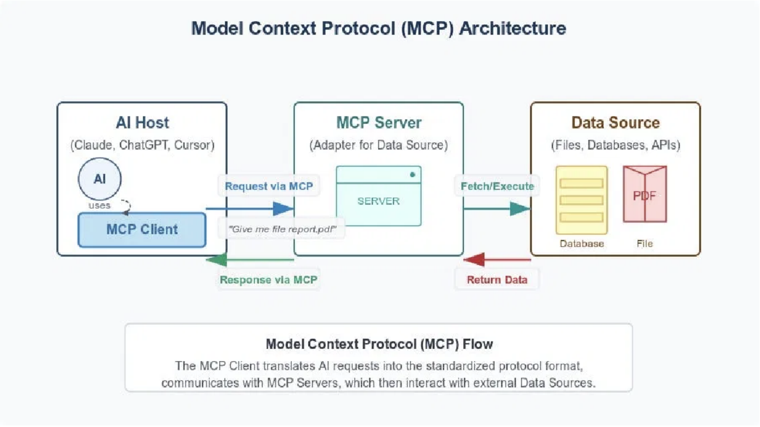
EMPOWERING DEVELOPERS WITH THE MODEL CONTEXT PROTOCOL

Intelligent Knowledge Search and CI/CD Database Querying

Technical Presentation
January 2026



Model Context Protocol (MCP) Flow

The MCP Client translates AI requests into the standardized protocol format, communicates with MCP Servers, which then interact with external Data Sources.

MCP BRIDGES THE GAP BETWEEN AI AND PROPRIETARY DATA

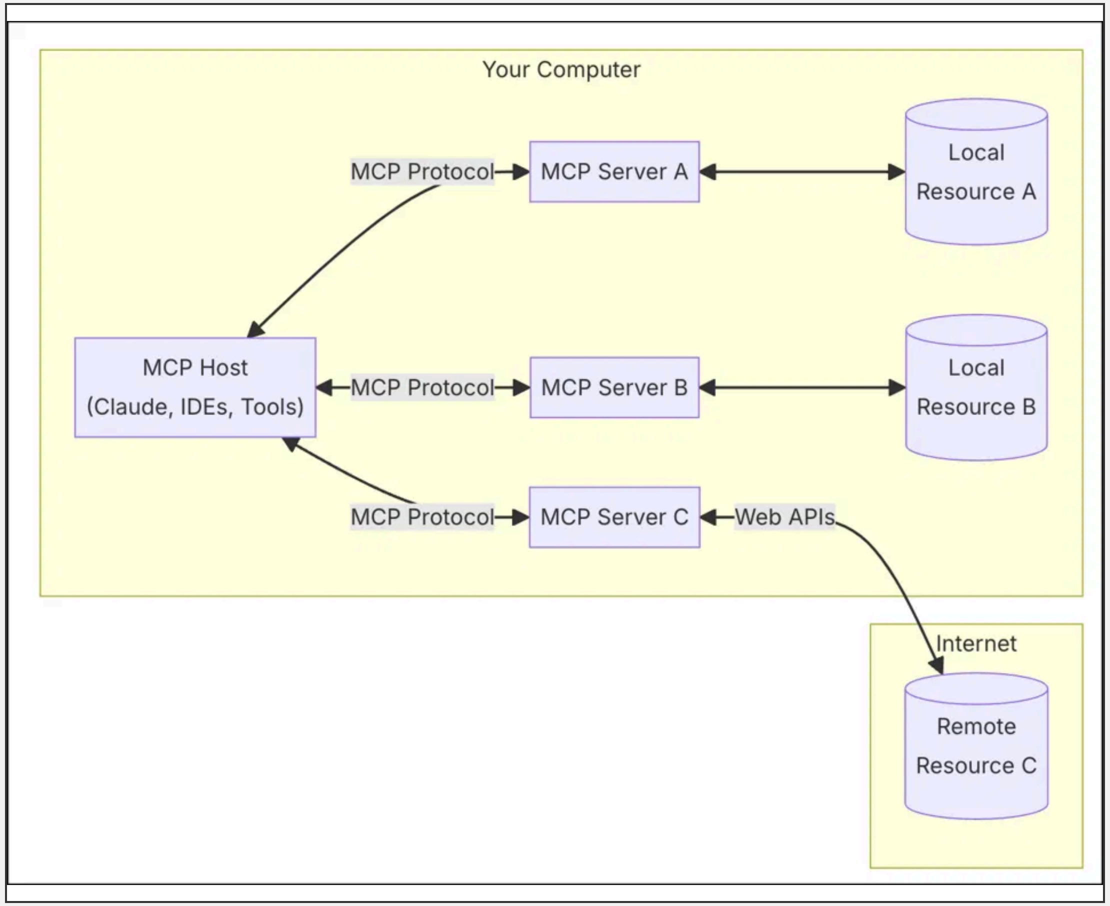
The Problem: Data Isolation

Modern AI assistants are powerful but often operate in a vacuum, lacking access to the real-time, internal data that defines a developer's daily work. This isolation limits their effectiveness in specialized environments.

The Solution: Standardized Bridge

The Model Context Protocol solves this by enabling secure, standardized communication between the IDE and specialized local or remote services. This ensures that data stays within controlled environments.

Key Insight: MCP provides the specific context AI needs to be truly effective, without compromising security or data sovereignty.



A UNIFIED ECOSYSTEM FOR DEVELOPER INTELLIGENCE

The ecosystem consists of three interconnected components that work in harmony to provide a comprehensive intelligence layer, creating a single point of entry for all developer queries.

COMPONENT	ROLE	PRIMARY FUNCTION
kbsearch-mcp-server	Central Hub	Orchestrates tool registration and editor integration across VS Code, Cursor, and IntelliJ.
rag-mcp	Knowledge Expert	Performs high-performance RAG-based search across internal documentation and technical wikis.
nl2sql-mcp	Data Analyst	Converts natural language questions into optimized SQL queries for CI/CD metrics and history.

KBSEARCH-MCP-SERVER ORCHESTRATES THE AI WORKFLOW

TOOL REGISTRATION

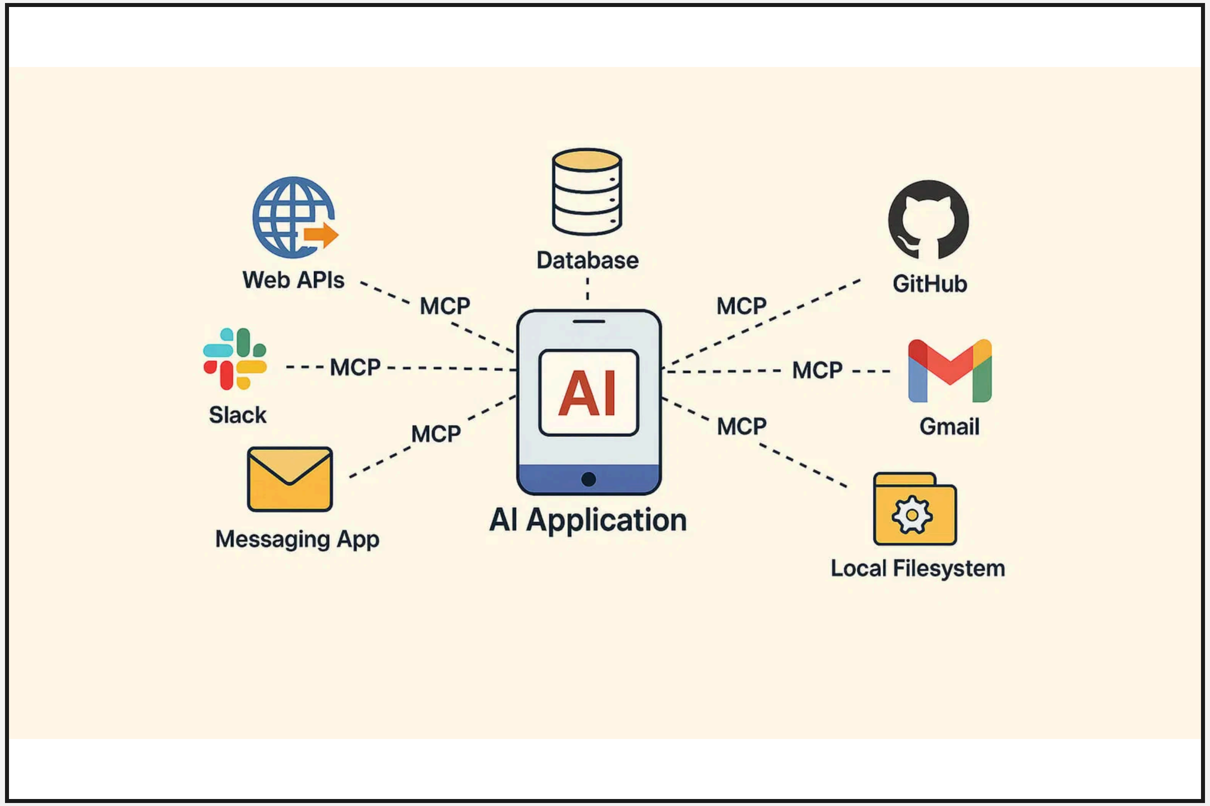
Acts as the primary interface for the AI assistant, handling the registration of specialized tools like search_knowledge_base and query_cicd.

MULTI-EDITOR SUPPORT

Provides a consistent experience across various IDEs including VS Code, Cursor, and IntelliJ through standardized MCP endpoints.

SEAMLESS WORKFLOW

Simplifies the integration of complex backend services into the developer's chat interface, ensuring a single point of entry for all queries.



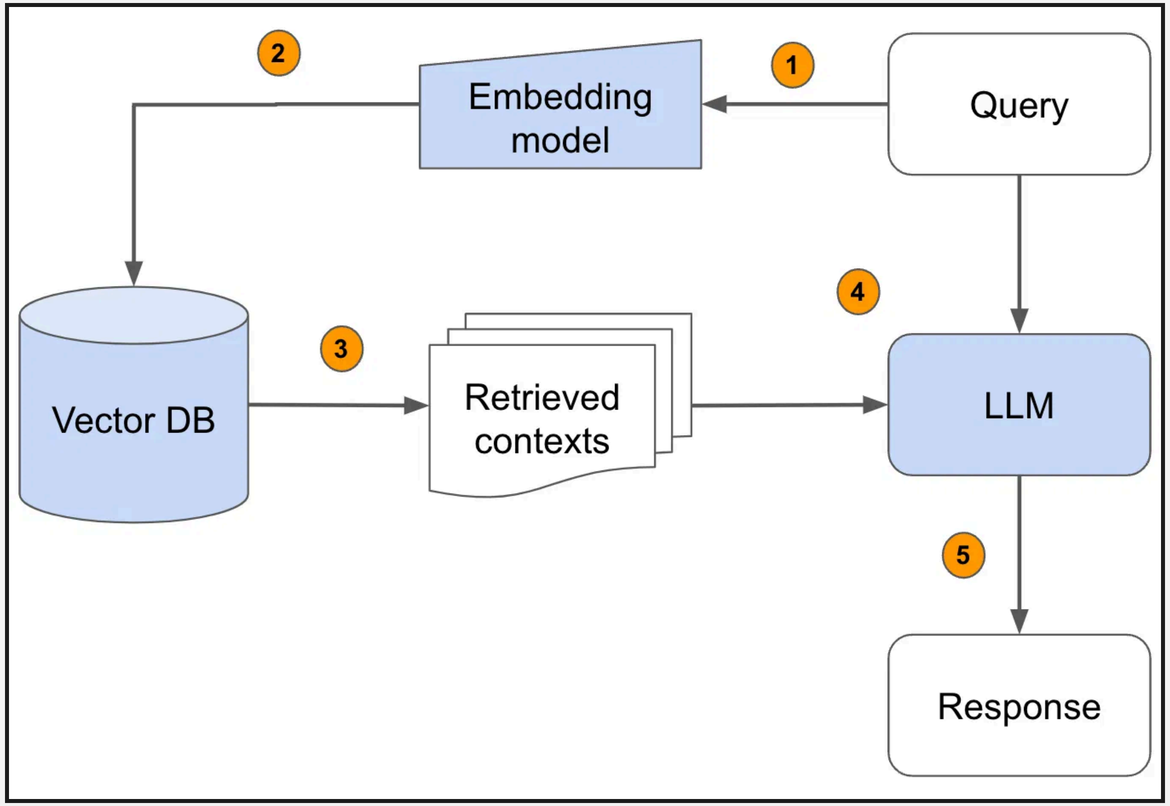
RAG-MCP DELIVERS INTELLIGENT KNOWLEDGE RETRIEVAL

Interactive Knowledge Base

The rag-mcp service transforms static documentation into an interactive resource. By automating the ingestion of technical content from URLs, it ensures that the AI has the most up-to-date context available.

Instant Technical Answers

Eliminate manual searching through wikis and READMEs. Developers can ask "How-to" and "What-is" questions directly in their IDE, receiving precise answers grounded in their own documentation.



ADVANCED RAG ARCHITECTURE SUPPORTS DIVERSE NEEDS

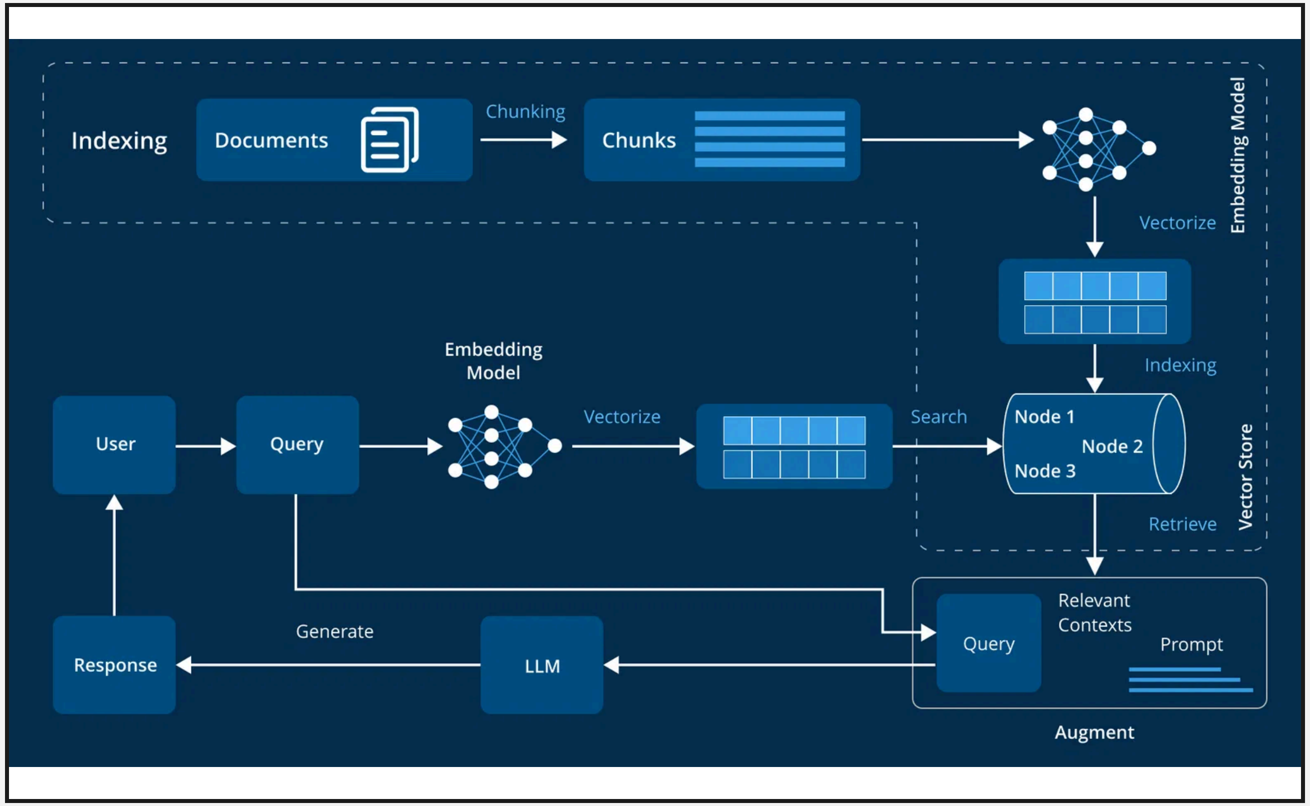


Figure 1: End-to-end RAG pipeline from ingestion to contextual response.

INGESTION PIPELINE

Automated Scrapers: Extracts content from technical documentation and wikis.

Semantic Chunking: Intelligent boundary detection for better context retention.

CONFIGURATION PROFILES

Versioned Settings: Manage embedding models and retrieval parameters.

Optimization: Tailor strategies for accuracy, cost, or latency.

PROVIDER FLEXIBILITY

Hybrid Support: Local (Ollama) and Cloud (OpenAI, Gemini, Bedrock).

PostgreSQL **pgvector** **FastAPI** **Ollama**

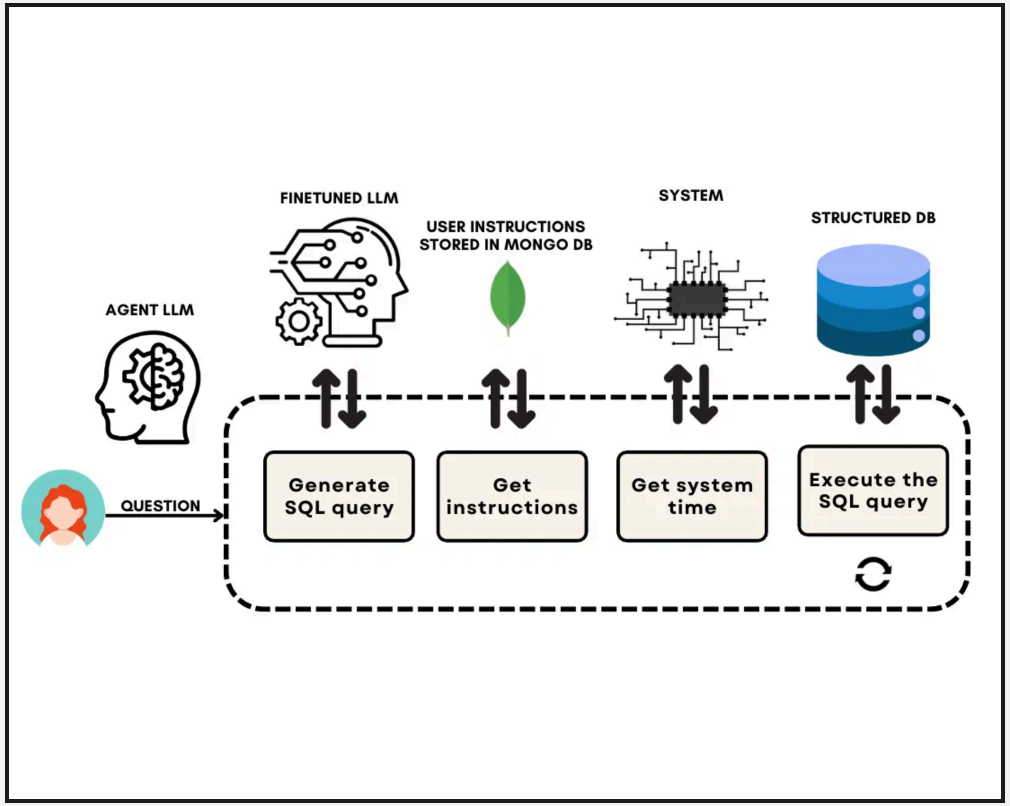
NL2SQL-MCP SIMPLIFIES COMPLEX DATABASE QUERYING

The Challenge: Manual SQL Complexity

Querying CI/CD databases for deployment metrics often requires writing complex SQL that is both time-consuming and error-prone for developers focused on delivery.

The Solution: Natural Language Interface

1. **Prepare:** AI identifies intent (e.g., "last 5 deployments") and extracts slots like app name and environment.
2. **Execute:** The system generates optimized SQL based on the schema and returns formatted results instantly.



INTELLIGENT CACHING DRIVES HIGH-PERFORMANCE DATA ACCESS

STEP 1

Prepare Phase

The system identifies user intent and extracts parameters (slots). It checks the intelligent cache for existing query patterns to avoid redundant LLM calls.

STEP 2

Execute Phase

If a cache miss occurs, optimized SQL is generated based on the schema. Verified results are then returned and stored in the cache for future instant access.

PERFORMANCE METRIC	CACHE HIT	CACHE MISS	ACCURACY
Response Time	**~50ms**	2 - 5 seconds	100% (Verified)
LLM Cost	**$0.00**	Minimal (Generation)	High (Schema-Aware)

STRATEGIC OPTIMIZATION FOR ACCURACY, COST, AND SPEED

ACCURACY

Prioritize answer quality and technical depth for complex engineering queries.

Embeddings	3072-dim
Chunking	Semantic
LLM	GPT-4o / Pro

COST

Minimize operational expenses while maintaining acceptable quality levels.

Embeddings	Local (768d)
Chunking	Fixed (250)
LLM	Ollama / Flash

SPEED

Optimize for the fastest possible response times in high-frequency environments.

Search	Pure Vector
Context	Small (Top 3)
LLM	Flash Models

REAL-TIME METRICS TRACKING: LATENCY, COST, AND USER SATISFACTION (0-10) ARE MONITORED PER PROFILE TO ENSURE CONTINUOUS IMPROVEMENT.

REAL-WORLD USE CASES ENHANCE DAILY PRODUCTIVITY

SCENARIO A

Rapid Developer Onboarding

A new engineer asks: "What is our AKS deployment strategy?" Instead of searching through fragmented wikis, the AI provides the exact documentation instantly.

Primary Tool:
rag-mcp Knowledge Search

SCENARIO B

Incident Failure Analysis

A developer needs to know: "Show me the last 5 failures for the API gateway." The AI queries the CI/CD database and returns formatted results in seconds.

Primary Tool:
nl2sql-mcp Data Query

SCENARIO C

Root Cause Investigation

Combining both tools: Find the documentation for a failing service and then query its recent deployment history—all without leaving the chat interface.

Primary Tool:
Unified MCP Ecosystem

SEAMLESS DEPLOYMENT AND SCALABLE INTEGRATION

DOCKER-FIRST DEPLOYMENT

Containerized Architecture

All components—the MCP hub, RAG service, and NL2SQL tool—are fully containerized. This ensures consistent environments from local development to production clusters.

Enterprise Scalability

Start with local models (Ollama) for privacy and cost-efficiency, then scale to cloud providers (OpenAI, Gemini, Bedrock) as your team's needs grow.

IDE INTEGRATION

Supported Environments

VS Code: Simple configuration via the `.mcp.json` file in your workspace root.

Cursor: Native, built-in support for MCP servers with zero-config detection.

IntelliJ: Compatible via AI assistant plugins like Continue or specialized MCP extensions.

Integration is standardized across editors, providing a unified chat interface for all your documentation and data queries.

FUTURE-PROOFING DEVELOPMENT WITH OPEN STANDARDS

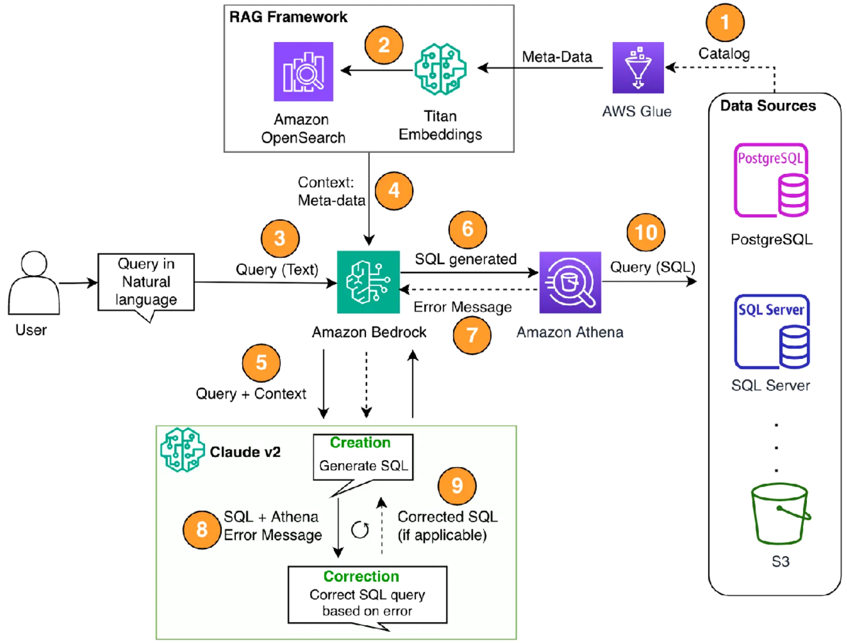
Unified Intelligence: Combine internal documentation and live operational data in a single interface.

Developer Productivity: Drastically reduce context switching and manual information retrieval.

Open Ecosystem: Built on the Model Context Protocol for long-term compatibility with evolving AI models.

GET STARTED TODAY

Clone the repositories, configure your local MCP server, and transform your development workflow with intelligent, context-aware tools.



THE FUTURE OF AI-ASSISTED DEVELOPMENT