Architectural Orchestration of Local Multimodal AI Stacks: A Comparative Study of Integrated Environments for Windows 11 Systems

The shift toward localized artificial intelligence signifies a critical transition in computational autonomy, where the imperatives of data privacy, cost efficiency, and operational resilience converge. For professionals operating on Windows 11 architectures with substantial memory capacity yet limited graphical acceleration—specifically 64GB RAM CPU-only systems—the challenge lies in orchestrating a multi-app stack that maintains high utility without the benefit of massive parallelization afforded by high-end GPUs. This report evaluates the most effective methodologies for deploying integrated environments utilizing Ollama, Open WebUI, and Model Context Protocol (MCP) servers, specifically configured for secondary drive storage on an 'L' drive.

Structural Foundation: The Ollama-Open WebUI-MCP Nexus

At the core of a modern local AI ecosystem is the decoupling of the inference engine from the user interface. Ollama functions as the primary back-end orchestrator, managing the lifecycle of large language models (LLMs) through a streamlined service that exposes an OpenAI-compatible API on port 11434. This background service handles the nuances of model loading and memory optimization, which is particularly vital for CPU-only systems where memory bandwidth becomes the primary bottleneck for token throughput. Open WebUI serves as the primary gateway for user interaction, providing a feature-rich, self-hosted environment that replicates the conversational fluidity of cloud-based assistants while facilitating deep extensibility through its plugin architecture.
The Model Context Protocol (MCP), a standardized open standard introduced to unify the interaction between AI models and external data sources, represents the third pillar of this architecture. MCP effectively acts as a universal bridge, allowing LLMs to execute tools, query databases, and perform web research through a standardized JSON-based protocol. On Windows 11, the integration of MCP servers into Open WebUI typically requires an intermediary proxy known as mcpo (Model Context Protocol Orchestrator), which translates stdio-based MCP communication into the OpenAPI-compatible format required by the WebUI front-end.

Comparative Analysis of Top 5 Local AI Project Guides

The following analysis identifies and compares five verified guides that provide robust paths toward an integrated multimodal stack. These guides are selected based on their community validation, technical depth, and compatibility with the specific hardware and storage requirements of professional Windows users.

The LobeHub mcpo Orchestration Framework

The LobeHub methodology is highly regarded for its focus on agentic research and browser control. It leverages the mcpo proxy to create a secure bridge for a wide array of community-developed MCP servers, including Microsoft's Playwright for browser automation and Firecrawl for advanced web scraping. This guide is particularly suited for users who require the AI to navigate complex web environments or interact with real-time data sources.

Feature Category	Implementation Strategy	Key Components
Core Backend	Host-resident Ollama	Ollama Windows Setup
UI Layer	Dockerized Open WebUI	ghcr.io/open-webui/open-webui:ollama
Integration Proxy	mcpo Bridge	ghcr.io/open-webui/mcpo:main
Web Control	Playwright MCP	@modelcontextprotocol/server-playwright
Content Analysis	Firecrawl MCP	@mendable/firecrawl-mcp-server

The LobeHub approach emphasizes the "Native" function-calling mode in Open WebUI, which utilizes the inherent tool-calling capabilities of models like DeepSeek-R1-7B or Qwen3-7B to interact directly with the defined MCP endpoints. This results in a highly autonomous system capable of multi-step research tasks.

The ahmad-act Enterprise Integrated Stack

The ahmad-act repository provides a more monolithic Docker Compose strategy, which is optimized for stability and ease of initial deployment. It integrates specific business-process MCP tools directly into the stack, such as leave management and personalized greetings, serving as a template for enterprise tool development. This guide is essential for users looking for a pre-configured environment where the connection between the UI and the MCP servers is handled within a single network bridge.

This guide provides explicit instructions for the host.docker.internal gateway, which is a common stumbling block for Windows users attempting to connect Docker-based interfaces to Windows-native Ollama services. For a CPU-only system, this guide recommends 4-bit quantized versions of the DeepSeek-R1-7B model, which offers high reasoning capabilities while maintaining a memory footprint that allows for parallel execution of other system tasks.

The Savantskie Persistent AI Memory System

Context retention is often the weakest link in localized AI. The Savantskie guide addresses this through a dedicated Persistent AI Memory System, which implements a hierarchical embedding service. This system allows the AI to store long-term context in a local database (SQLite), retrieving relevant memories via semantic search when the current conversation requires historical insights.

Memory Feature	Implementation Mechanism	System Impact
Persistent Storage	SQLite/JSON	Minimal RAM, High Disk Utility
Embedding Engine	Ollama Fallback (qwen2.5:1.5b)	High Performance on CPU
Search Method	Semantic Vector Similarity	Low Latency Retrieval
Relationship Mapping	Intelligent Linking	Improved Contextual Nuance

This guide is uniquely valuable for the target 64GB system as it uses lightweight models like Qwen2.5-1.5B for embedding generation, ensuring that the primary reasoning model (e.g., a 7B parameter variant) has full access to the system's computational resources.

The Joshua-Hub mcp_things2 Developer Suite

The Joshua-Hub suite is designed for power users who require observability and secure code execution. This guide bundles mcpo with a series of development-focused MCP servers, including sandboxed Python execution environments and SDXL image generation APIs. It is one of the few guides that integrates Prometheus and Grafana for real-time monitoring of inference metrics and system health.
The secure execution environment is particularly critical for database management, as it allows the AI to run complex data analysis scripts within an isolated container, protecting the host system from potential code injection vulnerabilities while providing the AI with the ability to manipulate structured data directly.

The Rancher Desktop Managed Extension Stack

For users who prefer a graphical management interface over command-line Docker interactions, the Rancher Desktop guide provides a streamlined path via its Extension Catalog. This method installs Open WebUI, Ollama, and SearXNG as managed extensions, providing native-like integration with the Windows host.
The Rancher approach is highly effective for CPU-only systems as it provides better visibility into the resource allocation between the host and the virtualized containers. It also includes bundled MCP servers for Docker and Kubernetes management, allowing the user to control their local development infrastructure through natural language prompts in the WebUI.

Detailed Storage Optimization: The 'L' Drive Paradigm

A primary requirement for professional AI deployment on Windows 11 is the relocation of heavy model weights and container data to a dedicated high-capacity drive, designated here as the 'L' drive. Standard installations of Ollama and Docker default to the system drive ('C'), which can lead to rapid storage depletion and performance degradation during disk-intensive operations.

Ollama Model Relocation

Ollama's model storage is governed by the OLLAMA_MODELS environment variable. To redirect this to the 'L' drive, the following procedure is community-validated:

1. Terminate the Ollama service from the system tray.
2. Access the Windows "Edit the system environment variables" menu via sysdm.cpl.
3. Under "User variables," define a new variable: OLLAMA_MODELS with the value L:\Lily\Ollama\models.
4. Move any existing models from %USERPROFILE%\.ollama\models to the new path to avoid redownloading.

Docker and Open WebUI Data Persistence

For Docker-based deployments, moving the underlying storage is essential. This can be achieved through a directory junction (symbolic link) or by reconfiguring the Docker Desktop "Disk image location" in the settings. A junction is often more reliable for legacy scripts:

```
# Stop Docker Desktop
# Move %LOCALAPPDATA%\Docker to L:\Lily\Docker
```

```
mklink /j "C:\Users\<User>\AppData\Local\Docker" "L:\Lily\Docker"
# Restart Docker Desktop
```

When running the Open WebUI container, the volume mapping must be explicitly set to the 'L' drive to ensure all chat logs, RAG indices, and user data are stored on the dedicated partition: -v L:\Lily\OpenWebUI\data:/app/backend/data.

Multimodal Capability Integration

The target stack must support multimodal 7B models, voice chat, RAG, and screen sharing. These capabilities are achieved through the orchestration of specific plugins and hardware workarounds.

Multimodal Support and Vision Models

For a CPU-only system, selecting the right multimodal model is a balancing act between visual accuracy and inference speed. LLaVA (Large Language-and-Vision Assistant) 1.6 7B is the community standard for general vision tasks, while Moondream 2 is a lightweight alternative that offers significantly higher frame rates on CPU hardware.

Vision Model	Parameter Count	Quantization	Best Use Case
LLaVA 1.6	7B	Q4_K_M	Complex image analysis
Moondream 2	1.6B	Q8_0	Fast screen reading
MiniCPM-V	8B	Q4_K_M	OCR and high-res vision
Qwen2-VL	7B	Q4_K_M	Precise document understanding

In a 64GB RAM environment, these models can reside comfortably in memory alongside a dedicated 7B reasoning model (like DeepSeek-R1-7B), allowing the system to switch between visual analysis and deep reasoning without reloading the weights.

Implementation of Live Voice Chat

Hands-free voice chat requires a chain of three distinct services: Speech-to-Text (STT), the LLM reasoning loop, and Text-to-Speech (TTS). Open WebUI facilitates this through its Audio settings tab.

- **STT**: Utilizing the local Whisper engine (specifically the base or small model) ensures high-quality transcription with minimal CPU latency.
- **TTS**: For the most responsive experience, the "Browser Kokoro" or "Web API" engines are recommended, as they utilize the client's browser for speech synthesis, offloading the task from the host's CPU.
- **Security Requirements**: Modern browsers (Chrome/Edge) restrict microphone access to secure origins. Users must serve Open WebUI over HTTPS, which can be accomplished locally using a reverse proxy or by whitelisting the local IP in browser flags.

Screen Sharing via OBS Virtual Camera

Because local web interfaces cannot natively "capture" the Windows desktop for security reasons, the OBS Virtual Camera serves as a critical bridge. By installing OBS and activating the Virtual Camera, a user can designate their screen or a specific application window as a "webcam" source. In Open WebUI, the user initiates a video chat with a vision model and selects the OBS Virtual Camera as the input, allowing the AI to "see" and interact with the user's desktop in real-time.

Advanced Knowledge Management: RAG and Database Integration

The Retrieval-Augmented Generation (RAG) system in Open WebUI allows for the ingestion of local documents and web content into the model's context window. This is managed by chunking files and generating vector embeddings, which are stored in an internal vector database (ChromaDB).

RAG and Web Research Strategies

Open WebUI provides two primary methods for real-time research:

1. **Direct URL Ingestion**: By prefixing a prompt with # followed by a URL, the system scrapes the page content and injects it into the prompt.
2. **Web Search Plugins**: Configuring providers like SearXNG, Google PSE, or DuckDuckGo in the settings allows the model to perform autonomous searches before responding.

For a CPU-only system, the performance of the RAG system is highly dependent on the embedding model. Using nomic-embed-text is highly recommended, as it is optimized for local execution and maintains high semantic accuracy with minimal overhead.

Database and Notion Integration via MCP

Structured data management is a primary use case for MCP servers. The Notion MCP integration allows the AI to read and write to Notion databases, providing a seamless way to maintain project trackers or personal wikis. For local SQL databases, the Joshua-Hub and ahmad-act guides demonstrate how to expose SQLite or MySQL tables to the AI through FastMCP servers. This allows users to perform complex data analysis using natural language: "Analyze the last six months of project expenses in my SQL database and identify any anomalies."

Hardware Optimization and Performance Analysis

Operating an AI stack on a 64GB CPU-only system requires a shift from "speed-first" to "intelligence-first" optimization. While a GPU might produce 50-100 tokens per second (t/s), a high-end CPU (such as a Ryzen 9 or Intel Core i9) typically yields 3-8 t/s for 7B models.

Token Throughput and Memory Bandwidth

The inference speed on a CPU is mathematically limited by the memory bandwidth. For a 7B model at 4-bit quantization, each token generation requires the CPU to read approximately 5GB of weights from the RAM. The relationship can be modeled as:
On a system with DDR4-3200 memory (approx. 25.6 GB/s per channel), a 7B model (5GB) can theoretically reach 5-10 t/s, but this is often reduced by overhead from the operating system and other running containers.

Quantization and Reliability

Quantization is the process of compressing model weights from higher precision (FP16) to lower precision (4-bit or 5-bit). For 64GB systems, the goal is not to fit the model into memory (as 64GB is plenty for even a 30B model), but to minimize the amount of data the CPU must read for each token.

Model Class	Quantization	RAM Usage	Intelligence Loss	Use Case
7B	Q4_K_M	~5.0 GB	Negligible	Daily Assistant
7B	Q8_0	~7.5 GB	Near-Zero	High-precision coding
14B	Q4_K_M	~9.5 GB	Negligible	Complex Reasoning
32B	Q4_K_M	~19.0 GB	Minor	Science/Math
70B	Q4_K_M	~42.0 GB	Moderate	Benchmarking

The Q4_K_M quantization is widely regarded as the "sweet spot" for 7B models, providing a dramatic reduction in size with nearly imperceptible loss in quality.

Technical Roadmap for Deployment

For a professional user to successfully deploy this stack on Windows 11 with the specified constraints, the following implementation roadmap is recommended.

Environment Preparation (L:\Lily)

The first step is establishing the directory structure and environment variables to ensure all data resides on the 'L' drive.

```
# Create directories
New-Item -Path "L:\Lily\Ollama\models" -ItemType Directory -Force
New-Item -Path "L:\Lily\OpenWebUI\data" -ItemType Directory -Force
New-Item -Path "L:\Lily\Docker" -ItemType Directory -Force

# Set Environment Variables
::SetEnvironmentVariable('OLLAMA_MODELS', 'L:\Lily\Ollama\models',
'User')
::SetEnvironmentVariable('DATA_DIR', 'L:\Lily\OpenWebUI\data', 'User')
```

Installation and Integration

Following the ahmad-act or Joshua-Hub guides, the user should utilize Docker Compose to launch the stack. The use of the :ollama image tag for Open WebUI is recommended if the user wants an all-in-one container, although a host-resident Ollama is generally faster on Windows due to better direct hardware access.

Feature Configuration

1. **Audio**: In Open WebUI, navigate to Settings > Audio. Set the STT engine to "Whisper" and the TTS engine to "Browser".
2. **Vision**: Configure OBS for Virtual Camera and pull the moondream model via Ollama.

3. **MCP Tools**: Define search and browser control tools in the config.json for mcpo, and add the proxy address (http://host.docker.internal:8000) in the Open WebUI Tools settings.

Conclusion: Strategic Implications of the Local AI Stack

The orchestration of a local AI stack on Windows 11 represents a significant milestone in personal computing, where the capabilities of frontier cloud models are successfully replicated within a private, hardware-constrained environment. For the professional user with 64GB of RAM, the ability to maintain multimodal support, persistent memory, and agentic research capabilities without external dependencies provides a competitive advantage in data security and operational cost.

The primary success factor in such a deployment is not just the hardware, but the meticulous configuration of the interplay between Ollama, Open WebUI, and the Model Context Protocol. As models become more efficient through MoE architectures and improved quantization techniques, the "CPU-only" bottleneck will continue to diminish, allowing for even more complex agentic behaviors to be executed locally. By adhering to a modular architecture and utilizing dedicated secondary storage for model weights, users can ensure their AI infrastructure remains scalable, resilient, and entirely under their control.

Through the strategic use of MCP servers like Playwright for browser control and Savantskie's Persistent Memory for long-term context, the local stack evolves from a simple conversational agent into a comprehensive cognitive assistant. The integration of OBS for screen sharing further bridges the gap between digital content and AI analysis, fulfilling the promise of a truly multimodal, locally-hosted intelligence platform.

Works cited

1. Local AI with OpenWebUI, Ollama & MCP | doubleSlash Blog, https://blog.doubleslash.de/en/software-technologien/kuenstliche-intelligenz/openwebui-ollama-and-mcp-a-local-ki-stack-for-companies/ 2. How to Install Ollama on a Different Drive in Windows - blackMORE Ops, https://www.blackmoreops.com/how-to-install-ollama-on-a-different-drive-in-windows/ 3. open-webui/open-webui: User-friendly AI Interface (Supports Ollama, OpenAI API, ...) - GitHub, https://github.com/open-webui/open-webui 4. Open WebUI + Ollama: Local AI Chat with RAG on Ubuntu Guide - iTecs, https://itecsonline.com/post/openwebui-ollama-rag-guide 5. Tools | Open WebUI, https://docs.openwebui.com/features/extensibility/plugin/tools/ 6. Run Local AI with Open WebUI + Docker Model Runner, https://www.docker.com/blog/open-webui-docker-desktop-model-runner/ 7. Using the Model Context Protocol with Open WebUI | DevCentral - F5, https://community.f5.com/kb/technicalarticles/using-the-model-context-protocol-with-open-webui/344960 8. Open-sourced an MCP Server Quickstart - give AI assistants custom tools : r/LocalLLaMA, https://www.reddit.com/r/LocalLLaMA/comments/1qotgyi/opensourced_an_mcp_server_quickstart_give_ai/ 9. MCP Server Finder Directory | The Ultimate Model Context Protocol Directory, https://www.mcpserverfinder.com/ 10. Open WebUI and MCPO | MCP Servers · LobeHub, https://lobehub.com/mcp/baronco-open-webui-and-mcpo 11. GitHub - ahmad-act/Local-AI-with-Ollama-Open-WebUI-MCP-on ..., https://github.com/ahmad-act/Local-AI-with-Ollama-Open-WebUI-MCP-on-Windows 12. Run AI Locally: The Best LLMs for 8GB, 16GB, 32GB Memory and Beyond - Micro Center, https://www.microcenter.com/site/mc-news/article/best-local-llms-8gb-16gb-32gb-memory-guide.

aspx 13. savantskie/persistent-ai-memory: A persistent local memory ... - GitHub, https://github.com/savantskie/persistent-ai-memory 14. CPU-only, no GPU computers can run all kinds of AI tools locally : r/LocalLLaMA - Reddit, https://www.reddit.com/r/LocalLLaMA/comments/1qxgkd1/cpuonly_no_gpu_computers_can_run_all_kinds_of_ai/ 15. MCP Tools for Open WebUI | Awesome MCP Servers, https://mcpservers.org/servers/joshua-hub/mcp_things2 16. Create a Co-Lab Adviser with OWUI (part 2) - AI, https://ai.colab.duke.edu/colab-ai-blog/all-blogs/create-a-co-lab-adviser-with-owui-part-2/ 17. Working with LLMs using Open WebUI - Rancher Desktop Docs, https://docs.rancherdesktop.io/tutorials/working-with-llms/ 18. OpenWebUI: How/Where do I change the directory where models are downloaded? - Reddit, https://www.reddit.com/r/ollama/comments/1g2oh9l/openwebui_howwhere_do_i_change_the_directory/ 19. (windows), HOW TO INSTALL IT on DIFFERENT drives than C???? · Issue #2546 - GitHub, https://github.com/ollama/ollama/issues/2546 20. Move Ollama Models to different location | by Rost Glukhov - Medium, https://medium.com/@rosgluk/move-ollama-models-to-different-location-755eaec1df96 21. Open WebUI: Home, https://docs.openwebui.com/ 22. OBS Virtual Camera + Open WebUI video chat mode = ALMOST a ..., https://www.reddit.com/r/LocalLLaMA/comments/1fk05n3/obs_virtual_camera_open_webui_video_chat_mode/ 23. 64 GB vs 128 GB RAM for a local AI “learning machine” for absolute newbie? - Reddit, https://www.reddit.com/r/framework/comments/1qjz02u/64_gb_vs_128_gb_ram_for_a_local_ai_learning/ 24. [MEGATHREAD] Local AI Hardware - November 2025 : r/LocalLLaMA - Reddit, https://www.reddit.com/r/LocalLLaMA/comments/1olq14f/megathread_local_ai_hardware_november_2025/ 25. Audio | Open WebUI, https://docs.openwebui.com/troubleshooting/audio/ 26. Getting Started with Open WebUI: A Self-Hosted AI Interface - DEV Community, https://dev.to/vpjigin/getting-started-with-open-webui-a-self-hosted-ai-interface-16mk 27. Features | Open WebUI, https://docs.openwebui.com/features/ 28. FAQ | Open WebUI, https://docs.openwebui.com/faq/ 29. RAG Tutorial - Open WebUI, https://docs.openwebui.com/tutorials/tips/rag-tutorial/ 30. Multi-Source RAG with Hybrid Search and Re-ranking in OpenWebUI | by Richard Meyer, https://medium.com/@richard.meyer596/multi-source-rag-with-hybrid-search-and-re-ranking-in-openwebui-8762f1bdc2c6 31. Local AI Chatbots: Setting Up Open WebUI | The AI Tester's Kit, https://aitesterkit.netlify.app/docs/learning-resources/open-source-llms/open-webui/ 32. Part 1: Getting Started with Local AI - Ollama & Open WebUI | by John Wong | Medium, https://medium.com/@able_wong/getting-started-with-local-ai-ollama-open-webui-part-1-bcaafcce6df7 33. Notion (MCP) - Open WebUI, https://docs.openwebui.com/tutorials/integrations/mcp-notion/ 34. I finally found a local LLM I actually want to use for coding - XDA Developers, https://www.xda-developers.com/finally-found-local-llm-want-use-coding/ 35. Best Local LLMs - 2025 : r/LocalLLaMA - Reddit, https://www.reddit.com/r/LocalLLaMA/comments/1pwh0q9/best_local_llms_2025/ 36. Deep Dive: Building a Self-Hosted AI Agent with Ollama and Open WebUI - Cohorte Projects, https://www.cohorte.co/blog/deep-dive-building-a-self-hosted-ai-agent-with-ollama-and-open-webui