

Local models

System Overview & Hardware Architecture

The local AI inference stack on `ul` leverages dedicated AMD hardware acceleration paired with extensive system memory and high-capacity storage:

- **GPU: AMD Radeon AI PRO R9700** (Navi 48, RDNA 4 / `gfx1201`)
 - **VRAM: 32 GB GDDR6** high-bandwidth dedicated video memory
 - **Compute Runtime:** Vulkan (RADV GFX1201) / ROCm 7.2 stack
- **Host CPU:** AMD Ryzen Threadripper PRO 5955WX (16 Cores / 32 Threads @ up to 4.5 GHz)
- **Host Memory: 128 GB DDR4 ECC RAM** (125 GiB addressable)
- **Model Storage:** `/data/ollama` on the 160 TB ZFS storage pool
- **Inference Server: Ollama 0.34.2** managed via systemd (`ollama.service`)

With 32 GB of dedicated GPU VRAM and 128 GB of host RAM, the system supports both **100% GPU-offloaded execution** (models up to ~27-30 GB running at maximum throughput) and **hybrid GPU/CPU offload** (models up to 70B+ spanning across VRAM and system memory).

Active Model Roster & Hardware Allocation

All models are tuned with `num_ctx 32768` and verified with autonomous tool calling.

1. Tier 1: 100% GPU VRAM Offload (Ultra-Fast, ~40-55 tok/s)

These models fit completely inside the 32 GB GDDR6 VRAM buffer with 10–16 GB of VRAM headroom for deep 32k KV-caches:

- **qwen3-coder:30b (30B Active-MoE)**
 - **Quantization:** Q5_K_M (~21 GB)
 - **Architecture:** Mixture-of-Experts (30B parameters with 3B active per token).
 - **Throughput:** ~50–60 tokens/sec (100% GPU VRAM offload).
 - **Best For:** Cutting-edge software engineering, autonomous tool calling, refactoring, and multi-file code generation.
- **deepseek-r1-abliterated:32b (32B Uncensored)**
 - **Quantization:** Q4_K_M (~19 GB)
 - **Throughput:** ~35–45 tokens/sec (100% GPU VRAM offload).
 - **Best For:** Deep multi-step `<think>` chain-of-thought reasoning with **mathematical refusal vectors stripped**. Unfiltered, unaligned analysis for security research, reverse engineering, exploit analysis, and technical problem solving.
- **qwen3.8:27b (27B)**
 - **Quantization:** Q4_K_M (~16 GB)
 - **Throughput:** ~45–55 tokens/sec (100% GPU VRAM offload).
 - **Best For:** Modern general intelligence, agent execution, and command orchestration.
- **qwen2.5:32b (32B)**
 - **Quantization:** Q4_K_M (~19 GB)
 - **Throughput:** ~35–42 tokens/sec. Reliable fallback workhorse.
- **deepseek-r1:32b (32B)**
 - **Quantization:** Q4_K_M (~19 GB)
 - **Throughput:** ~35–40 tokens/sec. Standard alignment reasoning.

2. Tier 2: Hybrid VRAM + RAM Partitioning (~6–9 tok/s)

- **llama3.3:70b (70B)**
 - **Quantization:** Q4_K_M (~42 GB)
 - **Memory Partition:** 48 of 81 layers offloaded to GPU VRAM (~24 GB); remaining layers in system DDR4 ECC memory.
 - **Best For:** Frontier-grade multi-document synthesis and complex creative analysis.
-

Censorship, Alignment & Abliteration

Standard commercial and base open weights (e.g. standard Llama releases) include safety-tuning that often produces unwanted refusals, disclaimers, or moralizing lectures on benign technical queries (e.g. penetration testing, reverse engineering, security research, or creative writing).

What is Weight Abliteration?

Abliteration mathematically identifies the specific activation directions and weight vectors responsible for refusal responses and orthogonally projects them out of the model layers.

- Unlike coarse fine-tuning, abliteration does **not** induce catastrophic forgetting or degrade benchmark reasoning capabilities.
 - The model complies directly with technical, security, and administrative tasks without artificial guardrail refusals.
-

Running Models via CLI

Ollama is running as a local service listening on `127.0.0.1:11434`.

Interactive Chat Session

```
# Run Qwen 2.5 32B interactively
ollama run qwen2.5:32b

# Run DeepSeek-R1 32B reasoning model
ollama run deepseek-r1:32b

# Run Llama 3.3 70B
ollama run llama3.3:70b
```

Management Commands

```
# List all locally cached models and their disk usage
ollama list

# Check currently active loaded models in VRAM/RAM
ollama ps
```

```
# Pull a new model
ollama pull <model-name>

# Remove a model
ollama rm <model-name>
```

Using with OpenCode CLI (`opencode`)

You can run `opencode` directly against the local Ollama daemon using its OpenAI-compatible API interface.

1. Configuration in

`~/.config/opencode/opencode.jsonc`

Add the local Ollama provider to your OpenCode configuration:

```
{
  "$schema": "https://opencode.ai/config.json",
  "instructions": [
    "~/.config/opencode/INSTRUCTIONS.md"
  ],
  "provider": {
    "ollama": {
      "name": "Local Ollama",
      "npm": "@ai-sdk/openai-compatible",
      "options": {
        "baseUrl": "http://127.0.0.1:11434/v1",
        "apiKey": "ollama"
      },
    },
    "models": {
      "qwen2.5:32b": {
        "name": "Qwen 2.5 32B Local",
        "limit": {
```

```
{
  "context": 32768,
  "output": 8192
},
{
  "deepseek-r1:32b": {
    "name": "DeepSeek R1 32B Local",
    "reasoning": true,
    "limit": {
      "context": 32768,
      "output": 8192
    }
  },
  "llama3.3:70b": {
    "name": "Llama 3.3 70B Local",
    "limit": {
      "context": 32768,
      "output": 8192
    }
  }
}
```

2. Invoking OpenCode with Local Models

```
# Launch interactive session with local Qwen 32B
opencode -m ollama/qwen2.5:32b

# Run a one-off prompt
opencode run -m ollama/qwen2.5:32b "Analyze the system logs in /var/log/nginx/access.log"

# Use DeepSeek-R1 for complex architectural reasoning
opencode -m ollama/deepseek-r1:32b
```

3. Model Inspection & Auto-Sync Tool (`opcode_list`)

A dedicated CLI tool `/home/luke/bin/opencode_list` is installed in `$PATH` to monitor local models, their hardware execution profile, installation date/age, and capabilities:

```
# View all installed models with stats, age, and capabilities (newest at bottom)
opencode_list

# Sort by size or name if preferred
opencode_list --sort size
opencode_list --sort name

# Output raw JSON for scripting
opencode_list --json
```

Auto-Synchronization: Whenever you pull a new model via `ollama pull <name>`, simply running `opencode_list` automatically detects the new weights, derives its capabilities, and writes the model entry directly into `~/.config/opencode/opencode.jsonc`. No manual JSON configuration is required.

Sourcing & Discovering Models: Why NOT to Rely on Ollama's Catalog

1. Ollama's Library is Severely Outdated

Ollama's official online catalog (`ollama.com/library`) is manually curated by the Ollama maintainers. Official tags (e.g. `ollama pull ...`) are only created weeks or months after open-source model releases have saturated the community.

Never rely on Ollama's catalog to see what models exist.

2. Hugging Face is the Upstream Source of Truth

All cutting-edge open weights drop first on **Hugging Face** (<https://huggingface.co/>), often within hours of release:

- **Quantizers to watch:** `unsloth` (`unsloth/Qwen...`), `bartowski`, and model creators directly (`Qwen`, `meta-llama`, `mistralai`).
- **VRAM Sizing Rule of Thumb (for 32 GB R9700):**
 - **27B - 32B Models (Q4_K_M):** ~15-18 GB file size. Consumes ~17-21 GB VRAM with full 32,768 context KV-cache. **100% GPU offload, zero system RAM penalty.**
 - **70B Models (Q4_K_M):** ~42-44 GB file size. Automatically partitions across VRAM (~24 GB) and host DDR4 RAM.

3. The Direct Hugging Face → Ollama Workflow

Because Ollama's internal pull tool sometimes blocks CDN redirects from Hugging Face (`hf.co`), download directly and import:

1. Download the GGUF file to `/data/llm/models/`:

```
curl -L -C - -o /data/llm/models/Qwen3.8-27B-UD-Q4_K_M.gguf \  
  "https://huggingface.co/unsloth/Qwen3.8-27B-GGUF/resolve/main/Qwen3.8-27B-UD-  
  Q4_K_M.gguf"
```

2. Create a clean Modelfile (`/tmp/Modelfile`):

```
FROM /data/llm/models/Qwen3.8-27B-UD-Q4_K_M.gguf  
PARAMETER num_ctx 32768
```

3. Import into Ollama:

```
sudo -u ollama ollama create qwen3.8:27b -f /tmp/Modelfile
```

Tool Calling & Autonomous Agent Execution (OpenCode / Aider)

For local models to successfully execute commands, access the filesystem, and write code inside autonomous agents:

1. The 2k Context Window Trap (`num_ctx`)

- By default, **Ollama sets a 2,048-token context window** on all models if not specified in the Modelfile.
- When agents like OpenCode or Aider send system instructions and 15+ JSON tool schemas, the initial prompt is often **8,000+ tokens**.
- **Ollama silently discards everything past 2,048 tokens**, chopping off the tool definitions before the model even reads your message.
- **Fix:** Every Modelfile must explicitly define `PARAMETER num_ctx 32768`.

2. Schema Validation Strictness (JSON Typing)

OpenCode validates tool arguments with strict TypeScript schemas. If a model passes numeric parameters (like `timeout`) as quoted strings (`"timeout": "120000"`), the schema validator rejects the call.

- Add an explicit instruction to `~/.config/opencode/INSTRUCTIONS.md`:

```
# Tool Calling Schema Rule
When invoking tools, all numeric parameters (such as timeout) MUST be passed as
unquoted integers (e.g. timeout: 120000), NEVER as strings.
```

3. Upstream Parser Differences: Qwen 3.8 & Qwen 2.5 vs. Qwen-Coder

- `qwen3.8:27b` and `qwen2.5:32b`: Properly emit structured `<tool_call>` XML tags that Ollama recognizes, allowing OpenCode to execute `bash`, `read`, `write`, and `edit` commands in real time.
- `qwen2.5-coder:32b`: Dumps raw JSON without `<tool_call>` tags, causing Ollama's regex to fail and dump the text to the chat window instead of triggering the tool.

Other Interfaces & Integrations

The local Ollama instance exposes standard APIs that integrate seamlessly with any LLM client or Web UI.

1. OpenAI-Compatible API Endpoint

- **Base URL:** `http://127.0.0.1:11434/v1`
- **API Key:** Any string (e.g. `ollama`)

Example `curl` call:

```
curl http://127.0.0.1:11434/v1/chat/completions \  
-H "Content-Type: application/json" \  
-d '{  
  "model": "qwen2.5:32b",  
  "messages": [  
    {"role": "system", "content": "You are a Linux systems engineering assistant."},  
    {"role": "user", "content": "Provide a Bash one-liner to find open sockets by PID."}  
  ],  
  "temperature": 0.2  
'
```

2. Python Client (`openai` SDK)

```
from openai import OpenAI  
  
client = OpenAI(  
    base_url="http://127.0.0.1:11434/v1",  
    api_key="ollama",  
)  
  
response = client.chat.completions.create(  
    model="qwen2.5:32b",  
    messages=[{"role": "user", "content": "How do I optimize ZFS ARC on Linux?"}],  
)  
print(response.choices[0].message.content)
```

3. Open WebUI (Rootless Podman Setup)

To deploy a ChatGPT-style web interface connected directly to the local Ollama instance (`169.254.1.2:11434` mapped via `passt` user-mode networking to `127.0.0.1:11434`):

```
(cd /tmp && sudo -u docker-data podman run -d \
  --name open-webui \
  --network pasta:--map-host-loopback,169.254.1.2 \
  -p [fd77:7777:5010:0:1337:f1ac:d00d:0e81]:8080:8080 \
  -v /data/open-webui:/app/backend/data \
  -e OLLAMA_BASE_URL=http://169.254.1.2:11434 \
  ghcr.io/open-webui/open-webui:main)
```

(Configure reverse proxy with Authentik SSO in `/etc/nginx/sites-available/`).

4. VS Code / JetBrains (Continue.dev Extension)

In `~/.continue/config.json`:

```
{
  "models": [
    {
      "title": "Local Qwen 2.5 32B",
      "provider": "ollama",
      "model": "qwen2.5:32b",
      "apiBase": "http://127.0.0.1:11434"
    }
  ],
  "tabAutocompleteModel": {
    "title": "Local Qwen 2.5 Coder 7B",
    "provider": "ollama",
    "model": "qwen2.5-coder:7b",
    "apiBase": "http://127.0.0.1:11434"
  }
}
```

Service Configuration & Storage

- **Service Unit:** `/usr/lib/systemd/system/ollama.service`
- **Override Config:** `/etc/systemd/system/ollama.service.d/override.conf`
 - `OLLAMA_MODELS=/data/ollama` (routes all model weights to the fast 160 TB pool)

- `OLLAMA_SCHED_SPREAD=1`
- `OLLAMA_MEMORY_VALIDATION=false`
- **GPU Permissions:** User `ollama` belongs to `video` and `render` groups (`/dev/kfd` and `/dev/dri/renderD128`).
- **Service Management:**

```
sudo systemctl restart ollama  
sudo journalctl -u ollama -f
```

Revision #2

Created 2026-09-21 13:05:54 UTC by luKe

Updated 2026-10-20 03:46:09 UTC by luKe