



GOING LOCAL

Own your stack.

Cogito Ergo Creo · OpenClaw Bulgaria #2

by Nikola

| Why local has always made sense

Data ownership predates the headlines

Your code.
Your data.
Your stack.

— *always was*

Local inference keeps your code, your data, and your operating constraints under your own roof. That was true on April 4. It was true on April 3. It will be true tomorrow.

Recent headlines don't make the case.

They make it visible.

Where the dial has been turning

Recent examples, with timestamps

APR 4, 2026

Anthropic blocks Pro/Max from third-party agents

Subscriptions stop working through OpenClaw, opencode, and similar harnesses. Console access unchanged. API access at API rates.

Boris Cherny, Anthropic (announcement)

APR 7, 2026

Anthropic ships Claude Mythos – not for general release

Most capable Claude model. Available only to a limited group of cybersecurity partners via Project Glasswing. "We do not plan to make Mythos Preview generally available."

red.anthropic.com

APR 16, 2026

Opus 4.7 ships new tokenizer – same rate card, bigger bill

Anthropic confirms up to 35% more tokens for the same text. OpenRouter measured 12–27% real-world cost increase.

Anthropic pricing docs · OpenRouter

ONGOING

Models retired on 3–6 month cycles

Claude 3.7 Sonnet shut down May 11 (Vertex). GPT-4o retired Feb 13–17 (ChatGPT + API). Six-month deprecation windows are now standard.

OpenAI · Anthropic · Google Cloud deprecation notices

Waiting for Anthropic outage to pass be like:



The economics underneath

Where the pressure comes from

\$14B

OpenAI projected 2026 loss

The Information · WSJ

\$1.69

spent for every \$1 earned

WSJ leaked docs · Fortune

\$121B

OpenAI compute spend by 2028

WSJ, April 2026

WSJ

APR 2026

Confidential financials, leaked to investors

OpenAI projects \$121B compute spend in 2028, \$85B cash burn that year — even after revenue nearly doubles in 2027. "Money going out is still far greater than money coming in."

FORTUNE

NOV 2025

"Stunning annual losses through 2028"

Per internal documents: OpenAI burn rate 57% of revenue through 2027, profit not before 2030. Anthropic drops to 9% by 2027, expects breakeven 2028.



The recurring theme

Same pattern, every provider

Terms change.

Closed providers can re-price, re-route, or re-rule overnight.

Your stack is rented.

Cache, quotas, latency, model versions — all decided elsewhere.

Tomorrow's terms are unknown.

Today's allowed workflow may not be tomorrow's.

What going local buys you

The case for owning the stack



Privacy

Your code never leaves your network.



Sovereignty

No mid-project policy changes.



No limits

No rate caps, no daily quotas.



Repeatability

Frozen weights. No silent version drift.



Fine-tuning

Specialize on your codebase.



Marginal \$0

Free after the hardware investment.

What it actually costs

The honest trade-offs



Quality gap

Frontier models still win on hard tasks.



Hardware

€1–5k+ to get seriously useful.



Setup tax

Model picking, tuning, troubleshooting.



Power & heat

350–450W GPUs run hot and loud.



You're SRE now

Updates, crashes, OOMs — all yours.



Context limits

VRAM caps your usable context window.

Setting expectations

Local wins some, loses some



LOCAL HANDLES THIS

- Focused code edits & refactors
- Autocomplete-style completion
- Boilerplate / scaffolding
- Classification & extraction
- Code review on small diffs
- Doc generation
- Deterministic, repeatable tasks



STILL USE THE CLOUD

- Complex multi-step reasoning
- Very long contexts (200k+)
- Novel / niche domains
- Aggressive agentic loops
- Ambiguous instructions
- First-of-a-kind problems
- Tasks demanding peak accuracy

The 2026 open-weight landscape

What's actually worth running

Gemma 4

26B-A4B / 31B

Google's local star — Apache 2, 256K context, frontier-per-param

Qwen 3.6-35B-A3B

35B-A3B

Hybrid MoE, 256K context, multimodal — Apache 2.0

DeepSeek V4

MoE w/ DSA

Best perf-per-cost when self-hosted; sparse attention

GLM-4.7-Flash

30B-A3B

Built for consumer GPUs — 60–100 tok/s on a 3090, MIT

Llama 4 Scout

109B-A17B

Meta's MoE — 10M context, partial-offload on Apple Silicon

Mistral Small 4

Small + Large 3

European, code-tuned, strong instruction-following

Capability per GB of VRAM has been climbing fast. What was "big" two years ago fits on a 16GB card today.



The inference engine landscape

Pick the one that matches your style

LM Studio

GUI

Beginner-friendly, OpenAI server built-in. We use this today.

llama.cpp

ENGINE

The C++ engine under most local setups. CLI master.

Ollama

CLI

Simple CLI, registry of pre-quantized models. Great for scripts.

vLLM

PRODUCTION

Throughput-focused. Multi-user serving, batching.

MLX-LM

APPLE

Apple Silicon native, faster than GGUF on M-series.

text-generation-webui

KITCHEN SINK

Oobabooga. Many backends, many knobs.

Why LM Studio for today

Lowest possible friction



Cross-platform GUI (macOS, Windows, Linux)



Built-in Hugging Face model browser



OpenAI-compatible server out of the box



Native MLX support on Apple Silicon



Live model swap without re-config



Zero terminal knowledge required to start

Hardware reality check

What you can run, by memory budget

MEMORY	FITS	WHAT YOU GET
8 GB VRAM	Gemma 4 E4B, Phi-4-mini	Lightweight tasks, autocomplete-class
12 GB VRAM	Gemma 4 26B-A4B (offload)	Big-model speed via MoE, partial GPU
16 GB VRAM	Gemma 4 26B-A4B, Qwen 3.6	Sweet spot for budget builds
24 GB (3090/4090)	Gemma 4 31B, Mistral Small 4	Serious local coding rig
32-64 GB Apple	Llama 4 Scout (109B-A17B)	Unified memory unlocks big MoEs
48 GB+ VRAM	GLM-5.1, big MoE variants	Approaching cloud quality on focused tasks

NVIDIA vs Apple Silicon

Two paths up the mountain



NVIDIA discrete GPU

- + Highest raw tokens/sec
- + CUDA ecosystem dominance
- + Cheaper per GB at consumer tier
- VRAM is the hard ceiling (24–32 GB)
- Power-hungry (350–450W)
- Loud, hot, large case needed



Apple Silicon (M-series)

- + Unified memory up to 192 GB
- + Power-efficient (40–80W)
- + MLX format gives a big speedup
- + 70B-class on a laptop, silently
- Slower raw throughput than 4090
- Fewer tools, smaller ecosystem

Where the work happens

GPU, CPU, and the hybrid in between

ALL ON GPU

50–100+ tok/s

Entire model fits in VRAM. The fast path.

PARTIAL OFFLOAD

10–30 tok/s

Some layers in VRAM, rest in system RAM. Most common for big models on consumer GPUs.

ALL ON CPU

2–10 tok/s

Slow but possible. Fine for small models or batch tasks.

Knob: **GPU offload layers** in LM Studio · **-ngl N** in llama.cpp. Push it up until you run out of VRAM.

Model formats

GGUF, MLX, or raw weights

GGUF

Default on NVIDIA

- llama.cpp ecosystem
- Runs on CPU + GPU + hybrid
- Every model published as GGUF
- Many quantization choices

MLX

Apple Silicon

- Native to M-series chips
- Noticeably faster than GGUF on Mac
- Fewer models published
- LM Studio handles it transparently

Safetensors / FP16

Servers, training

- Raw weights, no quant
- vLLM, transformers, fine-tuning
- Huge — full precision
- Skip for laptop/desktop use

Quantization

The biggest single lever you control

30B model

FP16

60 GB

↓ Q4_K_M

~17 GB

Fits a 24GB card.

QUANT	QUALITY	SIZE	VERDICT
Q8_0	99%	~1.0 B/param	Maximum quality
Q6_K	98%	~0.75 B/param	If you have headroom
Q5_K_M	97%	~0.6 B/param	Quality-leaning
Q4_K_M	95%	~0.5 B/param	★ Sweet spot
Q3_K_M	90%	~0.4 B/param	Noticeable degradation
Q2_K	<85%	~0.3 B/param	Last resort

Rule of thumb: pick the largest model whose Q4_K_M fits in your VRAM.

Architecture: Dense vs MoE

Same memory, very different speed

DENSE

Gemma 4 31B, Qwen 3.6 dense, Mistral Small 4

Every parameter activates for every token.

Active per token: **100%**

Memory: all params loaded

Speed: scales with total size

31B dense → every param touched, every token.

MIXTURE OF EXPERTS

Gemma 4 26B-A4B, Llama 4 Scout, GLM-5.1, Kimi K2.6

Subset of "experts" activates per token.

Active per token: **~15%**

Memory: still all params loaded

Speed: near-active-size speed

Gemma 4 26B-A4B → 26B quality, 4B-class speed.

Context length & KV cache

The silent VRAM killer

KV cache scales linearly with context length.

Doubling your context window roughly doubles the memory it eats. On a 30B model at FP16 KV cache, 32k context can consume 8–12 GB on top of weights.

Start small

Set context to 8k or 16k first. Bump only when a task actually needs it.

Quantize the KV cache

Q8 KV cache is nearly free in quality, halves memory. Q4 if desperate.

Flash Attention

Reduces memory pressure during attention compute. Turn it on if available.

Settings that actually matter

Default tune for any new model

- | | |
|-----------------------------|---|
| ✓ Format | GGUF on NVIDIA, MLX on Apple Silicon |
| ✓ Quantization | Q4_K_M (sweet spot); Q5_K_M / Q6_K if you have headroom |
| ✓ GPU offload layers | Max out until just before OOM |
| ✓ Context length | Start at 8k–16k; grow only as needed |
| ✓ KV cache quant | Q8_0 (nearly free quality, halves cache memory) |
| ✓ Flash Attention | On |
| ✓ Architecture pick | Prefer MoE if it fits memory — same RAM, more speed |



LIVE

From zero to OpenClaw running on a local model

1 Install LM Studio, load a model

2 Start the server (port 1234)

3 Run openclaw onboard

Pointing OpenClaw at LM Studio

5 commands — onboarding writes the config for you

● ● ● zsh — openclaw onboarding

1. install LM Studio

```
curl -fsSL https://lmstudio.ai/install.sh | bash
```

2. start the server

```
lms server start --port 1234
```

3. token (any non-empty value if unauthed)

```
export LM_API_TOKEN="placeholder-key"
```

4. onboard — pick LM Studio + a model

openclaw onboard

5. swap models later, no re-onboard

```
openclaw models set lmstudio/qwen/qwen3.5-9b
```

HOW IT WORKS

openclaw onboard

Interactive wizard. Writes the provider config for you — no JSON editing.

Model key format

author/model-name. OpenClaw refs prepend the provider: lmstudio/<key>.

Find exact keys

curl localhost:1234/api/v1/models — read the "key" field.

Enable JIT in LM Studio

Loads models on first request. Avoids "model not loaded" errors.



Performance on this rig

Real numbers from the demo

RIG: RTX 3090 24GB · Gemma 4 26B-A4B · Q4_K_M · 16k context · Q8 KV

XX

tok/s

Decode speed

Sustained generation

XX

tok/s

Prompt processing

Initial pass through context

X.X

s

Time to first token

From submit to first byte

XX

GB

VRAM used

Model + KV cache



When to still reach for the cloud

The 80/20 honesty slide



Complex multi-step reasoning chains



Maxing out 1M+ contexts (KV cache budget)



Novel or niche domains outside training



Agentic loops with non-standard tools



Ambiguous instructions needing inference



First-of-a-kind, high-stakes tasks

Local is for the 80%. Keep an API key for the 20%.

Your turn

Bring laptop, project, questions



Wire it up live

Get OpenClaw + a local model running on your laptop. We'll help.



Local vs cloud

Run the same task on local and cloud — compare the outputs.



Debug a slow workflow

Bring something that's underperforming. We diagnose together.



Open discussion

MCP servers locally, memory plugins, fine-tuning, whatever you want.

Resources

Go deeper after the talk

LM Studio

lmstudio.ai

Today's demo tool

llama.cpp

github.com/ggerganov/llama.cpp

The engine under everything

Ollama

ollama.com

CLI-friendly alternative

Hugging Face GGUF

huggingface.co/models?library=gguf

Where to find quantized models

MLX-LM

github.com/ml-explore/mlx-lm

Apple Silicon native

r/LocalLLaMA

reddit.com/r/LocalLLaMA

Community pulse, hardware guides

Thanks. Go build. 🦊