MCP SERVER | NO CODE | CLOUD HOSTED

Open WebUI MCP

Manage Models, Knowledge, and Chats from Anywhere

Open WebUI gives your AI agent full control over local and cloud large language models. List available models, manage knowledge bases by uploading files or processing websites, and run controlled chat sessions—all through natural conversation.

A+ Quality Score 100/100

llm-management | rag | model-inference | self-hosted | chat-interface | automation

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JUNE 2026
vinkius.com/mcp/open-webui

SCAN TO VIEW

The connectivity layer between AI and the world's software.

Vinkius sits between AI and every application. All communication passes through Vinkius Cloud via the Model Context Protocol (MCP) — with governance, observability, and security at every layer.

Your AI Connections Run Through Vinkius Cloud

The world's largest managed MCP catalog

Vinkius is the connectivity layer where AI connects to the software your business already runs. We handle the hosting, the security, the credentials, the uptime — you get agents that actually do things.

We operate the world's largest managed MCP catalog. Major SaaS platforms, CRMs, databases, and cloud providers — running, monitored, production-ready. This MCP server is hosted and maintained by the Vinkius Cloud for AI Agents.

> *The agent doesn't manage credentials, doesn't manage uptime, doesn't manage security. Vinkius does.*
>
> — Architecture principle

Four Pillars of the Vinkius Runtime

01 — Security by design

Credentials stay encrypted at rest via AES-256. The AI agent never touches raw keys — they're injected into a sandboxed V8 isolate at runtime. Actions are logged, and connections have an emergency kill switch.

02 — Built on MCP Fusion

This MCP server was built with **MCP Fusion**, the open-source framework (Apache 2.0) that powers the entire Vinkius catalog. Schema-as-firewall strips undeclared fields, compiled PII redaction runs at zero overhead, and cryptographic lockfiles produce git-diffable audit trails.

03 — Deterministic observability

Eight immutable metrics per endpoint: request volume, p95 latency, error rate, active connections, cost attribution. A live payload feed logs every tool call with mutation detection.

04 — Autonomous operations

Servers are deployed, monitored, and patched autonomously. New capabilities and security patches ship weekly. Zero-downtime deployments ensure continuous availability across all managed MCP servers.

AES-256	Ed25519	24h TTL	V8 Isolate
Encryption at rest	PKI vault signatures	Ephemeral session keys	Sandboxed execution

One Token. Instant Access.

Every MCP server on Vinkius is accessed through a **Connection Token**. Tokens are generated in the cloud dashboard and produce a unique MCP endpoint URL. Paste this URL into any MCP-compatible client — no SDK required.

A single token can serve **multiple AI clients simultaneously**, or you can issue separate tokens per client for granular access control. Each token tracks its own request count, last activity timestamp, and can be individually enabled or revoked.

MCP ENDPOINT

```
https://edge.vinkius.com/{token}/mcp
```

Claude · Cursor · VS Code · Windsurf · Grok · Gemini

Security Is the Architecture

Security in Vinkius is not a feature — it's the foundation of the runtime. The gateway enforces multiple independent protection layers between AI agents and third-party APIs.

01 — Ed25519 PKI Vault

Every workspace has an Ed25519 Master Key. Session keys are generated ephemerally (24h TTL) and signed by the Master Key. Credentials never leave the vault boundary.

02 — V8 Isolate Sandboxing

Tool code runs inside isolated-vm V8 isolates with 64 MB memory caps and per-request timeouts. No filesystem access, no network access except through the SSRF-guarded fetch bridge.

03 — SSRF Guard

All outbound HTTP requests are DNS-resolved and validated before execution. Private IP ranges (10.x, 172.16-31.x, 192.168.x, AWS metadata 169.254.x) are blocked at the network layer.

04 — DLP & PII Redaction

A ResponseGuard pipeline intercepts every tool response. Configurable redaction patterns strip sensitive fields (emails, SSNs, card numbers) before data reaches the AI agent.

05 — Cryptographic Audit Trail

Every request is signed into a SHA-256 hash chain with Ed25519 signatures. Events form a tamper-proof, SIEM-exportable forensic record.

06 — Honeytoken Trap System

Phantom credentials are injected into isolated environments. If a honeytoken is used outside Vinkius infrastructure, the server is quarantined instantly.

Emergency Kill Switch

EU AI Act Art. 14(1) Compliant

The kill switch is an **emergency halt** mechanism — not a simple toggle. When triggered, it executes three actions atomically:

01 — Server deactivated

The MCP server is immediately taken offline across the entire cluster.

02 — All tokens revoked

Every connection token is invalidated. Total lockout — reconnection blocked until new tokens are issued.

03 — WebSocket connections killed

Active connections terminated via Redis pubsub broadcast. Propagates to every runtime node in the cluster.

Full Visibility. Zero Guesswork.

The Vinkius cloud dashboard includes a full MCP Governance suite — real-time analytics and security controls for production AI operations.

Control Plane

KPI dashboard with request volume, latency, success rate, token consumption, and AI-generated operational briefings.

Agent Activity

Which AI clients are connecting, how often, and what they're doing — real-time session tracking.

FinOps

Cost tracking per tool, payload compression savings, budget optimization signals, and consumption trends.

Tool Health

Slowest and most error-prone tools, with actionable root-cause insights and performance baselines.

Firewall & DLP

PII redaction activity, sensitive data protection counters, and security event timeline.

Incident Log

Error trends, failure rates, status-code breakdowns, and forensic audit trail access.

> Get started at **cloud.vinkius.com** — connect your AI agent in under 60 seconds.

Open WebUI MCP

12 tools available
Cloud-hosted on Vinkius

This MCP connects your Open WebUI instance to any AI client, letting you handle complex LLM tasks without needing the command line. Instead of jumping between model APIs and document storage systems, you tell your agent what you need done. You can check which models are available (Ollama, OpenAI, etc.), upload documents or paste web URLs to build a knowledge collection, and then use those resources for chat completions. It's about treating your LLM infrastructure like another service endpoint. If you're building an internal toolchain, Vinkius makes it simple to connect this control layer to any agent in the catalog, giving you comprehensive oversight of both local and cloud model performance.

Core Capabilities

01 — Inventory Available Models

Retrieve a list of all connected language models, including those running locally via Ollama.

02 — Build Knowledge Collections

Ingest new information by uploading files or processing web URLs and organizing them into searchable collections for RAG context.

03 — Manage Chat Sessions

Start, manage, and complete controlled chat conversations using standard OpenAI/Anthropic compatible endpoints.

04 — Execute Local Inference Tasks

Directly interact with the Ollama API to generate completions or create embeddings for local model testing.

05 — Monitor Data Status

Check the status of uploaded documents and web content to confirm when they are ready for use in your knowledge base.

One Click on Vinkius — From Prompt to Execution

Available at **vinkius.com/mcp/open-webui** — connect your AI agent in three steps.

01 Subscribe to this MCP on Vinkius and provide your Open WebUI Base URL and API Key.

02 Your AI client connects, giving it the necessary permissions to read model lists and manage files in your Open WebUI backend.

03 You request a specific action—like adding a document or starting a chat—and the agent executes it through the MCP's exposed tools.

The bottom line is you gain a single, conversational interface to manage complex model and data pipelines.

Built For

AI Engineers who need to test dozens of model combinations. Knowledge Managers stuck manually feeding documentation into chat bots. DevOps teams monitoring local LLM deployments.

AI Engineer

Runs tests comparing different models, managing the lifecycle of RAG collections, and debugging complex prompt chains without leaving their IDE.

Knowledge Manager

Needs to quickly ingest corporate documentation or large sets of web articles into a centralized knowledge base for team-wide Q&A.

DevOps Engineer

Monitors the availability and performance of self-hosted Ollama instances, ensuring model tags are correct across development environments.

What Changes When You Connect

01 Gain full model visibility by using `list_models` to see every available LLM endpoint, whether it's running on Ollama or a cloud provider.

02 Build powerful knowledge bases by letting the agent process web content via `process_web_url`, automatically indexing external data into your collections.

03 Run controlled chat sessions and full conversation flows using `create_new_chat` to ensure proper history tracking and context management.

04 Test local inference directly with Ollama tools. You can use `ollama_generate` or `ollama_embed` to validate local model performance instantly.

05 Keep your RAG pipelines moving by checking file readiness using `get_file_status`, ensuring no time is wasted waiting on document ingestion.

Real-World Applications

Updating the Company Handbook

A Knowledge Manager needs to update internal FAQs. Instead of manually downloading PDFs, they simply ask their agent to process a batch of new web URLs and use `add_file_to_collection` to add them all to the 'HR Policies' knowledge base.

Benchmarking Local LLMs

An AI Engineer needs to compare Llama 3 vs. Mistral on a specific query. They use `ollama_generate` for both models side-by-side, allowing them to programmatically benchmark performance without manual API calls.

Handling Live Customer Feedback

A support team wants to analyze recent blog posts. They ask the agent to process a live URL and then use `chat_completions` against that newly indexed data, getting instant insights into customer pain points.

Debugging Chat Flows

A developer needs to ensure their complex multi-step chat flow works. They use `create_new_chat` and monitor the session via `send_message` to verify that context is passed correctly between steps.

Patterns to Avoid

Assuming model availability

✗ AVOID

The user tries to generate content using a specific model name, but doesn't know if it was properly loaded or tagged in the local Ollama environment.

✓ INSTEAD

First, run `list_models` and then check available tags using `ollama_tags`. This confirms your agent has access to the correct endpoint before attempting generation via `ollama_generate`.

Treating files as instant context

✗ AVOID

The user uploads a massive document and immediately asks questions about it, but the system hasn't finished indexing or processing the file.

✓ INSTEAD

After uploading content with `upload_file`, always check the ingestion status first. Use `get_file_status` to confirm the data is fully processed before querying.

Mixing API standards

✗ AVOID

Trying to mix proprietary chat endpoints with standard OpenAI calls without proper context management.

✓ INSTEAD

When working on structured conversations, always start a fresh session using `create_new_chat`. This ensures the agent manages the message history correctly, regardless of whether you use `send_message` or `chat_completions`.

The Right Fit

Use this MCP if your primary job involves managing and connecting multiple LLM sources—local Ollama instances alongside cloud endpoints like OpenAI or Anthropic. It's the control layer for complex RAG systems, making sure data gets indexed before it's asked about. Don't use this if you just need a single chat interface; then, a basic messaging MCP is enough. Also, don't use it if your primary task is simply writing code—you'll want an IDE-focused tool instead. This MCP excels when the job requires coordinating data ingestion (files/URLs), model selection (`list_models`), and structured output generation.

Dealing with disconnected LLM services is exhausting.

Today, managing a knowledge base means jumping through hoops. You upload a PDF to one service, process a website URL in another, and then you have to manually verify the status of each piece before you can even begin asking questions using your chat bot. It's copy-pasting URLs into dedicated ingestion tabs, waiting for separate background jobs to finish, and constantly switching between model API dashboards.

With this MCP, that manual process collapses into a single conversation thread. Your agent handles the messy backend work—from uploading files with `upload_file` to processing web content via `process_web_url`. You just tell it: 'Use all of this data,' and you get actionable answers instantly.

Open WebUI MCP: Centralized Model & Data Control

You eliminate the need to manually track which models are available or if your document ingestion jobs actually finished. Instead of checking multiple dashboards for model status, you use `list_models` and then validate content readiness with `get_file_status`—all in one flow.

The difference is control. You stop reacting to siloed system alerts and start giving direct, guided commands to manage your entire LLM stack.

Open WebUI: 12 Tools for LLM Management

These tools give you granular control over model listing, file uploading, web scraping, and structured chat management within Open WebUI.

#	TOOL	DESCRIPTION
01	`add_file_to_collection`	Adds a specified file to an existing knowledge collection so it can be used for retrieval.
02	`chat_completed`	Runs specific filters or processing steps after a chat conversation has finished.
03	`chat_completions`	Generates responses based on prompts using an OpenAI-compatible standard chat completion endpoint.
04	`create_new_chat`	Initiates a new, structured chat session that is controlled by the backend flow for tracking and history.
05	`get_file_status`	Checks the current status of file processing to ensure documents are indexed and ready for querying.
06	`list_models`	Retrieves a list of all available language models configured in your Open WebUI instance.
07	`ollama_embed`	Generates vector embeddings for text using the local Ollama API embedding function.
08	`ollama_generate`	Requests a completion response directly from a specified model running on the local Ollama API.
09	`ollama_tags`	Lists all models currently available and tagged within your local Ollama environment.
10	`process_web_url`	Scrapes content from a given web URL, extracts the text, and indexes it into a knowledge collection.
11	`send_message`	Generates messages using an Anthropic-compatible message generation standard.
12	`upload_file`	Uploads a file, extracts its content, and stores it in the vector database for future retrieval.

See It in Action

Real prompts you can use once this MCP is connected to your AI agent through Vinkius Cloud.

U List all models available in my Open WebUI instance.

I've retrieved the models. You have access to 'llama3:latest', 'gpt-4o', and several custom Open WebUI functions. Would you like to use one of them for a chat completion?

U Process the URL 'https://docs.openwebui.com/' into my 'Documentation' collection.

I have started processing the URL. The content is being scraped and indexed into the 'Documentation' collection. You can now ask questions based on this data.

U Generate a response using the 'llama3' model for the prompt 'Explain quantum computing'.

Using the `ollama_generate` tool with 'llama3': Quantum computing is a type of computing that uses quantum-mechanical phenomena... Would you like more details?

Frequently Asked Questions

01 **How do I check if my uploaded files are ready for use with Open WebUI MCP?**

You use the `get_file_status` tool. This function checks the processing status of your documents, letting you know exactly when the data is fully indexed and available for retrieval.

02 **Can I list models running on Ollama using Open WebUI MCP?**

Yes, use `ollama_tags` to retrieve a list of all currently tagged and available models in your local Ollama environment. This confirms which specific models you can generate completions with.

03 **Is this MCP only for OpenAI-style chats?**

No, it supports multiple standards. You can use `chat_completions` for OpenAI compatibility or `send_message` if your workflow requires Anthropic's specific message generation format.

04 **What is the best way to get new knowledge into my collection using Open WebUI MCP?**

For web content, use `process_web_url`. If you have physical files like PDFs or TXT documents, it's better to use `upload_file` first.

05 **How do I start a structured conversation flow with Open WebUI MCP?**

Use the `create_new_chat` tool. This initiates a backend-controlled chat session, which is ideal for multi-step processes where history and context need strict management.

Go Live in 60 Seconds

Get your connection token from **cloud.vinkius.com**, then paste the endpoint URL into any MCP-compatible client.

YOUR MCP ENDPOINT

```
https://edge.vinkius.com/[TOKEN]/mcp
```

CLIENT	WHERE TO CONFIGURE
Claude AI	Profile → Customize → Connectors → "+" → Add custom connector → Paste endpoint
Cursor	Settings → Features → MCP Servers → "+ Add New MCP Server" → Type: SSE → Paste endpoint
VS Code	Ctrl/Cmd+Shift+P → "MCP: Add Server" → add `"open-webui": { "url": "…" }`
Windsurf	MCP Settings → `mcp_settings.json` → Add endpoint URL
ChatGPT	Settings → Tools & plugins → Add MCP server → Paste endpoint
Gemini	Extensions → Add MCP Server → Paste endpoint URL

ASK AN AI ABOUT THIS

Let your preferred AI explain this MCP server

- Ask ChatGPT
- Ask Claude
- Ask Perplexity
- Ask Gemini
- Ask Grok

READY TO CONNECT

Open WebUI is live on Vinkius Cloud.

Get your connection token, paste it into your AI agent, and start building. No SDK. No deployment. Just results.

Start at cloud.vinkius.com →

vinkius.com · support@vinkius.com

INDEPENDENT PLATFORM DISCLAIMER

Vinkius is an independent platform and is not affiliated with, endorsed by, sponsored by, verified by, or otherwise authorized by Open WebUI. All third-party trademarks, logos, and brand names are the property of their respective owners. Their use in this document is strictly for informational purposes to identify service compatibility and interoperability.

DOCUMENT INFORMATION

Generated	June 2026
MCP Server	Open WebUI MCP
Server ID	`019e38cc-df7c-7157-9766-18c9eb569b67`
Platform	Vinkius Cloud for AI Agents
Endpoint	`https://edge.vinkius.com/{token}/mcp`

LICENSE & USAGE

This document is generated automatically by the Vinkius PDF Engine. Content reflects the MCP server configuration at the time of generation and may change as updates are deployed. For the most current information, visit vinkius.com/mcp/open-webui.