Running OpenClaw on 4GB, 6GB, and 8GB GPUs: What Actually Works

March 5, 2026 · by Mark Bartlett

Download this guide as PDF

Quick Answer: 4GB VRAM can't run OpenClaw locally with any reliability -- use cloud API mode instead. 6GB can run Qwen 3.5 4B for simple single-skill tasks, but tool calling fails often. 8GB is the real floor: Qwen 2.5 Coder 7B at Q4 (~5GB) handles basic agent workflows with room to spare. Below 8GB, you're better off running OpenClaw in gateway mode with Claude or GPT-4 -- the gateway itself needs no GPU.

More on this topic: OpenClaw Setup Guide · VRAM Requirements · Best Local Coding Models · OpenClaw Token Optimization

OpenClaw is lightweight. The gateway runs on a Raspberry Pi. The problem isn't OpenClaw itself – it's the local model behind it.

AI agent tasks are harder than chat. The model has to produce valid JSON tool calls on every turn, keep track of a multi-step plan, and not hallucinate functions that don't exist. Small models fail at all of this. Bigger models handle it, and bigger models need more VRAM.

So what happens when you only have 4, 6, or 8 GB?

The Quick Reference

VRAM	Best Model for OpenClaw	Size (Q4)	Tok/s	Tool Calling	Verdict
4 GB	Qwen 3.5 2B	~1.5 GB	20-30	Unreliable	Use cloud API instead
6 GB	Qwen 3.5 4B	~2.5 GB	25-35	Single-step only	Marginal – simple skills work
8 GB	Qwen 2.5 Coder 7B	~5 GB	30-40	Solid	First usable tier
8 GB	Qwen 3.5 9B	~6.6 GB	25-35	Solid	Better reasoning, tighter fit

Token speeds assume NVIDIA consumer GPUs (RTX 3060 Ti / 4060 class). AMD will be similar or slightly slower depending on the card.

Why Agent Tasks Are Harder Than Chat

Regular chat is forgiving. If a model phrases something awkwardly, you still understand it. Agent tasks aren't like that. When OpenClaw asks a model to call a tool, the response must follow a strict JSON format. One wrong bracket and the skill call fails.

Three things break at low VRAM:

Tool-call formatting. Small models produce malformed JSON more often. A 3B model might return `search("weather")` instead of the structured `{"tool": "search", "args": {"query": "weather"}}` that OpenClaw expects. The tool-call failure guide covers recovery, but prevention is better.

Multi-step planning. OpenClaw chains skills together – search the web, extract data, format a response, send it. Each step requires the model to remember what came before and decide what comes next. Models under 7B lose the thread after 2-3 steps.

Context length. OpenClaw works best with 64K+ context for multi-step tasks. On low VRAM, you'll be running 4K-8K context to fit the model at all. That means shorter conversation history, fewer skill results in memory, and more frequent context-window overflows.

4GB VRAM: Be Honest With Yourself

GPUs: GTX 1650, GTX 1050 Ti, some laptop 3050s.

After the OS and display compositor take their share, you have roughly 3-3.5 GB available for inference. That limits you to models under 3B parameters at Q4 quantization.

What Fits

Model	Size (Q4)	Context	Notes
Qwen 3.5 2B	~1.5 GB	4K max	Best option here. Multimodal, thinking mode
Qwen 2.5 Coder 1.5B	~1 GB	4K max	Has FIM, but too small for reliable tool calls

Model	Size (Q4)	Context	Notes
Phi-4 Mini (3.8B)	~2.3 GB	4K max	Tight fit, needs browser closed

What actually happens

4GB isn't enough for local OpenClaw. The models that fit are too small to reliably format tool calls. You'll get maybe 40-50% success on single-step skills (web search, read a file) and near-zero on multi-step chains. The model hallucinates tool names, forgets the JSON schema mid-response, and loses context within a few turns.

What to do instead: Run OpenClaw in gateway mode with a cloud API.

```
npx openclaw@latest
# During setup wizard, choose Claude or GPT-4 as your LLM provider
```

The OpenClaw gateway itself uses almost no resources – a few hundred MB of RAM, no GPU. Your 4GB card can handle everything else on the machine while Claude or GPT-4 does the reasoning. At $0.003/1K tokens (Claude Haiku) or $0.01/1K (Claude Sonnet), a moderate-use OpenClaw setup costs $5-15/month. That's less than the electricity cost of running a GPU 24/7.

6GB VRAM: Simple Skills, Short Chains

GPUs: RTX 3050 (6GB), RTX 4050 laptop, some GTX 1660 variants.

You have roughly 4.5-5 GB available after system overhead. This opens up the 3B-4B model tier.

What Fits

Model	Size (Q4)	Context	Tool Calling
Qwen 3.5 4B	~2.5 GB	8K	Best you'll get here
Qwen 2.5 Coder 3B	~2 GB	8K	FIM support, weaker reasoning
Qwen 3 4B	~2.5 GB	8K	Solid, no multimodal

Setup

```
ollama pull qwen3.5:4b
```

Then point OpenClaw at your local Ollama instance during setup:

```
npx openclaw@latest
# Choose Ollama as LLM provider → http://localhost:11434
# Select qwen3.5:4b as your model
```

What Works

Qwen 3.5 4B is the best you'll get at 6GB. It has thinking mode (enable it – it helps with tool formatting), 262K native context (though you'll cap at 8K on this VRAM), and native multimodal if you need image understanding.

With 8K context and Qwen 3.5 4B, expect:

- **Single-skill calls:** ~65-70% success rate. Web search, file read, simple calculations. Usable if you can tolerate retries.
- **Two-step chains:** ~40-50% success. Search then summarize, read then format. Fails often enough to be frustrating.
- **Three+ step chains:** Below 30%. The model loses track of the plan. Not practical.

What Doesn't Work

Long conversations. Browser automation (needs context for page content). Complex skill orchestration. Anything that requires the model to hold multiple tool results in context simultaneously.

My take

6GB with Qwen 3.5 4B is fine for learning how OpenClaw works. Don't rely on it for real tasks. If you're retrying every other skill call, that's not a bug. That's the model.

8GB VRAM: The Real Starting Line

GPUs: RTX 3060 Ti, RTX 3070, RTX 4060, RX 6600 XT.

This is where local OpenClaw goes from “technically runs” to “actually useful.” You have 6-7 GB available for inference, which fits 7B-9B models with room for context.

What Fits

Model	Size (Q4)	Context	Best For
Qwen 2.5 Coder 7B	~5 GB	8-16K	Tool calling + FIM
Qwen 3.5 9B	~6.6 GB	4-8K	Reasoning, multimodal
DeepSeek Coder V2 Lite	~5 GB	8-16K	Reasoning-heavy tasks
DeepSeek R1 Distill 8B	~4.5 GB	8-16K	Step-by-step reasoning

The Two Best Options

For tool calling: Qwen 2.5 Coder 7B. The pick for most OpenClaw users at 8GB. Good instruction following, handles structured JSON output well, supports FIM if you’re running coding skills, and at ~5GB Q4 it leaves 2GB+ for context and KV cache. You can push context to 16K without memory pressure.

```
ollama pull qwen2.5-coder:7b
```

For harder reasoning: Qwen 3.5 9B. Better at planning and multi-step logic. Thinking mode produces noticeably better tool-call formatting when you turn it on. But at 6.6GB Q4, it’s a tighter fit. Cap context at 4-8K and close your browser to free VRAM. Worth it if your OpenClaw skills involve analysis or multi-step reasoning.

```
ollama pull qwen3.5:9b
```

You don’t need both loaded at once. Ollama swaps models automatically – one runs while the other stays on disk.

What Works at 8GB

- **Single-skill calls:** 80-85% success rate with Qwen 2.5 Coder 7B. Reliable enough for daily use.
- **Two-step chains:** 65-70%. Needs occasional retries but gets the job done.
- **Three-step chains:** 50-60%. Hit or miss, fine for non-critical tasks.
- **Simple coding skills:** Autocomplete, explain code, write tests. Good.
- **Web search + summarize:** Reliable at 8K context.

What Still Struggles

- **Long multi-step chains (5+):** Context fills up and the model loses earlier results. Use token optimization to mitigate.
- **Browser automation:** Full page content in context eats your available window fast.
- **Parallel skill execution:** Running two skills simultaneously doubles memory pressure. Stick to sequential.

Context Length: The Hidden Bottleneck

OpenClaw's documentation recommends 64K context for multi-step tasks. At 8GB VRAM with a 7B model, you're running 8-16K. That's enough for simple workflows but not for the complex chains that make OpenClaw impressive.

This matters more than raw model intelligence. A smarter model at 4K context loses to a slightly dumber model at 16K because the smarter one literally can't see the earlier tool results. Prioritize context headroom:

```
# In your Ollama Modelfile or via API
PARAMETER num_ctx 16384    # Push to 16K if model fits
```

If you're bumping into context limits, see our num_ctx VRAM overflow guide – setting context too high silently kills performance.

Partial GPU Offload: The Middle Ground

If your model doesn't fully fit in VRAM, Ollama splits layers between GPU and CPU. This is better than pure CPU but worse than full GPU.

How It Works

Ollama does this automatically. If a model needs 6GB but you have 4GB free, it loads as many layers as fit on the GPU and puts the rest in system RAM. `ollama ps` shows the split:

```
NAME                SIZE    PROCESSOR          UNTIL
qwen2.5-coder:7b    5.3 GB  60% GPU/40% CPU    4 minutes from now
```

Is It Worth It?

Setup	Speed (7B Q4)
Full GPU (8GB card)	30-40 tok/s
60% GPU / 40% CPU	12-18 tok/s
30% GPU / 70% CPU	6-10 tok/s
Full CPU	3-6 tok/s

Partial offload at 60/40 is tolerable for OpenClaw. Agent tasks involve waiting for API calls and skill execution anyway, so 15 tok/s doesn't feel as painful as it would in interactive chat. Below 50% GPU, you're not getting much benefit over pure CPU.

The tradeoff: If you're offloading more than half the layers to CPU, you're better off with a smaller model that fits entirely on the GPU. A 3B model at 35 tok/s will complete tasks faster than a 7B model at 8 tok/s, even if the 7B is smarter per-token.

CPU-Only: When You Have No GPU

Running OpenClaw with no GPU at all? It works, but calibrate your expectations.

Model	CPU Speed (good desktop)	Viable?
Qwen 3.5 2B	8-12 tok/s	Simple skills only
Qwen 3.5 4B	5-8 tok/s	Slow but functional
Qwen 2.5 Coder 7B	3-5 tok/s	Painfully slow

At 3-5 tok/s, a 7B model takes 30-60 seconds to generate a tool-call response. That's annoying in chat but manageable for background agent tasks where you're not watching every token. If you're running OpenClaw as a background service on a headless box, CPU-only with a 4B model is a legitimate (if slow) option.

You'll want plenty of system RAM though. The model sits in RAM instead of VRAM, and the KV cache grows there too. Budget 8-10 GB of RAM for a 7B model at Q4 with 8K context.

The Cloud API Escape Hatch

If you've read this far and your GPU is 6GB or less, the practical move is hybrid mode: run the OpenClaw gateway locally, point it at a cloud API for reasoning.

```
npx openclaw@latest
# Choose Claude as LLM provider
# Use claude-3-haiku for cheap background tasks
# Use claude-3-sonnet for complex multi-step tasks
```

Your code, skills, and data stay on your machine. Only the prompts and responses go through the API. The tool-call success rate jumps to 95%+ because frontier models don't struggle with structured output.

What it costs: Claude Haiku at $0.25/million input tokens handles most simple agent tasks for pennies. A moderately active OpenClaw instance runs $3-8/month on Haiku. Cheaper than the electricity to keep a GPU spinning.

If the privacy angle is what drew you to local models, consider this: your skill outputs (file reads, web scrapes, database queries) go into the prompt. That's the data you might care about. OpenClaw's token optimization can reduce how much context you send to the API, and you can use the model routing system to send sensitive tasks to a small local model while routing complex tasks to the cloud.

Which Setup Should You Pick?

Your Situation	Recommendation
4GB GPU, care about privacy	CPU-only with Qwen 3.5 4B. Slow but local
4GB GPU, want it to work	Cloud API (Claude Haiku)

Your Situation	Recommendation
6GB GPU, tinkering	Qwen 3.5 4B on GPU. Learn the system, expect rough edges
6GB GPU, need reliability	Cloud API + local gateway
8GB GPU	Qwen 2.5 Coder 7B locally. This actually works
8GB GPU, harder tasks	Qwen 3.5 9B locally. Tighter fit, better reasoning
8GB + want best of both	Local model for simple skills, cloud API for complex chains

Bottom Line

The minimum for local OpenClaw that doesn't make you want to throw the computer out the window is 8GB VRAM with Qwen 2.5 Coder 7B. That gets you 80%+ tool-call success on single skills and workable performance on short chains.

Below 8GB, the models that fit can't reliably handle structured tool calling. You'll spend more time debugging failed skill calls than using the agent. At that point, the cloud API isn't a compromise. It's the better tool for the job.

If you're building toward a local setup, the upgrade path is clear: 8GB gets you started, 16GB opens up 14B models with much better reasoning, and 24GB lets you run Qwen 2.5 Coder 32B which matches cloud models on most tasks.

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/openclaw-low-vram-gpus/

Free guides for running AI locally