SmarterRouter: A VRAM-Aware LLM Gateway for Your Local AI Lab

February 21, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** SmarterRouter (v2.1.1, Feb 2026) is a Python-based LLM gateway that sits between your apps and your local backends (Ollama, llama.cpp, or OpenAI-compatible APIs). It profiles each model's speed and VRAM footprint on your hardware, then auto-routes each prompt to the best model for the task. When VRAM gets tight, it auto-unloads the least recently used model. It caches responses to similar queries so the same question doesn't burn GPU time twice. Drop-in OpenAI-compatible API on port 11436 — point OpenWebUI, Continue, or any OpenAI client at it and it handles model selection automatically. Very new (31 stars, single developer), but the design solves a real problem: managing multiple models on limited VRAM without babysitting.

📚 **Related:** llama.cpp vs Ollama vs vLLM · Open WebUI Setup Guide · VRAM Requirements · Model Formats Explained · Planning Tool

If you run local AI seriously, you hit the multi-model wall. Qwen-Coder for code. Llama for general chat. A vision model for images. Maybe a small model for quick tasks. But you have one GPU — maybe 24GB if you're lucky — and you can't load them all at once.

So you babysit. You manually swap models in Ollama. You watch `nvidia-smi` to make sure nothing OOMs. You re-run the same queries because you forgot you asked yesterday. It works, but it's tedious.

SmarterRouter (GitHub, v2.1.1, released Feb 2026) automates the whole thing. It profiles your models, tracks your VRAM, caches responses to similar queries, and auto-routes each prompt to the best available model. Drop-in OpenAI-compatible API. Point OpenWebUI at it, and it handles the rest.

The Multi-Model Problem

Ollama handles model switching, but it's reactive — it loads what you ask for and evicts what doesn't fit. There's no intelligence about which model is best for a given prompt, no caching of responses, and no awareness of how much VRAM each model actually consumes on your specific hardware.

llama.cpp's router mode (added late 2025) offers native model management with LRU eviction, but it only works within the llama.cpp ecosystem and has no caching or intelligent routing.

Tools like llama-swap and Olla solve pieces of the puzzle — model orchestration, load balancing, failover — but none combine VRAM management, semantic caching, and intelligent model selection in one package designed for home labs.

That's the gap SmarterRouter fills.

Setup

Docker (Recommended)

```
git clone https://github.com/peva3/SmarterRouter.git
cd SmarterRouter
docker-compose up -d
```

The container runs on port **11436** and needs to reach your backend. If Ollama runs on the host, use `http://172.17.0.1:11434` as the backend URL (Docker's default bridge gateway).

GPU-specific compose templates exist in `docs/` for NVIDIA, AMD ROCm, Intel Arc, and multi-GPU setups.

Manual Install

```
git clone https://github.com/peva3/SmarterRouter.git
cd SmarterRouter
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
cp ENV_DEFAULT .env
# Edit .env for your setup
python -m uvicorn main:app --host 0.0.0.0 --port 11436
```

Requires Python 3.11+.

Backend Configuration

All config lives in `.env`. The essentials:

```
# Ollama (default)
ROUTER_PROVIDER=ollama
ROUTER_OLLAMA_URL=http://localhost:11434

# Or llama.cpp
ROUTER_PROVIDER=llama.cpp
ROUTER_LLAMA_CPP_URL=http://localhost:8080

# Or any OpenAI-compatible endpoint
ROUTER_PROVIDER=openai
ROUTER_OPENAI_BASE_URL=https://your-endpoint/v1
ROUTER_OPENAI_API_KEY=your-key
```

On first startup, SmarterRouter auto-discovers all models from your backend and begins profiling. The initial profiling run takes 30-60 minutes depending on how many models you have — it benchmarks each one with standardized prompts across reasoning, coding, and creativity categories.

How the Routing Works

SmarterRouter analyzes each incoming prompt and scores available models using four signals:

Query difficulty prediction. It classifies prompts as Easy, Medium, or Hard based on logic indicators, code structures, and instruction complexity. A hard coding prompt won't get routed to a 3B model.

Model profiling. MT-Bench-inspired benchmarks run locally on your hardware, scoring each model on reasoning, coding, and creativity. This isn't a generic leaderboard — it's your actual tok/s and quality on your GPU.

Static benchmarks. Optional external data from HuggingFace Open LLM Leaderboard and LMSYS Chatbot Arena to supplement local profiling.

User feedback. A `/v1/feedback` endpoint lets you rate responses (0.0-2.0), and the router adjusts future routing based on accumulated ratings.

A global quality-vs-speed tuner (`ROUTER_QUALITY_PREFERENCE`, 0.0-1.0) lets you bias toward faster responses or higher quality. And if the selected model fails, cascading fallback automatically retries with the next best option.

The router presents itself as a single model (`smarterrouter/main`) to clients. You point OpenWebUI or any OpenAI-compatible tool at `http://localhost:11436/v1` and it handles model selection transparently.

VRAM Management

This is the core differentiator and the reason the project is interesting.

How It Works

A background monitor polls GPU memory at 30-second intervals. SmarterRouter supports four GPU vendors:

Vendor	Detection Method
NVIDIA	`nvidia-smi` query
AMD	`rocm-smi` or sysfs fallback
Intel Arc	sysfs `lmem_total`
Apple Silicon	75% of unified system RAM

When a prompt arrives that needs a model not currently loaded, the VRAM manager checks whether it fits:

```
available = (max_vram - 1.5GB buffer) - sum(loaded models)
```

If it doesn't fit, the manager auto-unloads models using one of two strategies:

- **LRU** (default): evict the least recently used model
- **Largest**: evict the biggest model first to free the most memory

You can pin critical models that should never be unloaded:

```
ROUTER_PINNED_MODEL=qwen3:8b
```

What Gets Profiled

During initial profiling, SmarterRouter measures each model's actual VRAM footprint — not the theoretical size from the model card, but the real memory consumption on your GPU. It takes a baseline reading, loads the model, waits for memory stabilization, and measures the delta. Three samples averaged. This means a Q4_K_M Qwen3 14B that theoretically needs 9GB but actually uses 10.2GB on your specific setup gets profiled at 10.2GB.

Key VRAM settings:

Setting	Default	Purpose
`ROUTER_VRAM_MAX_TOTAL_GB`	Auto (90% of detected)	VRAM budget
`ROUTER_VRAM_AUTO_UNLOAD_ENABLED`	`true`	Auto-evict when full
`ROUTER_VRAM_UNLOAD_STRATEGY`	`lru`	Eviction strategy
`ROUTER_VRAM_DEFAULT_ESTIMATE_GB`	`8.0`	Fallback for unprofiled models
`ROUTER_MODEL_KEEP_ALIVE`	`-1` (indefinite)	How long loaded models persist

Semantic Caching

Three-layer caching system, each progressively smarter:

Layer 1 — Exact match (always active). SHA-256 hash of the prompt. If you send the exact same prompt twice, instant response from cache. No embedding model needed.

Layer 2 — Semantic similarity (optional). Enable by setting an embedding model:

```
ROUTER_EMBED_MODEL=nomic-embed-text:latest
ROUTER_CACHE_SIMILARITY_THRESHOLD=0.85
```

Now "explain how transformers work" and "how do transformer models function" can hit the same cache entry. Uses cosine similarity on embeddings. The 0.85 threshold is conservative — lower it for more cache hits at the cost of occasional mismatches.

Layer 3 — Response cache. Caches the actual LLM output, keyed by model + prompt hash + generation parameters (temperature, top_p, seed). A cached response at temperature 0.7 won't be served for a request at temperature 0.1.

Setting	Default
`ROUTER_CACHE_ENABLED`	`true`
`ROUTER_CACHE_MAX_SIZE`	`500` entries
`ROUTER_CACHE_TTL_SECONDS`	`3600` (1 hour)
`ROUTER_CACHE_RESPONSE_MAX_SIZE`	`200` entries

For a home lab where you repeatedly ask similar questions during development, coding, or research, semantic caching alone can save significant GPU time.

Connecting to OpenWebUI

The primary use case. In OpenWebUI:

1. Settings → Connections → Add Connection
2. Base URL: `http://localhost:11436/v1`
3. API Key: leave empty (or set `ROUTER_ADMIN_API_KEY` in production)
4. Select model: `smarterrouter/main`

That's it. Every prompt you send through OpenWebUI now gets auto-routed to the best model for the task, with VRAM managed automatically and responses cached. You see one "model" in the UI but the router is selecting from your full model library behind the scenes.

Also works with Continue (VS Code), Cursor, SillyTavern, and any tool that speaks the OpenAI API.

How It Compares

	SmarterRouter	Ollama (native)	llama.cpp Router	llama-swap
Intelligent routing	Auto (profiled)	Manual	None	Manual config
VRAM management	Full (monitor + auto-unload)	Basic swap	LRU eviction	Basic swapping
Semantic caching	Yes (exact + embedding)	No	No	No
Model profiling	Yes (MT-Bench-style)	No	No	No

	SmarterRouter	Ollama (native)	llama.cpp Router	llama-swap
Multi-backend	Ollama + llama.cpp + OpenAI	Ollama only	llama.cpp only	Multi-engine
GPU support	NVIDIA, AMD, Intel, Apple	NVIDIA, AMD, Apple	NVIDIA, AMD, Apple	Delegates
Feedback loop	Yes (user ratings)	No	No	No
Maturity	6 days old, 31 stars	Established	Native	Established

SmarterRouter does more than any single alternative. The question is whether it does them reliably at this stage.

Honest Limitations

Six days old. Created February 16, 2026. One contributor. 31 stars. The version numbers jumped from v1.5 to v2.1 in less than a week, which suggests rapid iteration but also instability. Expect breaking changes.

Ambitious scope for a solo project. Intelligent routing, VRAM management, semantic caching, multi-vendor GPU support, model profiling, feedback loops, Prometheus metrics, admin API — all maintained by one person in Python. That's a lot of surface area for bugs.

Profiling takes time. The initial 30-60 minute profiling run is a real cost. If you frequently add or remove models, you'll be re-profiling regularly.

Routing accuracy depends on prompt analysis. The difficulty classifier uses heuristics — code keywords, logic indicators, instruction density. It will misclassify prompts. The quality-vs-speed tuner helps, but expect occasional suboptimal routing.

Only Ollama supports full VRAM management. The llama.cpp and OpenAI backends lack load/unload API support, so VRAM management is limited to monitoring on those backends.

No community yet. No Reddit discussion, no Hacker News thread, no Discord. If you hit a bug, you're filing a GitHub issue and hoping peva3 responds.

Who Should Use This

Now: Anyone running 3+ models on a single GPU through OpenWebUI who's tired of manually managing model switching. The VRAM management and caching alone justify the setup time, even if you ignore the intelligent routing.

Also now: Multi-GPU setups where VRAM accounting across GPUs is a genuine headache. SmarterRouter's multi-GPU monitoring (NVIDIA + AMD + Intel) consolidates this.

Watch for: Distributed setups where you'd want a central router dispatching to multiple inference nodes. The OpenAI-compatible backend support means you could point SmarterRouter at multiple llama.cpp servers on different machines. This is exactly the kind of infrastructure a distributed thinking network needs.

Skip for now if: You run one model at a time and switch manually without friction. Or if you need production-grade reliability — this project is too new to trust with anything critical.

Bottom Line

The multi-model VRAM management problem is real, and it's one of the least-solved problems in local AI. Ollama gives you model switching. llama.cpp gives you LRU eviction. Neither gives you intelligent routing, semantic caching, or hardware-specific profiling.

SmarterRouter is the first tool that tries to solve all of these in one place, designed for the home lab rather than the datacenter. It's risky to depend on a 6-day-old project from a single developer. But the design is right, the scope is right, and the problem is real.

If you run multiple models on limited VRAM and want a smarter way to manage them, SmarterRouter is worth 20 minutes of setup time to see if it fits your workflow.

GitHub: peva3/SmarterRouter
