Best Way to Run 31B Models on a Laptop? Treat Them Like Databases

April 21, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** LARQL is an open-source CLI by chrishayuk that extracts a transformer's feed-forward layers into a queryable graph database called a vindex. Per the project docs, a browse-only vindex for a 4B model fits in ~3GB, and knowledge queries run in under 50ms with no GPU required. The pitch: stream the vindex from disk, patch in facts with small .vlp files, and run inference as a KNN walk instead of a dense matmul. Supported today: Gemma, Llama, Mistral, Mixtral, Qwen, Phi, DeepSeek, GPT-2. MoE support is partial — MXFP4 weights produce noisy walks per the docs. This is research-stage, the performance numbers come from the author, and no independent benchmarks exist yet. Worth understanding even if it's not yet worth replacing Ollama.

📚 **More on this topic:** VRAM Requirements · Qwen 3.5 by GPU · llama.cpp vs Ollama vs vLLM · Apple Silicon Local AI · Best Local LLMs for RAG

The standard take on local LLMs is that they’re opaque matrix-multiply machines that need a GPU to run. Load the weights into VRAM, do dense linear algebra, sample a token, repeat. If you want a bigger model, you buy more VRAM.

LARQL is built on a different premise. The argument is that the feed-forward network inside your transformer is already a graph database — one the model constructed during training. Features are edges. Entities are nodes. Relations are edge labels. Inference isn’t a dense matrix multiply; it’s a K-nearest-neighbor walk through that graph, touching only the subgraph a given query needs.

If the premise holds, three things follow. You can extract that graph to disk. You can query it without loading the full model into VRAM. And you can patch it — add a fact, remove a bias, swap in a domain — without retraining anything.

That’s the pitch. Now for the reality check.

What LARQL actually is

LARQL is an open-source project by chrishayuk — a Rust core with a CLI, a Python API, an HTTP/gRPC server, and a query language called LQL (Lazarus Query Language). Apache-2.0 license, active development, eight Rust crates in a clean dependency chain.

The core artifact is a **vindex** — short for vector index. You point LARQL at an open-weight model (safetensors, GGUF, or MLX format), and it reorganizes the weights into a queryable directory structure. The README puts it plainly:

> "The model IS the database. Query neural network weights like a graph database."
>
> "Gate vectors become a KNN index. Embeddings become token lookups. Down projections become edge labels."

Per the project docs, vindex extraction comes in three sizes for a 4B model:

Level	Size (f16)	Use case
Browse	~3 GB	Knowledge queries only (DESCRIBE, WALK, SELECT)
Inference	~6 GB	Full forward pass with attention
All	~10 GB	Everything including training metadata

Those are the author's numbers from the LARQL docs. They haven't been independently verified.

LQL itself is a SQL-shaped language with 20+ statement types across lifecycle (EXTRACT, COMPILE, DIFF, USE), browse (WALK, DESCRIBE, SELECT), inference (INFER), and mutation (INSERT, DELETE, UPDATE, MERGE). You write queries. The engine walks the vindex.

The three moving parts

vindex files hold the model's knowledge graph — gate vectors as the KNN index, embeddings as token lookups, down projections as edge labels, plus config and relation-cluster metadata. They're immutable and memory-mappable.

.vlp patch files sit on top as JSON overlays. The docs describe them as capturing INSERT/DELETE/UPDATE operations without touching the base vindex. They stack, they reverse, and `DIFF` generates them from vindex deltas. Per the README, "a single fact is ~10 KB. A 1,000-fact domain patch is ~10 MB. Compared to the full model at 8 GB, that's 1/800th the size." Again — author's claim, not independently benchmarked.

Vindexfile is the declarative build system. Think Dockerfile, but for knowledge:

```
FROM hf://chrishayuk/gemma-3-4b-it-vindex
PATCH hf://medical-ai/drug-interactions@2.1.0
PATCH ./patches/company-facts.vlp
INSERT ("Acme Corp", "headquarters", "London")
```

Run `larql build . --output custom.vindex` and you get a reproducible, versioned assembly of a base model plus curated patches. If the approach works at scale, this is the interesting part.

The “walk” is the key idea

A walk is what replaces dense matmul at inference time. Classical transformer inference materializes every vocabulary dimension through every FFN feature at every layer. LARQL’s walk uses gate KNN (K ≈ 10 per layer, per the docs) to pick only the features that matter for the current residual, then reads the corresponding down-projection rows from mmap’d storage.

The headline benchmark chrishayuk cites for Gemma 3 4B on Apple Silicon:

Operation	Latency (claimed)
Gate KNN per layer	0.008 ms
Walk across 34 layers	0.3 ms
Load vindex	8 ms
Walk prediction (full)	33 ms
INFER with attention	517 ms
Dense baseline	535 ms

Two things to note. First, the walk-vs-dense comparison (517 vs 535 ms) is close — not an order of magnitude — and it’s the author’s own measurement on their own build. Second, “<50 ms per query” is cited for knowledge-only queries in the spec doc, which is a different regime than full autoregressive generation.

Treat these as “worth watching,” not proven.

Why this matters for running bigger models on modest hardware

Here's where the InsiderLLM angle actually lives. The project's pitch is that because a vindex is memory-mappable and queries only touch a small subgraph per token, you can in principle run a model bigger than your VRAM by streaming from disk.

From the vindex ecosystem spec:

> "On a laptop with 8GB RAM, browse a 4B model's knowledge."
>
> "Each layer is ~50-100MB (f16)."
>
> 70B browse: "~25 GB, CPU" vs "140 GB, multi-GPU" for full inference.
>
> "Gate vectors at int4: 0.42 GB — a 4B model's knowledge in 400 MB."

If those numbers hold for the 27-32B range — the size class most InsiderLLM readers care about — a browse-only vindex for a ~31B model should land somewhere in the 15-20 GB range, which is laptop territory on a modern machine with a fast SSD. The docs don't publish a measurement for that exact size, so this is extrapolation from the author's scaling claims, not a confirmed benchmark.

The longer-horizon pitch is decoupled serving: the workstation hosts the vindex, and a laptop or Pi runs attention plus queries it over the network. The project ships `larql-server` for HTTP/gRPC, and the spec describes the vindex files as "independently loadable" and "CDN-cacheable." The README doesn't spell out a production-grade remote-FFN-plus-local-attention split; that architecture is implied by the file layout more than it's documented as a supported mode.

Curated knowledge, not RAG duct tape

Most people building domain-specific local assistants today bolt RAG onto a generic model. You pick a chunker, an embedding model, a vector DB, a retrieval threshold, a prompt template — and when the model hallucinates, you're in a four-hour debug session trying to figure out which layer of the stack betrayed you.

LARQL's alternative framing is that the knowledge should live in the model's own graph structure, as a patch stacked on top of a base vindex. No separate retrieval step. No prompt-injection of chunks. No embedding-model mismatch.

The `.vlp` patch format makes this concrete. You write facts as triples, run `DIFF` or `INSERT`, and the patch sits alongside the base vindex as a readable JSON overlay. Two domain patches

can be merged. A patch can be removed. The base is read-only, so you never corrupt ground truth.

Whether this actually produces better factual recall than RAG is an open research question. But the ergonomic story is real — a 10 MB patch file you can version-control is a much nicer object than a ChromaDB directory plus an embedding pipeline plus a chunking strategy.

How this stacks against what you already use

Approach	What it does	What it can't do
RAG	Bolts retrieval onto a generic model	Retrieval quality is a separate stack; knowledge isn't "in" the model
Fine-tuning / LoRA	Shifts model behavior via weight updates	Changes are global and opaque; hard to inspect what changed
MoE	Sparsity at inference — route to a subset of experts	The router is a black box; sparsity decided at train time
Quantization	Shrinks weights (Q4, Q8, etc.)	Full model still loads into memory
LARQL (per the docs)	Sparsity at inference via KNN walk; knowledge is patchable	Research-stage; no independent benchmarks yet

The useful mental model: RAG is external knowledge, fine-tuning is implicit knowledge, LARQL is claimed to be explicit knowledge — visible as graph nodes, addressable, patchable.

What LARQL doesn't do yet

This is where the honesty check matters. There are real limitations.

Architecture support is uneven. The README lists Gemma 2/3, Llama 2/3, Mistral, Mixtral, Qwen, Phi, DeepSeek, and GPT-2 as supported input formats. That's broader than you might expect. But extraction isn't copy-paste across architectures — each new FFN variant needs a feature-extraction pass written for it. The mature demos are on Gemma 3 4B.

MoE is a partial story. Dense and full-precision MoE models support all operations per the docs, but MXFP4 MoE (the format used by GPT-OSS) gets this caveat: "can be extracted and served but DESCRIBE/WALK produce noisy results due to 4-bit weight precision." For Kimi K2.6,

DeepSeek V3, and similar routed-expert models at production quantization, the vindex approach is research-stage at best — sparsity is already handled by the router at training time, and re-expressing that as a per-expert graph is an unresolved problem.

Vindex build is heavy one-time compute. You're analyzing every FFN feature in the source model. The docs list operation latencies (KNN: 0.008 ms/layer, load: 8 ms) but don't publish a full extraction time. Expect hours, not minutes, for a mid-size model.

There are no independent benchmarks. Every compression ratio, every speed number, every accuracy claim on this page is from the LARQL project itself. The 3,100× boundary-residual compression, the 1/800th patch-vs-model size, the walk-beats-dense timing — all author-reported. Treat this as a research artifact, not a production spec sheet.

It is not a drop-in Ollama replacement. If you want a working local chatbot today, llama.cpp, Ollama, or vLLM is still the right stack. LARQL is a different kind of tool solving a different problem.

Who should actually try this today

Yes:

- Researchers and tinkerers working on Gemma or Qwen who want to poke at model internals
- People building domain-specific local assistants (legal, medical, technical docs) who've hit the ceiling on RAG quality
- Multi-node local AI setups where network inference and heterogeneous hardware are already in play
- Interpretability-curious readers who want a hands-on way to inspect what a model "knows"

No:

- Anyone who just wants a working chatbot on Apple Silicon — MLX or Ollama on Mac is still the answer
- Production deployments that need stability guarantees
- Users on MoE models at aggressive quantization (MXFP4), per the project's own caveats

---

Bottom line

LARQL is early, niche, and important.

The performance claims are unverified and come entirely from the author — no third-party benchmarks exist yet. Do not treat the 3,100× compression number or the "walk beats dense" timing as proven. Treat them as hypotheses worth testing.

But the underlying insight — that a trained FFN can be interpreted as a graph database, and inference can be re-expressed as a walk over that graph — is durable. Even if LARQL itself doesn't become the dominant tool, expect to see similar approaches in the next twelve months. The interpretability community has been circling this idea for a while, and a working CLI that lets you `EXTRACT`, `PATCH`, and `WALK` an open-weight model is the first practical handle on it.

If you care about local AI beyond "which quant fits in 24GB," this is worth understanding now so you're not playing catchup when a production-ready version — from chrishayuk or someone else — lands.

Read the LARQL repo, watch the author's talk "LLMs Are Databases — So Query Them", and try an extraction on a Gemma 3 4B checkpoint if you've got a spare afternoon. That's the fastest path to forming your own opinion on whether the premise holds.

Sources & further reading

- LARQL on GitHub — main repository, README, Rust source
- Vindex ecosystem spec — file format and serving architecture
- Residual trace docs — tiered context storage and compression claims
- "LLMs Are Databases — So Query Them" — chrishayuk's talk on the motivating idea

---