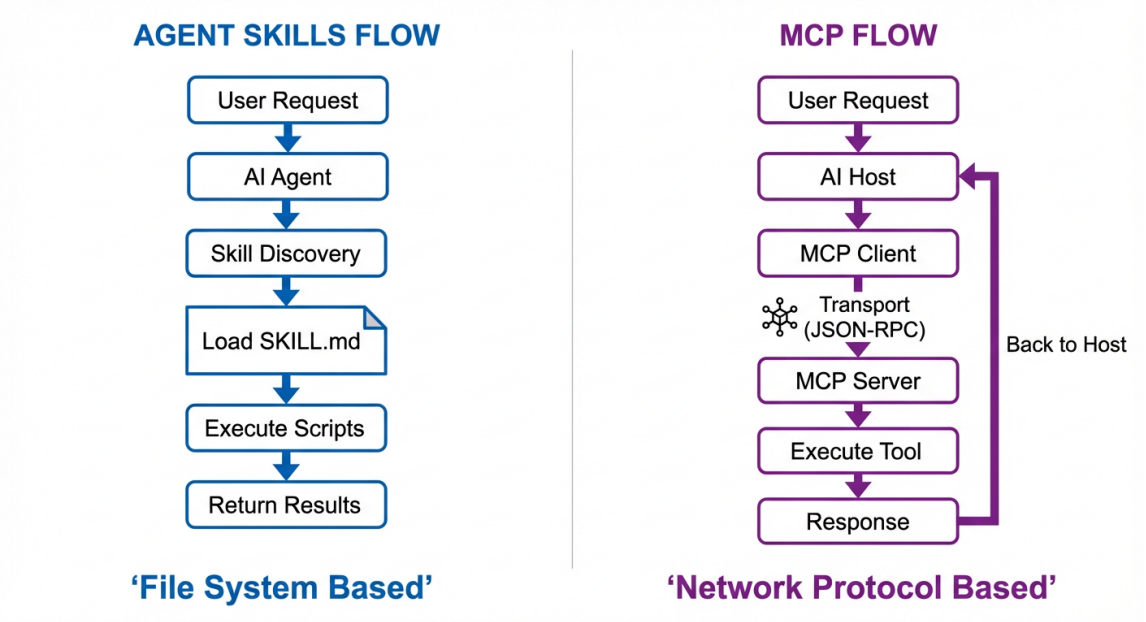


Figure 4: Data flow comparison between Agent Skills (file-based) and MCP (protocol-based) approaches.

5.2.1 Agent Skills Data Flow

1. User Request
2. AI Agent
3. Skill Discovery (scan directories)
4. Match Task to Skill
5. Load SKILL.md Instructions
6. Execute Scripts (if needed)
7. Return Results to User

Characteristics: Synchronous file-based operations, no connection management, lazy loading of resources, simple error handling.

5.2.2 MCP Data Flow

1. User Request
2. AI Host Application
3. MCP Client Selection

4. Transport Layer (JSON-RPC 2.0)
5. MCP Server Processing
6. Tool Execution
7. Response via Transport
8. Result to Host → User

Characteristics: Asynchronous message-based operations, connection lifecycle management, capability negotiation, structured error handling.

5.3 Security Comparison

Security Aspect	Agent Skills	MCP
Authentication	Trust-based (allowlisted skills)	OAuth 2.1, API keys, bearer tokens
Authorization	Per-skill allowlisting	Per-operation access control
Transport	Local file system	HTTPS, TLS for remote
Isolation	Sandbox execution	Server-side isolation
Audit	Execution logging	Full message logging

Table 8: Security Model Comparison

5.4 Developer Experience

5.4.1 Creating New Capabilities

Agent Skills (Time to first capability: ∼5 minutes):

```
mkdir my-skill
cat > my-skill/SKILL.md << 'EOF'
---
name: my-skill
description: Does something useful
---
# My Skill
Instructions here...
EOF
```

MCP (Time to first capability: ∼15-30 minutes):

```
from mcp import MCPServer

server = MCPServer("my-server")

@server.tool()
def my_tool(param: str) -> str:
    """Does something useful"""
    return f"Result: {param}"

server.run()
```

5.4.2 Integration Effort

Task	Agent Skills	MCP
Setup	Drop folder into skills directory	Configure client, install SDK
Dependencies	None (file-based)	SDK, transport configuration
Testing	Read files, check syntax	MCP Inspector, integration tests
Debugging	Inspect files directly	Trace JSON-RPC messages
Deployment	Copy folders	Deploy server, configure connections

Table 9: Integration Effort Comparison

5.5 Ecosystem and Interoperability

5.5.1 Agent Skills Ecosystem

Strengths:

- Extremely portable (just files)
- Human-readable and auditable
- Version control friendly
- Cross-platform compatibility
- Low barrier to entry

Limitations:

- Limited to file-system operations
- No real-time external system access
- Execution depends on agent capabilities
- Newer ecosystem, fewer pre-built skills

5.5.2 MCP Ecosystem

Strengths:

- Mature protocol specification
- Official SDKs in multiple languages
- Large collection of pre-built servers
- Real-time integration capabilities
- Strong tooling (Inspector, debugging)

Limitations:

- Higher implementation complexity

- Requires server deployment
- Connection management overhead
- More infrastructure requirements

5.6 Performance Considerations

Metric	**Agent Skills**	**MCP**
Startup Time	Minimal (read metadata)	Connection handshake required
Operation Latency	File I/O only	Network + processing
Memory Overhead	Low (load on demand)	Connection state management
Scalability	Linear with skill count	Depends on server capacity

Table 10: Performance Comparison

6 Use Case Scenarios

6.1 When to Choose Agent Skills

Ideal for:

1. **Domain Expertise Packaging**
   - Legal document workflows
   - Medical diagnosis procedures
   - Financial analysis frameworks
2. **Standardized Workflows**
   - Code review checklists
   - Documentation generation
   - Testing procedures
3. **Portable Knowledge**
   - Cross-organization skill sharing
   - Training and onboarding materials
   - Best practices codification
4. **Offline Scenarios**
   - Air-gapped environments
   - Limited connectivity situations
   - Local-only deployments

6.2 When to Choose MCP

Ideal for:

1. **External System Integration**
   - Database queries
   - API interactions
   - File system operations
2. **Real-Time Data Access**
   - Calendar integration
   - Live monitoring dashboards
   - Dynamic content retrieval
3. **Enterprise Workflows**
   - Multi-system orchestration
   - Authenticated service access
   - Audit-compliant operations

4. **Dynamic Capabilities**
    - Tools that change at runtime
    - Context-dependent functionality
    - Real-time capability negotiation

6.3 Hybrid Approach

Many sophisticated deployments benefit from using **both** technologies:

- **Use Agent Skills for**: What the agent should know and how it should approach problems
- **Use MCP for**: What external systems the agent can interact with

7 Conclusion and Recommendations

7.1 Summary of Findings

Agent Skills and MCP represent two fundamentally different but complementary approaches to extending AI agent capabilities:

Agent Skills excels at:

- Packaging domain expertise in portable, human-readable formats
- Defining standardized workflows and procedures
- Low-friction integration with minimal setup
- Cross-platform, offline-capable deployments

MCP excels at:

- Real-time integration with external systems
- Secure, authenticated access to sensitive resources
- Dynamic capability negotiation
- Enterprise-grade workflows with audit requirements

7.2 Recommendations by Use Case

Scenario	Recommendation
Packaging institutional knowledge	**Agent Skills**
Integrating with databases/APIs	**MCP**
Cross-organization sharing	**Agent Skills**
Real-time data access	**MCP**
Workflow standardization	**Agent Skills**
Multi-system orchestration	**MCP**
Offline/air-gapped environments	**Agent Skills**
Authenticated enterprise access	**MCP**

Table 11: Technology Selection by Use Case

7.3 Implementation Strategy

For organizations building AI agent infrastructure, we recommend:

1. **Start with Agent Skills** for capturing domain expertise and workflows
   - Low barrier to entry
   - Immediately portable across tools
   - Easy to version control and audit
2. **Add MCP** for external system integration needs

- When real-time data access is required
- When authentication/authorization is critical
- When dynamic capabilities are needed

3. **Use both in concert** for comprehensive agent platforms
    - Skills define *how* the agent approaches tasks
    - MCP provides *what* external systems the agent can access

7.4 Future Outlook

Both standards continue to evolve rapidly:

- **Agent Skills** is expanding its ecosystem with more tooling and pre-built skills
- **MCP** is adding new primitives and improving the developer experience

As AI agents become more prevalent, standardization through approaches like these will be essential for building maintainable, secure, and interoperable AI systems.

8 References

1. Agent Skills. (2026). Agent Skills Specification. Retrieved from https://agentskills.io/specification
2. Agent Skills. (2026). What Are Agent Skills? Retrieved from https://agentskills.io/what-are-skills
3. Agent Skills. (2026). Integrating Agent Skills. Retrieved from https://agentskills.io/integrate-skills
4. Model Context Protocol. (2025). Introduction to MCP. Retrieved from https://modelcontextprotocol.io/docs/getting-started/intro
5. Model Context Protocol. (2025). MCP Architecture. Retrieved from https://modelcontextprotocol.io/docs/learn/architecture
6. Model Context Protocol. (2025). Protocol Specification v2025-11-25. Retrieved from https://modelcontextprotocol.io/specification/2025-11-25
7. JSON-RPC Working Group. (2013). JSON-RPC 2.0 Specification. Retrieved from https://www.jsonrpc.org/specification