AI INFRASTRUCTURE

The Local LLM Handbook

Run AI models on your own hardware: complete guide to setup, models, and cost savings

by Rook

First Edition · February 2026

Table of Contents

The Complete Local LLM Handbook

A practical guide to running large language models on your own hardware.

CHAPTER 01

Why Go Local

Every prompt you send to ChatGPT, Claude, or Gemini travels to someone else's computer, gets processed on someone else's GPU, and passes through someone else's content filters. Your data — your code, your documents, your ideas — becomes training fodder for the next model version. For personal use, that trade-off might be acceptable. For anything serious, it's a liability.

Privacy is the obvious reason, but it's not the only one.

Running models locally means your data never leaves your machine. No terms of service. No data retention policies. No wondering whether that proprietary code you pasted into the chat window is now part of a training dataset. For anyone working with client data, medical records, legal documents, or proprietary code, local inference isn't a luxury — it's a requirement.

Then there's cost. Let's do the maths. A moderate ChatGPT Plus user pays $20/month — that's £192/year. A developer using the API for coding assistance might spend £30–80/month on tokens, totalling £360–960/year. A team of five? Multiply accordingly. Meanwhile, a one-time investment of £400–800 in hardware gives you unlimited inference forever. No per-token billing. No surprise invoices. The hardware pays for itself within 6–12 months, and you own it outright.

Offline capability matters more than people think. Aeroplanes, trains, rural areas, security-restricted environments — local models work everywhere. No internet required. No API outages. No rate limits.

Customisation is the final piece. Local models can be fine-tuned on your data, run without content filters, and configured exactly as you need them. You control the system prompt, the temperature, the context window, and the model weights. No one can deprecate your model or change its behaviour overnight.

The barrier to entry has collapsed. Two years ago, running a capable model locally required a £3,000 GPU. Today, a £400 mini PC runs models that rival GPT-3.5. The question is no longer *whether* you can run LLMs locally — it's *why you haven't started yet*.

CHAPTER 02

Hardware Guide

▸ How LLMs Actually Use Your Hardware

Before spending money, understand what you're buying and why.

LLMs are, at their core, enormous matrices of numbers (called **weights** or **parameters**). When you run a model, your hardware loads these weights into memory and performs billions of mathematical operations to generate each token. The speed of generation depends on two things: where the weights live, and how fast they can be read.

Memory capacity determines which models you can load. A 7-billion parameter model quantised to 4-bit precision needs roughly 4–5 GB of memory. A 70B model needs 35–40 GB. If the model doesn't fit in memory, it either won't load or will partially offload to disk — which is unusably slow.

Memory bandwidth determines how fast tokens are generated. This is the single most important spec for local LLM performance. DDR5 RAM delivers roughly 50–80 GB/s. An RTX 3090's GDDR6X delivers 936 GB/s.

Apple's M4 unified memory sits at around 120 GB/s. This is why GPU inference is dramatically faster than CPU inference — not because of compute power, but because of memory bandwidth.

GPU vs CPU inference: If your model fits entirely in VRAM, GPU inference wins by a factor of 10–20x. If the model is too large for VRAM and must be split across GPU and system RAM, performance drops significantly. Pure CPU inference on modern DDR5 is surprisingly usable for models up to 14B parameters — expect 8–15 tokens per second, which is perfectly readable.

▸ Budget Tiers

£200 TIER: "GETTING STARTED"

At this price, you're looking at the second-hand market or entry-level mini PCs.

Best option: Used office PC + GPU. A Dell OptiPlex or Lenovo ThinkCentre from eBay (£80–120) paired with a used GTX 1070 8GB or P40 24GB (£60–100) gets you into GPU-accelerated inference. The Nvidia Tesla P40 is a hidden gem — 24 GB of VRAM for around £80–100 used, though it needs an aftermarket cooler and a PSU with the right power connector. It's ugly, loud, and astonishingly capable for the money.

Alternative: Beelink EQ12 or similar N100 mini PC (£130–160). Intel N100, 16 GB DDR5. Good enough for 7B models on CPU at ~5–8 tokens/second. Fanless, tiny, sips power. Decent for experimentation but you'll outgrow it quickly.

£400 TIER: "THE SWEET SPOT"

This is where local LLM gets genuinely useful.

Minisforum UM780 XTX (~£350–400 barebones). AMD Ryzen 7 7840HS with Radeon 780M iGPU. Supports up to 64 GB DDR5-5600. The iGPU can accelerate inference via ROCm/Vulkan, and the CPU alone handles 7B–14B models comfortably at 10–15 tok/s. Two M.2 slots, USB4, compact form factor. This is the mini PC I'd recommend to most people.

Beelink SER8 (~£380–420). Same Ryzen 7 8845HS platform. Very similar performance to the UM780 XTX with slightly newer silicon. Excellent build quality. Either of these paired with 32 GB DDR5 (£60–80) gives you a capable local LLM box for under £500 all-in.

AOOSTAR GEM12 (~£350–400). AMD Ryzen 9 6900HX variant with OCuLink port — this is significant because it means you can add an external GPU later without Thunderbolt overhead. If you think you might want a GPU path in the future, this is the mini PC to buy.

£800 TIER: "NO COMPROMISES"

Mac Mini M2 Pro (16-core GPU, 32 GB) — available refurbished for £700–800. Apple's unified memory architecture is genuinely special for LLM inference. The 32 GB of unified memory is accessible to both CPU and GPU at 200 GB/s bandwidth. This means a 32 GB Mac can run models that would require 32 GB of VRAM on a discrete GPU — but at roughly 2–3x the bandwidth of DDR5. For models in the 14B–30B range, the Mac Mini offers the best tokens-per-pound ratio of any hardware at this price.

Mac Mini M4 Pro (24 GB) — from £599 new. Faster CPU and GPU than the M2 Pro, but only 24 GB in the base model. The 48 GB variant (£999) is the ultimate local LLM machine for most users — it runs quantised 70B models comfortably.

RTX 3060 12GB route — pair a used RTX 3060 12GB (£180–220) with a decent desktop (or your existing PC). 12 GB VRAM handles quantised models up to 13B parameters entirely on GPU at 30–50 tok/s. The 3060's memory bandwidth (360 GB/s) is modest by GPU standards, but still 5–7x faster than DDR5. Best price-to-VRAM ratio in the Nvidia lineup.

RTX 3090 24GB — the king of consumer local LLM, available used for £550–700. 24 GB GDDR6X at 936 GB/s. Runs quantised 30B models on GPU, and handles 70B models with partial CPU offloading. If you have a desktop with a 750W+ PSU and a case that fits a triple-slot card, this is the single best GPU you can buy for local inference. The RTX 4090 is faster but costs £1,600+ — the 3090 offers 80% of the performance at 40% of the price.

▸ The eGPU Path

If you've got a mini PC or laptop and want GPU acceleration without building a desktop, external GPUs are an option.

OCuLink is the preferred connection — it's essentially a direct PCIe x4 link with minimal overhead. The AOOSTAR GEM12 and some Minisforum models support it. Pair an OCuLink enclosure (£30–60) with a desktop GPU and you've got external GPU inference with roughly 85–90% of native PCIe performance.

Thunderbolt 3/4 eGPU enclosures (Razer Core X, Sonnet Breakaway) work but add latency and bandwidth constraints. For LLM inference — where you're streaming weights from VRAM, not doing real-time rendering — the Thunderbolt bottleneck is less painful than for gaming. Expect 70–80% of native GPU performance. Enclosures cost £150–250 used.

My recommendation: If you're building new, buy the AOOSTAR GEM12 with OCuLink and add a GPU when budget allows. If you're adding to an existing Thunderbolt laptop, it works but manage your expectations.

CHAPTER 03

Software Setup

▸ Ollama — The Easy Path

Ollama is where most people should start. It handles model downloading, quantisation selection, and serving in a single binary with a clean CLI.

Installation:

```
# Linux / WSL
curl -fsSL https://ollama.com/install.sh | sh

# macOS
brew install ollama

# Or download from https://ollama.com/download
```

Running your first model:

```
# Start the Ollama service (runs in background)
ollama serve

# Pull and run a model
ollama run llama3.1:8b

# For a smaller, faster model
ollama run phi3:mini

# For coding
ollama run deepseek-coder-v2:16b
```

That's it. You're running a local LLM. Type a prompt and it responds. Press Ctrl+D to exit.

Useful commands:

```
ollama list              # Show downloaded models
ollama pull qwen2.5:14b  # Download without running
ollama rm mistral:7b     # Delete a model
ollama show llama3.1:8b  # Show model details
ollama ps                # Show running models
```

Exposing the API:

Ollama automatically runs an OpenAI-compatible API on `http://localhost:11434`. Any tool that supports the OpenAI API can point at this endpoint.

```
# Test with curl
curl http://localhost:11434/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3.1:8b",
    "messages": [{"role": "user", "content": "Hello"}]
  }'
```

▸ llama.cpp — Maximum Control

If Ollama is the friendly wrapper, llama.cpp is the engine underneath. It's a C/C++ implementation of LLM inference that compiles to a single binary. Use it when you need fine-grained control over quantisation, context length, batch size, or GPU layer offloading.

```
# Build from source
git clone https://github.com/ggerganov/llama.cpp
cd llama.cpp
cmake -B build -DGGML_CUDA=ON  # For Nvidia GPU support
cmake --build build --config Release -j

# Run a GGUF model
./build/bin/llama-cli -m models/your-model.gguf -p "Your prompt" \
  -ngl 35    # Number of layers to offload to GPU
  -c 4096    # Context window size
  -t 8       # CPU threads
```

The `-ngl` flag is crucial for partial GPU offloading. If your model is too large for VRAM, offload as many layers as fit and the rest run on CPU.

▸ LM Studio — The GUI Option

LM Studio provides a polished desktop application for downloading, running, and chatting with local models. It's built on llama.cpp but wraps everything in a clean interface with a model browser, chat UI, and local API server.

Download from lmstudio.ai. Available for Windows, macOS, and Linux.

Best for: people who want a graphical interface, easy model browsing from Hugging Face, and a one-click local API server. It auto-detects your hardware and suggests compatible models.

▸ text-generation-webui (oobabooga)

The Swiss Army knife. A web-based interface supporting multiple backends (llama.cpp, ExLlamaV2, transformers, AutoGPTQ). More complex to set up but offers features like character cards, chat history, extensions, and multi-model management.

```
# One-line installer
git clone https://github.com/oobabooga/text-generation-webui
cd text-generation-webui
./start_linux.sh  # or start_windows.bat / start_macos.sh
```

▸ Docker Setups

For reproducible deployments, Docker is the way:

```
# Ollama in Docker
docker run -d -v ollama:/root/.ollama \
  -p 11434:11434 --name ollama ollama/ollama

# With GPU support (Nvidia)
docker run -d --gpus all -v ollama:/root/.ollama \
  -p 11434:11434 --name ollama ollama/ollama

# Open WebUI — a ChatGPT-like interface for Ollama
docker run -d -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data \
  --name open-webui ghcr.io/open-webui/open-webui:main
```

Open WebUI deserves special mention — it gives you a polished, multi-user chat interface with conversation history, model switching, RAG document upload, and web search integration. It's essentially self-hosted ChatGPT pointed at your local models.

CHAPTER 04

Model Selection

▸ The Model Landscape

The open-weight model ecosystem has exploded. Here's what matters as of early 2025:

Meta Llama 3.1 / 3.2 — The default recommendation. Available in 1B, 3B, 8B, and 70B sizes. The 8B model is the best general-purpose model at its size. The 70B competes with GPT-4 on many benchmarks. Excellent instruction following, strong reasoning. Start here.

Mistral / Mixtral — Mistral 7B punches well above its weight. Mixtral 8x7B uses a mixture-of-experts architecture — only 12B parameters are active per token, but it has 46B total, giving it broader knowledge. Good for multilingual tasks.

Qwen 2.5 — Alibaba's models. The Qwen 2.5 14B and 32B variants are outstanding — arguably the best open models at their respective sizes. Particularly strong at coding and structured output. The 72B model rivals Llama 70B. Don't sleep on these.

Microsoft Phi-3 / Phi-4 — Small but mighty. Phi-3 Mini (3.8B) delivers surprisingly coherent output for its size. Phi-4 (14B) competes with much larger models on reasoning benchmarks. Ideal when you're RAM-constrained.

Google Gemma 2 — Available in 2B, 9B, and 27B. The 9B and 27B models are strong generalists. Google's training data gives them broad knowledge, particularly for factual queries.

DeepSeek-V3 / DeepSeek-R1 — DeepSeek-R1 is the reasoning specialist. It "thinks" through problems step-by-step and excels at maths, logic, and coding. Available in distilled versions (1.5B to 70B). The 14B distilled version is exceptional for its size. Use this when you need chain-of-thought reasoning.

▸ Quantisation Explained

Models are trained in 16-bit floating point (FP16/BF16). A 7B parameter model at full precision needs ~14 GB. Quantisation reduces precision to shrink the model and speed up inference, with minimal quality loss.

GGUF is the standard format for llama.cpp and Ollama. When downloading models, you'll see quantisation labels:

QUANTISATION	BITS	SIZE (7B)	QUALITY IMPACT
Q2_K	2-bit	~2.5 GB	Significant degradation. Emergency only.
Q4_K_M	4-bit	~4.0 GB	Sweet spot. Minimal quality loss. Use this.
Q5_K_M	5-bit	~5.0 GB	Slightly better than Q4. Worth it if it fits.
Q6_K	6-bit	~5.5 GB	Near-original quality.

QUANTISATION	BITS	SIZE (7B)	QUALITY IMPACT
Q8_0	8-bit	~7.0 GB	Indistinguishable from FP16 for most tasks.
FP16	16-bit	~14 GB	Full precision. Rarely needed locally.

The rule: Use Q4_K_M unless you have a specific reason not to. It offers 95%+ of the original model's capability at roughly a quarter of the memory. If you have spare RAM/VRAM, step up to Q5_K_M or Q6_K.

▸ Task Recommendations

Coding: DeepSeek-Coder-V2 16B, Qwen 2.5 Coder 14B/32B, or Llama 3.1 8B. For pure code completion, the dedicated coding models are noticeably better than general-purpose models.

Creative writing: Llama 3.1 70B (if you can run it), Mistral 7B, or Qwen 2.5 14B. Larger models produce more coherent long-form text.

Chat / assistant: Llama 3.1 8B or Qwen 2.5 14B. Fast, capable, good instruction following.

RAG (retrieval-augmented generation): Qwen 2.5 14B or Llama 3.1 8B. You want good instruction following and accurate information extraction. Speed matters here since RAG involves frequent queries.

Reasoning / maths: DeepSeek-R1 (distilled 14B or 32B). Purpose-built for step-by-step reasoning.

▸ Right Model for Your RAM — Cheat Sheet

AVAILABLE MEMORY	BEST MODEL (Q4_K_M)	TOKENS/SEC (CPU)	TOKENS/SEC (GPU)
4 GB	Phi-3 Mini 3.8B	6–10	25–40
8 GB	Llama 3.1 8B	8–12	30–50
12 GB VRAM	Qwen 2.5 14B	—	25–40
16 GB	Qwen 2.5 14B	5–8	—
24 GB VRAM	DeepSeek-R1 32B (distilled)	—	20–35
32 GB	Qwen 2.5 32B	3–6	—
48 GB	Llama 3.1 70B	2–4	—

AVAILABLE MEMORY	BEST MODEL (Q4_K_M)	TOKENS/SEC (CPU)	TOKENS/SEC (GPU)
24 GB VRAM + 32 GB RAM	Llama 3.1 70B (partial offload)	—	8–15

CPU speeds assume DDR5-5600. GPU speeds assume RTX 3090. Your results will vary.

CHAPTER 05

Integration

▸ VS Code + Continue

Continue is an open-source AI coding assistant that plugs into VS Code and JetBrains IDEs. Point it at your local Ollama instance and you've got autocomplete, inline editing, and chat — all running on your hardware.

Setup:

1. Install the Continue extension from the VS Code marketplace.
2. Open Continue settings (Ctrl+Shift+P → "Continue: Open Config").
3. Add your local model:

```
{
  "models": [
    {
      "title": "Local Qwen Coder",
      "provider": "ollama",
      "model": "qwen2.5-coder:14b"
    }
  ],
  "tabAutocompleteModel": {
    "title": "Local Autocomplete",
    "provider": "ollama",
    "model": "qwen2.5-coder:7b"
  }
}
```

Use a smaller model (7B) for autocomplete (speed matters) and a larger model (14B+) for chat and inline edits.

▸ Obsidian

The Obsidian Smart Connections plugin enables semantic search across your vault using local embeddings. The Copilot plugin connects to Ollama for in-note AI assistance — summarisation, rewriting, Q&A against your notes.

Configure Copilot to use `http://localhost:11434` as the API endpoint and select your preferred model. Your notes never leave your machine.

▸ Terminal Tools

aider — AI pair programming in the terminal. Connects to local models via Ollama or any OpenAI-compatible API. Excellent for code refactoring and working across multiple files.

```
pip install aider-chat
aider --model ollama/qwen2.5-coder:14b
```

fabric — AI-powered CLI for summarising, extracting, and transforming text. Pipe anything into it.

```
cat article.md | fabric --pattern summarize --model ollama/llama3.1:8b
```

shell-gpt / mods — Ask questions from the command line, pipe in data, get structured output. Useful for scripts and automation.

▸ OpenAI-Compatible Endpoints

This is the key architectural insight: **most local inference servers expose an OpenAI-compatible API.** Ollama, LM Studio, llama.cpp server, vLLM, and text-generation-webui all do this.

This means any application, library, or service that supports the OpenAI API can be redirected to your local model by changing two things:

```
Base URL: http://localhost:11434/v1  (or your server's address)
API Key:  any-string-works           (local servers don't validate keys)
```

In Python with the OpenAI SDK:

```
from openai import OpenAI

client = OpenAI(
    base_url="http://localhost:11434/v1",
    api_key="not-needed"
)

response = client.chat.completions.create(
    model="llama3.1:8b",
    messages=[{"role": "user", "content": "Explain quantum computing"}]
)
```

This compatibility layer is powerful. You can swap between cloud and local models by changing a single environment variable. Build against the OpenAI API, deploy against your local server.

CHAPTER 06

Advanced Topics

▸ Fine-Tuning with LoRA

Fine-tuning adapts a pre-trained model to your specific data or task. **LoRA** (Low-Rank Adaptation) makes this feasible on consumer hardware by training a small set of adapter weights rather than modifying the entire model.

Tools: Unsloth is the fastest option — it reduces VRAM requirements by 60–70% and supports QLoRA (quantised fine-tuning). You can fine-tune a 7B model on a single 12 GB GPU in under an hour.

Use cases: teaching a model your company's coding style, training on domain-specific terminology (legal, medical), or creating a model that outputs in a specific format consistently.

Start with a strong base model, prepare 500–2,000 high-quality examples in the Alpaca or ChatML format, and train for 1–3 epochs. More data isn't always better — quality and diversity matter far more than volume.

▸ RAG Pipeline

Retrieval-Augmented Generation connects your LLM to external knowledge. The basic pipeline:

1. **Chunk** your documents into passages (500–1,000 tokens each).
2. **Embed** each chunk using a local embedding model (e.g., `nomic-embed-text` via Ollama).
3. **Store** embeddings in a vector database (ChromaDB, Qdrant, or pgvector).
4. **Query:** embed the user's question, find the most similar chunks, inject them into the prompt.
5. **Generate:** the LLM answers using the retrieved context.

Tools: LangChain and LlamaIndex handle the orchestration. For a simpler approach, Open WebUI has built-in RAG — upload documents and ask questions directly.

▸ Monitoring and Performance

Track your local inference with:

- `ollama ps` — shows loaded models and VRAM usage.
- `nvidia-smi` — GPU utilisation and memory (Nvidia).
- `nvtop` / `btop` — real-time hardware monitoring.
- Tokens per second is shown in Ollama's output — watch this number. If it drops below 5 tok/s, your model is too large for your hardware.

CHAPTER 07

Quick Reference Appendix

▸ Hardware Comparison Table

HARDWARE	PRICE (£)	MEMORY	BANDWIDTH	BEST MODEL SIZE	TOK/S (EST.)
Intel N100 Mini PC	130–160	16 GB DDR5	38 GB/s	7B	5–8
Minisforum UM780 XTX	350–450*	32–64 GB DDR5	77 GB/s	14B–32B	8–15
Beelink SER8	380–450*	32–64 GB DDR5	77 GB/s	14B–32B	8–15

HARDWARE	PRICE (£)	MEMORY	BANDWIDTH	BEST MODEL SIZE	TOK/S (EST.)
Mac Mini M2 Pro 32GB	700–800	32 GB Unified	200 GB/s	14B–30B	12–20
Mac Mini M4 Pro 48GB	999	48 GB Unified	273 GB/s	30B–70B	15–25
Desktop + RTX 3060 12GB	400–550	12 GB VRAM	360 GB/s	8B–13B (GPU)	30–50
Desktop + RTX 3090 24GB	700–900	24 GB VRAM	936 GB/s	14B–30B (GPU)	40–70
Tesla P40 (used)	80–100	24 GB VRAM	346 GB/s	14B–30B (GPU)	15–25

Barebones price; add RAM and storage.

▸ Model Compatibility Matrix

MODEL	Q4 SIZE	8 GB RAM	16 GB	24 GB VRAM	32 GB	48 GB	BEST FOR
Phi-3 Mini 3.8B	2.2 GB	▯	▯	▯	▯	▯	Edge, mobile, quick tasks
Llama 3.1 8B	4.7 GB	▯	▯	▯	▯	▯	General purpose, chat
Gemma 2 9B	5.4 GB	⚠	▯	▯	▯	▯	Factual queries, QA
Qwen 2.5 14B	8.9 GB	▯	▯	▯	▯	▯	Coding, structured output
Mistral Nemo 12B	7.1 GB	▯	▯	▯	▯	▯	Multilingual, general
DeepSeek-R1 14B	8.9 GB	▯	▯	▯	▯	▯	Reasoning, maths

MODEL	Q4 SIZE	8 GB RAM	16 GB	24 GB VRAM	32 GB	48 GB	BEST FOR
Qwen 2.5 32B	19 GB						Best mid-range all-rounder
DeepSeek-R1 32B	19 GB						Advanced reasoning
Llama 3.1 70B	40 GB						Near-GPT-4 performance
Mixtral 8x7B	26 GB						Broad knowledge, multilingual

 = *runs well* | ⚠ = *tight, may be slow* | = *won't fit*

▸ Troubleshooting

Model loads but generation is extremely slow (<2 tok/s) Your model is too large for available memory and is swapping to disk. Solution: use a smaller model or a more aggressive quantisation (Q4_K_M instead of Q8).

CUDA out of memory Reduce the number of GPU layers (`-ngl` in llama.cpp) or use a smaller quantisation. In Ollama, try a smaller model variant.

Ollama can't find my GPU Ensure Nvidia drivers are installed (`nvidia-smi` should work). For AMD GPUs, you need ROCm installed. On macOS, Metal acceleration is automatic.

"Template not found" or garbled output You're likely using a base model instead of an instruct/chat variant. Always use the `-instruct` or `-chat` tagged versions for conversational use.

Context window exceeded Default context is often 2048–4096 tokens. Increase it: `ollama run llama3.1:8b --num-ctx 8192`. Note: larger context uses more memory.

Docker container can't access GPU Install the Nvidia Container Toolkit: `sudo apt install nvidia-container-toolkit` and restart Docker. Use `docker run --gpus all`.

Model quality seems poor compared to ChatGPT Local models are smaller. Adjust expectations: a local 8B model ≈ GPT-3.5 level. For GPT-4 level, you need 70B+. Also ensure you're using the instruct/chat variant and an appropriate system prompt.

Performance tips: - Close other applications to free RAM - Use `mlock` to prevent model from being swapped: set `OLLAMA_KEEP_ALIVE=-1` - On multi-GPU systems, specify which GPU: `CUDA_VISIBLE_DEVICES=0` - Flash attention can reduce VRAM usage: enable with `--flash-attn` in llama.cpp

This guide is maintained and updated regularly. You own it forever — including all future updates. Questions? Reach out via the email on your Gumroad receipt.

Last updated: February 2025

The Local LLM Handbook