OpenClaw on Raspberry Pi: What Actually Works (and What Doesn't)

March 5, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** The Raspberry Pi 5 (8GB) runs OpenClaw well as a gateway with cloud APIs like Claude or GPT-4 — the gateway itself only needs 2-4 GB RAM. Running local LLMs on the Pi is a different story: expect 2-7 tok/s on 1.5B-3B models via Ollama or llama.cpp. That's too slow for complex agent tasks but works for simple queries, offline setups, and air-gapped environments. Pi 4 works for gateway-only mode but can't handle local models at all. Get the 8GB Pi 5, an NVMe SSD, active cooling, and the 27W power supply. Use Qwen 2.5 3B or Gemma 2B for local inference. For anything serious, point OpenClaw at a cloud API and use the Pi as a $100 always-on agent server.

📚 **More on this topic:** OpenClaw Setup Guide · OpenClaw Security Guide · Best Models for OpenClaw · Ollama Troubleshooting

---

Running OpenClaw on a Raspberry Pi is one of those projects that sounds ridiculous until you actually do it. A $80 single-board computer running an AI agent that manages your messages, searches the web, and writes scripts? It works. With caveats.

Two things are true at once. The Pi 5 makes a solid OpenClaw gateway — it routes messages between you and a cloud LLM, runs 24/7 on 5-8 watts, and costs about $5 a year in electricity. That part is practical and I'd recommend it to anyone. Running local LLMs on the Pi is a different conversation. You'll get 2-7 tokens per second on tiny models. That's a learning project, not a productivity setup. I did both, and I'm glad I did.

Pi 5 vs Pi 4: Only One Real Option

The Pi 5 is the only viable choice for OpenClaw with local models. The Pi 4 works as a gateway-only box (pointing at Claude or GPT-4 via API), but the moment you try running a local model, the Pi 4 falls apart.

Spec	Pi 4 (8GB)	Pi 5 (8GB)
CPU	Cortex-A72, 1.8 GHz	Cortex-A76, 2.4 GHz
Memory bandwidth	~4 GB/s	~8.5 GB/s
PCIe	None	PCIe 2.0 x1 (NVMe)
LLM inference	Unusable (swap thrashing)	2-7 tok/s on 1.5B-3B
OpenClaw gateway	Works (barely)	Works well
Price	~$55	~$80
Verdict	Gateway-only, if you already own one	The real option

The Pi 5's Cortex-A76 cores are roughly 2-2.5x faster per clock than the Pi 4's Cortex-A72, and the doubled memory bandwidth is the real difference for LLM inference, where the bottleneck is reading model weights from RAM. On a Pi 4, even a 1.5B model triggers enough swap to make the system unusable. Multiple community reports confirm this: don't try to run Ollama on a Pi 4.

Get the 8GB model. The 4GB Pi 5 exists, but a 3B model at Q4 quantization needs around 2.5-3 GB just for weights, leaving nothing for the OS, OpenClaw's gateway process, or KV cache. The 8GB model costs $15 more and makes everything feasible.

The Full Hardware List

A complete OpenClaw Pi setup runs about $130-140:

Component	Price	Why
Raspberry Pi 5 (8GB)	$80	The brains
NVMe HAT + 256GB SSD	$25-35	SD cards are too slow for OpenClaw's SQLite writes
27W USB-C power supply	$12	Third-party chargers trigger under-voltage warnings
Active cooling case	$10-12	Non-negotiable — sustained inference causes throttling
Total	**~$130-140**	

The NVMe SSD is the upgrade that matters most after the Pi itself. OpenClaw hammers its SQLite memory database and log files with constant small reads and writes. On an SD card, the agent feels sluggish even with a cloud API backend. On NVMe (400-600 MB/s), it's responsive.

The official Raspberry Pi M.2 HAT paired with any budget 2242 NVMe drive is the single biggest quality-of-life upgrade you can make.

Annual electricity cost at 5-8 watts: about $4-6. Compare that to keeping an old laptop running 24/7.

Local LLM Performance: Setting Expectations

I ran these models and cross-referenced with testing by Stratosphere Labs and community benchmarks. Numbers on a Pi 5 (8GB) with Ollama:

Model	Parameters	RAM Used	Speed	Quality
Gemma 2 2B	2B	~3 GB	~7 tok/s	Good for simple tasks
Qwen 2.5 1.5B	1.5B	~2.5 GB	~8 tok/s	Efficient, good reasoning for size
Qwen 2.5 3B	3B	~5.4 GB	~5 tok/s	Best quality that fits comfortably
Nemotron-mini 4B	4B	~4 GB	~4 tok/s	Solid quality
Phi 3.5 3.8B	3.8B	~5 GB	~3 tok/s	Hallucinates frequently
Mistral 7B	7B	~5 GB (Q4)	~1 tok/s	Barely functional, heavy swap

For context: Claude on a cloud API responds at effectively infinite speed for the gateway. A desktop with an RTX 3060 runs the same Qwen 2.5 3B at 80+ tok/s. The Pi is roughly 10-20x slower than even a modest GPU setup.

What 3-5 tok/s actually feels like: You type a question, wait 10-30 seconds for a paragraph response. Short factual answers come back in 5-10 seconds. Multi-paragraph responses take over a minute. It's like texting someone who types slowly. Usable for simple queries. Painful for anything complex.

The Models That Actually Work

Stick to 1.5B-3B models. Anything larger either won't fit in RAM or swaps so aggressively that you'll get sub-1 tok/s.

Qwen 2.5 3B is what I'd pick. Best quality-to-size ratio, fits in 8GB with room for the OS and OpenClaw, and handles basic tool-calling for simple agent tasks. The Qwen model family keeps outperforming expectations at small sizes.

Gemma 2B if you need faster responses and can sacrifice some quality. Lowest RAM footprint, highest throughput.

Qwen 2.5 1.5B for the absolute minimum viable setup. Less capable, but responses come back fast enough to feel conversational.

Skip 7B models. They technically load on an 8GB Pi 5, but inference drops below 1 tok/s once you account for OS and gateway RAM usage. Life's too short.

Setup Option 1: Ollama (Easier)

Ollama is the fastest path to running a local model on your Pi. It handles model downloading, quantization selection, and serves an OpenAI-compatible API that OpenClaw can talk to.

Install Raspberry Pi OS

Flash **Raspberry Pi OS Lite (64-bit)** — Bookworm — using the Raspberry Pi Imager. No desktop needed. Pre-configure SSH, WiFi, and your username during flashing.

Boot, then update everything:

```
sudo apt update && sudo apt upgrade -y
```

Install Ollama

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

Ollama detects ARM64 automatically. Verify it's running:

```
ollama --version
systemctl status ollama
```

Pull a Model

```
ollama pull qwen2.5:3b
```

This downloads the Q4_K_M quantization by default — about 2 GB on disk. Test it:

```
ollama run qwen2.5:3b "What is a Raspberry Pi?"
```

Your first response will take a few seconds to start (model loading), then stream at 4-6 tok/s. Subsequent prompts in the same session are faster because the model stays loaded in RAM.

System Tuning

Reduce GPU memory allocation (the Pi's GPU is useless for LLM inference):

```
echo "gpu_mem=16" | sudo tee -a /boot/firmware/config.txt
```

Disable Bluetooth if you're not using it:

```
echo "dtoverlay=disable-bt" | sudo tee -a /boot/firmware/config.txt
```

Create swap space as a safety net (if you don't have one already):

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

Set swap aggressiveness low — you want the system to avoid swapping unless it has to:

```
echo "vm.swappiness=10" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

Reboot to apply config.txt changes.

Setup Option 2: llama.cpp (Faster, More Control)

Building llama.cpp from source on ARM gives you 10-20% better inference speed than Ollama, plus direct control over threading, batch size, and context length. Worth doing if you want to squeeze every tok/s out of the hardware.

Build Dependencies

```
sudo apt install -y build-essential cmake git
```

Clone and Build

```
cd ~
git clone https://github.com/ggml-org/llama.cpp.git
cd llama.cpp
cmake -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build --config Release -j4
```

The `-j4` uses all four cores for compilation. Build takes about 5-10 minutes on a Pi 5.

Download a Model

Grab a GGUF file directly from HuggingFace. Qwen 2.5 3B at Q4_K_M:

```
mkdir -p ~/models
cd ~/models
wget https://huggingface.co/Qwen/Qwen2.5-3B-Instruct-GGUF/resolve/main/qwen2.5-3b-instruct-q4_k
```

Run Inference

```
~/llama.cpp/build/bin/llama-cli \
  -m ~/models/qwen2.5-3b-instruct-q4_k_m.gguf \
  -t 4 \
  -n 256 \
  -c 2048 \
  --temp 0.7 \
  -p "You are a helpful assistant." \
  --interactive
```

Key flags:

- `-t 4` — use all 4 CPU threads
- `-c 2048` — context length (keep it low on Pi, higher = more RAM)
- `-n 256` — max tokens to generate per response

Run as an API Server

To connect OpenClaw, you need the API server:

```
~/llama.cpp/build/bin/llama-server \
  -m ~/models/qwen2.5-3b-instruct-q4_k_m.gguf \
  -t 4 \
  -c 2048 \
  --host 0.0.0.0 \
  --port 8080
```

This serves an OpenAI-compatible API at `http://localhost:8080`. Test it:

```
curl http://localhost:8080/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{"model": "qwen2.5-3b", "messages": [{"role": "user", "content": "Hello"}]}'
```

llama.cpp vs Ollama on Pi: Quick Comparison

	Ollama	llama.cpp
Setup time	2 minutes	15-20 minutes
Speed	Baseline	10-20% faster
Model management	Automatic	Manual GGUF downloads
API compatibility	OpenAI-compatible	OpenAI-compatible
RAM overhead	Slightly higher	Lower
Best for	Getting started fast	Squeezing max performance

If you just want it working, use Ollama. If every tok/s counts (and on a Pi, it does), build llama.cpp.

Installing OpenClaw

With your LLM backend running (Ollama, llama.cpp server, or a cloud API key), install OpenClaw itself.

Install Node.js 22

```
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt install -y nodejs
node --version  # Should show v22.x
```

Install OpenClaw

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

Then run the onboarding wizard:

```
openclaw onboard
```

The wizard walks you through:

1. **LLM backend** — choose "Local / Ollama" and point it at `http://localhost:11434` (Ollama) or `http://localhost:8080` (llama.cpp server)
2. **Messaging platform** — WhatsApp (QR scan), Telegram (BotFather token), or Discord
3. **Skills** — start with the defaults; you can add more later

For the full onboarding walkthrough, see our OpenClaw Setup Guide.

Configure for Local Models

Edit `~/.openclaw/config.yaml` and adjust these settings for Pi-level performance:

```
llm:
  provider: ollama          # or "openai-compatible" for llama.cpp
  base_url: http://localhost:11434
  model: qwen2.5:3b
  timeout: 120              # Local models are slow — increase timeout
  max_tokens: 256           # Keep responses short on limited hardware

agent:
  max_steps: 3              # Limit multi-step chains (each step = another inference)
  context_window: 2048      # Match your model's context setting
```

The `timeout: 120` is critical. OpenClaw's default timeout assumes cloud API speeds. A 256-token response at 4 tok/s takes over a minute. Without the extended timeout, the gateway kills the request before the model finishes.

Run as a Service

Create a systemd service so OpenClaw starts on boot:

```
sudo tee /etc/systemd/system/openclaw.service > /dev/null <<EOF
[Unit]
Description=OpenClaw AI Agent
After=network.target ollama.service
```

```
[Service]
Type=simple
User=$USER
WorkingDirectory=/home/$USER
ExecStart=/usr/bin/openclaw start
Restart=always
RestartSec=10
Environment=NODE_COMPILE_CACHE=/var/tmp/openclaw-compile-cache

[Install]
WantedBy=multi-user.target
EOF

sudo systemctl daemon-reload
sudo systemctl enable openclaw
sudo systemctl start openclaw
```

The `NODE_COMPILE_CACHE` environment variable speeds up CLI startup times noticeably on the Pi.

Where a Pi Setup Actually Makes Sense

Running OpenClaw on a Pi with local models is slow. That doesn't mean it's pointless. There are scenarios where this setup is exactly the right tool:

Air-gapped environments. No internet, no cloud APIs, no data leaving the building. Medical offices, legal firms, classified environments, or just the privacy-conscious. The Pi runs entirely offline once the model is downloaded. Pair it with a local network and you have a self-contained AI agent.

Always-on home server. The Pi draws 5-8 watts. Leave it running 24/7 as a personal assistant that responds to Telegram messages, runs cron-triggered tasks, or monitors RSS feeds. Point it at a cloud API for intelligence and use the Pi purely for orchestration. At $5/year in electricity, it's cheaper than any VPS.

Learning and experimentation. I learned more about LLM inference from a weekend with a Pi than from months of running models on a desktop GPU. When you only have 8GB of RAM, every decision about context length, quantization, and model size becomes visceral. You can't ignore the tradeoffs when they're staring at you in `htop`.

Low-power edge nodes. Part of a larger setup where a Pi handles simple queries locally (intent classification, quick lookups) and routes complex requests to a beefier machine. The distributed inference pattern applies here.

Offline travel assistant. Load a small model, pack the Pi, and you have a portable AI that works in airplane mode. Hotel WiFi in a foreign country isn't where you want to send sensitive queries through cloud APIs.

The Hybrid Approach: Cloud Brain, Pi Body

The most practical Pi setup isn't pure local. It's the Pi as gateway with a cloud API for the heavy lifting.

```
llm:
  provider: anthropic
  model: claude-sonnet-4-20250514
  api_key: sk-ant-...
```

This gives you:

- Fast, capable responses (cloud-speed intelligence)
- 24/7 uptime on $5/year of electricity
- All conversation data stored locally on the Pi
- No VRAM requirements, no model management

The Pi handles message routing, skill execution, and memory storage. Claude handles thinking. Each piece of hardware does what it's good at.

If the cloud API goes down or you want offline fallback, you can configure OpenClaw to fall back to a local model:

```
llm:
  provider: anthropic
  model: claude-sonnet-4-20250514
  fallback:
    provider: ollama
    model: qwen2.5:3b
```

You get cloud intelligence when connected and basic local capability when you're not.

Troubleshooting

Model loading fails with "out of memory": Your model is too large. Stick to 3B or smaller. Run `free -h` to check available RAM. If you see heavy swap usage during inference, downgrade to a smaller model.

Under-voltage warning (lightning bolt icon): Use the official 27W power supply. Third-party USB-C chargers frequently can't deliver enough power when the CPU is under sustained load with an NVMe drive attached.

Slow startup after reboot: Set `NODE_COMPILE_CACHE` (covered in the systemd section above). First boot after an OS update will be slower as caches rebuild.

OpenClaw times out waiting for response: Increase `timeout` in your config to 120-180 seconds. Local models on Pi hardware need time.

Temperature throttling during inference: Check with `vcgencmd measure_temp`. If you're hitting 80°C+, your cooling is insufficient. The official active cooling case keeps the CPU under 55°C during sustained inference.

Ollama consuming too much RAM at idle: Ollama keeps the last-used model loaded. Run `ollama stop` to unload it, or set `OLLAMA_KEEP_ALIVE=5m` to auto-unload after 5 minutes of inactivity.

What I'd Actually Build

If I were setting up an OpenClaw Pi today, here's exactly what I'd do:

1. Pi 5 (8GB) + NVMe SSD + active cooling case = ~$130
2. Install Ollama with Qwen 2.5 3B as a local fallback
3. Point OpenClaw at Claude Sonnet via API for primary intelligence
4. Connect Telegram for mobile messaging
5. Set up a systemd service for auto-start
6. Use it as an always-on personal agent that costs $5/year to run

The local model is there for offline situations and experimentation. The cloud API handles the real work. The Pi just keeps running.

For the complete setup walkthrough, start with our OpenClaw Setup Guide. For model selection beyond the Pi, see Best Local Models for OpenClaw. And before you connect any real accounts, read the OpenClaw Security Guide first.

Get notified when we publish new guides.

Subscribe — free, no spam
