OpenViking

OpenViking — Context Database for AI Agents

What it is: OpenViking is an open-source context database from ByteDance's Volcengine team that replaces flat vector storage with a hierarchical virtual filesystem. Instead of dumping everything into embeddings and hoping retrieval works, it organises all agent context into structured directories accessible via `viking://` URIs — like a Unix filesystem for your agent's brain.

Why it matters: In benchmarks with OpenClaw, OpenViking improved task completion from 35.65% to 52.08% (43% improvement) while cutting input tokens from 24.6M to 4.3M (91% reduction). The tiered loading system means your agent only pulls in what it needs, when it needs it.

GitHub: github.com/volcengine/OpenViking
Stars: ~17,900
License: Apache 2.0
Language: Python (core) + Rust (CLI) + C++ (vector extensions) + Go (AGFS server)
Requires: Python 3.10+

How It Works

Virtual Filesystem

All agent context lives in three root directories:

- `viking://resources/` — documents, repos, web pages, project files
- `viking://user/` — preferences, habits, personal context
- `viking://agent/` — skills, task memories, instructions

You interact with context using Unix-like commands: `ls`, `find`, `read`, `tree`, `grep`. The agent navigates its own memory the same way you'd navigate a filesystem.

L0/L1/L2 Tiered Loading

This is the key innovation that solves token bloat. Instead of loading full documents into context, OpenViking processes everything into three tiers:

- **L0 (Abstract):** ~100 tokens — one-sentence summary for quick identification
- **L1 (Overview):** ~2,000 tokens — structured overview with key details
- **L2 (Full Content):** Complete document, loaded on demand only

The agent starts with L0 summaries, drills into L1 when something looks relevant, and only loads L2 when it genuinely needs the full content. Claims 91-95% token cost reduction vs traditional RAG approaches.

Automatic Memory Self-Iteration

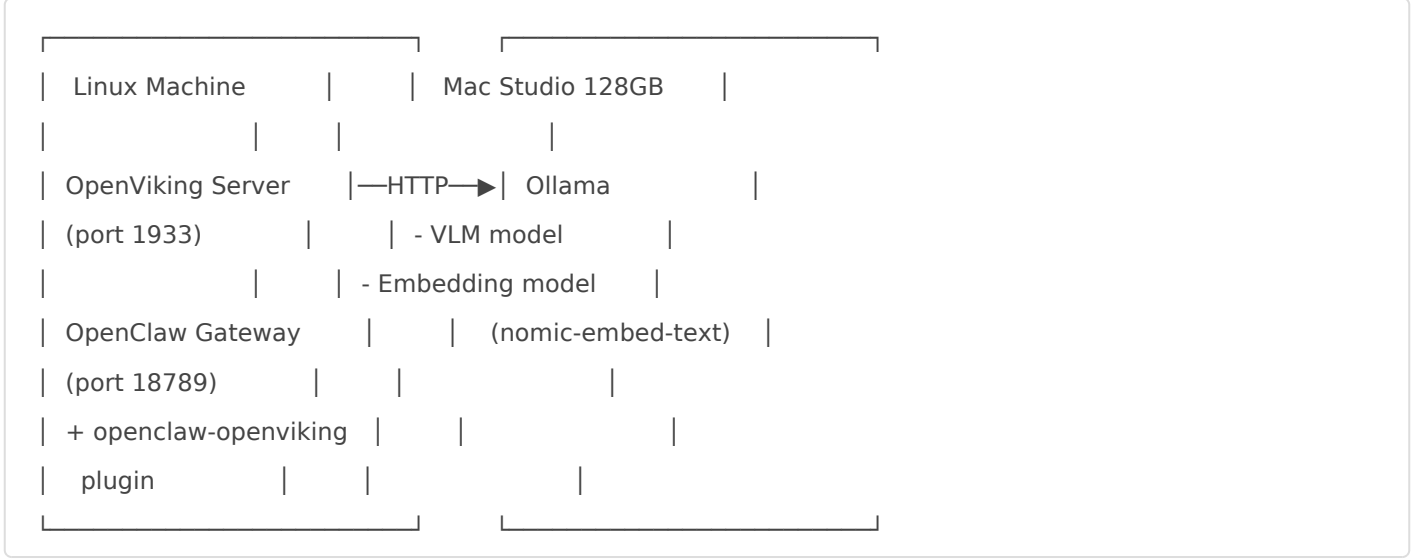
At session end, OpenViking automatically extracts six categories of memory and updates the appropriate directories:

1. **Profile** — factual information about the user
2. **Preferences** — how the user likes things done
3. **Entities** — people, projects, tools referenced
4. **Events** — decisions made, things that happened
5. **Cases** — problems solved, approaches taken
6. **Patterns** — recurring behaviours and workflows

This is significantly more structured than OpenClaw's default "hope the LLM remembers to write to MEMORY.md" approach.

Architecture For Our Setup

OpenViking runs on the Linux agent box. All LLM inference (VLM and embedding) routes to the Mac Studio over Headscale.



VLM Providers

OpenViking supports three VLM providers. For our local setup, use `openai` to bypass the LiteLLM dependency entirely:

Provider	Description	Use For Our Setup?
`volcengine`	ByteDance's Doubao models (cloud)	No — cloud only
`openai`	Any OpenAI-compatible API endpoint	**Yes — point at Ollama on Mac**
`litellm`	Multi-provider routing via LiteLLM	**No — compromised, avoid**

Installation

Prerequisites

```
# Install uv (Python package manager used by OpenViking)
curl -LsSf https://astral.sh/uv/install.sh | sh

# Install the Rust CLI
curl -fsSL https://raw.githubusercontent.com/volcengine/OpenViking/main/crates/ov_cli/install.sh | bash
```

Install OpenViking (with LiteLLM workaround)

```
# Clone the repo
git clone https://github.com/volcengine/OpenViking.git
cd OpenViking

# Create virtual environment
uv venv --python 3.11
source .venv/bin/activate

# IMPORTANT: Edit pyproject.toml to remove or comment out litellm dependency
# Find the line with litellm>=1.0.0 and remove it
nano pyproject.toml

# Install
uv pip install -e .
```

Install VikingBot (agent framework, optional)

```
uv pip install -e ".[bot]"
```

Install OpenClaw Plugin

```
# The official OpenClaw integration plugin
openclaw plugins install openclaw-openviking-plugin
```

Configuration

Create the config file at `~/.openviking/ov.conf`:

```
{
  "storage": {
    "workspace": "/home/conor/openviking_workspace"
  },
  "log": {
    "level": "INFO",
    "output": "stdout"
  },
  "embedding": {
    "dense": {
      "api_base": "http://100.64.0.9:11434/v1",
      "api_key": "not-needed",
      "provider": "openai",
      "dimension": 768,
      "model": "nomic-embed-text"
    },
    "max_concurrent": 10
  },
  "vlm": {
    "api_base": "http://100.64.0.9:11434/v1",
    "api_key": "not-needed",
    "provider": "openai",
    "model": "qwen3-coder-next",
    "max_concurrent": 100
  }
```

```
}
```

Notes:

- `100.64.0.9` is the Mac Studio's Headscale IP
- `provider: "openai"` works with any OpenAI-compatible endpoint including Ollama
- Embedding dimension must match the model — 768 for nomic-embed-text, 1024 for mxbai-embed-large
- The VLM model handles the intelligent context processing (summarisation, extraction, classification)

Running

```
# Start the OpenViking server
openviking-server

# Default address: 127.0.0.1:1933

# Start with VikingBot enabled (optional)
openviking-server --with-bot
```

Key Commands (ov CLI)

Adding Resources

```
# Add a GitHub repo
ov add-resource https://github.com/volcengine/OpenViking

# Add a local directory
ov add-resource /home/conor/.openclaw/workspace

# Wait for processing to complete (otherwise runs async)
ov add-resource /path/to/docs --wait
```

Browsing Context

```
# List root directories
ov ls viking://resources/

# Tree view (2 levels deep)
ov tree viking://resources/my-project -L 2

# Read a specific file at L1 (overview)
ov read viking://resources/my-project/README.md

# Check server status
ov status
```

Searching

```
# Semantic search across all context
ov find "what is the camera VLAN setup"

# Grep for exact terms within a scope
ov grep "RTSP" --uri viking://resources/home-automation
```

Interactive Chat (with VikingBot)

```
# Start interactive chat session
ov chat
```

OpenClaw Integration

Once the `openclaw-openviking-plugin` is installed, OpenViking becomes available as a context source for your agent. The plugin connects to the OpenViking server running on the same machine and provides the agent with access to the virtual filesystem commands.

The OpenViking server must be running before the OpenClaw gateway starts. Consider adding it as a systemd service:

```
# Example systemd service file
# /etc/systemd/user/openviking.service

[Unit]
Description=OpenViking Context Database
```

```
Before=openclaw.service

[Service]
ExecStart=/home/conor/OpenViking/.venv/bin/openviking-server
WorkingDirectory=/home/conor/OpenViking
Restart=always
RestartSec=5

[Install]
WantedBy=default.target
```

```
# Enable and start
systemctl --user enable openviking
systemctl --user start openviking
```

How OpenViking Differs From The Other Memory Tools

Feature	OpenClaw Default	memory-lancedb-pro	QMD	OpenViking
Storage model	Flat markdown files	Vector DB + BM25	SQLite + vector index	Virtual filesystem
Token efficiency	Loads everything	Top-N retrieval	Top-N retrieval	L0/L1/L2 tiered loading
Context organisation	None	Scopes (global/agent/project)	Collections	Hierarchical directories
Auto-categorisation	No	6 categories	No	6 categories + directory structure
Agent can browse	Read specific files	Search only	Search only	ls, tree, find, grep, read
Multi-resource support	Workspace only	Conversation memories	Markdown files	Git repos, URLs, docs, local dirs

OpenViking and memory-lancedb-pro are complementary, not competing. memory-lancedb-pro handles conversation-level memory (what you said, preferences extracted from chat). OpenViking handles resource-level context (your codebase, documentation, project files, knowledge bases). QMD provides the search layer across markdown files. Together they cover different retrieval needs.

Breaking Changes Warning

From the release notes: **after upgrading OpenViking, datasets/indexes generated by previous versions are not compatible.** You must rebuild:

```
# Stop the server
systemctl --user stop openviking

# Clear workspace (this deletes all indexed data — resources will need re-adding)
rm -rf /home/conor/openviking_workspace

# Restart
systemctl --user start openviking

# Re-add your resources
ov add-resource /path/to/your/stuff --wait
```

Key Files and Paths

- **Config:** `~/.openviking/ov.conf` (override with `OPENVIKING_CONFIG_FILE` env var)
- **Workspace:** Configured in ov.conf `storage.workspace` — stores all indexed data
- **CLI binary:** `ov` (Rust, installed via install script)
- **Server:** `openviking-server` (Python, from the pip install)
- **Default port:** 1933

Version: v0.2.1 (current as of March 2026)
Status: Alpha — performance and consistency not fully optimised

GitHub: github.com/volcengine/OpenViking

Docs: openviking.ai

Revision #2
Created 26 March 2026 11:26:12 by Conor
Updated 27 March 2026 22:40:07 by Conor