# OpenClaw on Mac: Setup, Optimization, and What Actually Works

February 27, 2026 · by Mark Bartlett

Download this guide as PDF

**Quick Answer:** Install via Homebrew (brew install openclaw-cli — 3,800+ installs last month) or the one-line installer (curl -fsSL https://openclaw.ai/install.sh | bash). Connect it to Ollama at localhost:11434 by setting OLLAMA_API_KEY=ollama-local. The gateway runs as a launchd service (ai.openclaw.gateway) that starts on boot. On a 16GB Mac, OpenClaw plus a 7B model is comfortable. On 8GB, it's tight — close browsers first. The real payoff is the Mac Mini as an always-on server: OpenClaw + Ollama + Telegram polling, no inbound ports needed, chat from your phone. Run openclaw doctor after install to catch config problems before they waste your afternoon.

More on this topic: OpenClaw Setup Guide · OpenClaw Security Guide · How OpenClaw Works · Best Models for OpenClaw · Ollama on Mac: Setup & Optimization

OpenClaw's general setup guide tells you to run a curl command and follow a wizard. That works — on Linux. On Mac, you'll spend 20 minutes figuring out why environment variables don't stick, why the gateway won't start after a reboot, and where the logs actually go. Then you'll spend another 20 minutes wondering why your model runs at 3 tok/s until you realize Safari is eating 4GB of your unified memory.

This guide is Mac-specific. Everything here assumes macOS and Apple Silicon. If you're on Linux, the general setup guide is what you want.

If you haven't read our general OpenClaw setup guide or the security guide, do that first. This builds on both.

---

## Installing on Mac

You have two options. Homebrew is cleaner.

### Homebrew (recommended)

```
brew install openclaw-cli
openclaw onboard --install-daemon
```

The Homebrew formula (openclaw-cli, version 2026.2.26) had 3,857 installs in the last 30 days. It pulls Node 25 as a dependency and installs the native ARM binary on Apple Silicon. The `--install-daemon` flag creates a launchd service so the gateway starts on boot.

Don't use `npm install -g openclaw` on Mac unless you have a reason. Homebrew manages dependencies and updates more cleanly, and you avoid the `sharp`/ `libvips` compile errors that trip up global npm installs on macOS. If you do go the npm route and hit a build failure on `sharp`:

```
SHARP_IGNORE_GLOBAL_LIBVIPS=1 npm install -g openclaw@latest
```

### One-line installer

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

This detects your OS, installs Node if missing, sets up the CLI, and runs the onboarding wizard. Skip the wizard with `--no-onboard` if you want to configure manually.

### After install: run the doctor

```
openclaw doctor
```

This validates your gateway service, checks file permissions, and catches config problems before they bite you. On Mac, it also inspects the launchd supervisor config and warns if the service is installed but not actually running, or if something else is already using port 18789. If it finds issues:

```
openclaw doctor --fix
```

This fixes what it can automatically: broken permissions, missing directories, stale config keys. I run it after every major update.

### Version pinning

If you don't want Homebrew auto-updating OpenClaw when you run `brew upgrade`:

```
brew pin openclaw-cli
```

The built-in auto-updater (`update.auto`) is off by default. The stable channel uses a rollout delay with jitter so not everyone updates at once. If you want to opt in:

```
openclaw update --channel stable
```

To preview what an update would do without applying it:

```
openclaw update --dry-run
```

---

## Apple Silicon vs Intel

Check which binary you're running:

```
file $(which openclaw)
```

You want `arm64`. If you see `x86_64`, you're running the Intel binary through Rosetta, which works but is noticeably slower for anything touching local model inference. Reinstall via Homebrew to get the native ARM build.

On Apple Silicon (M1 through M4), OpenClaw runs natively and the gateway is light. The gateway itself uses maybe 200-500MB. The memory question is really about the model you connect to it, not OpenClaw itself.

On Intel Macs, OpenClaw works but local model inference through Ollama is noticeably slower since there's no Metal GPU acceleration for LLM inference on Intel. If you're on Intel and want to use OpenClaw with local models, consider pointing the gateway at a separate Apple Silicon machine running Ollama instead.

---

## Connecting to local models

OpenClaw is a gateway. It routes messages and manages tools, but it doesn't run a model itself. You need something serving a model for it to talk to. On Mac, that's usually Ollama, sometimes LM Studio, and increasingly vllm-mlx.

### OpenClaw + Ollama (most common)

Start Ollama if it isn't running already:

```
ollama serve
```

Tell OpenClaw where to find it by setting the environment variable:

```
export OLLAMA_API_KEY="ollama-local"
```

That value can be anything. It just activates the Ollama provider. OpenClaw auto-discovers your pulled models by querying `http://127.0.0.1:11434/api/tags` and filters for models with tool-calling support.

Reference models in your config as `ollama/model-name:tag`:

```
{
  "primary": "ollama/qwen2.5-coder:32b",
  "fallbacks": ["ollama/llama3.3", "ollama/qwen3:8b"]
}
```

Auto-discovery reads context windows from the model metadata. If you need to override, define the provider explicitly in your `openclaw.json` with a custom `baseUrl` and context settings.

Quick test to verify the connection:

```
curl http://localhost:11434/api/tags
```

If that returns your model list, OpenClaw can see Ollama.

### OpenClaw + LM Studio

LM Studio exposes an OpenAI-compatible API on port 1234. In your OpenClaw config, add it as an openai-completions provider:

```
{
  "providers": [{
    "name": "lmstudio",
    "baseUrl": "http://127.0.0.1:1234",
    "api": "openai-completions",
    "apiKey": "lm-studio"
  }]
}
```

The catch: LM Studio needs to be open as an app. It’s not a background service. For always-on setups, Ollama is the better choice.

### OpenClaw + vllm-mlx (fastest on Mac)

This one is worth knowing about. vllm-mlx is a native MLX inference server built specifically for Apple Silicon. The performance gap over Ollama is real:

| Metric | vllm-mlx | Ollama (llama.cpp) | Difference |
|---|---|---|---|
| Single-stream tok/s (M3 Ultra, 80B Q4) | 76.0 | 35.1 | 2.2x |
| 8 concurrent requests | 267.1 tok/s | 44.5 tok/s | 6x |

| Metric | vllm-mlx | Ollama (llama.cpp) | Difference |
|---|---|---|---|
| Cached TTFT (time to first token) | 0.115s | 0.25s | 2.2x |

The big win for OpenClaw specifically is prefix caching. Agentic workloads send repetitive system prompts and tool definitions on every turn. vllm-mlx caches these prefixes, cutting time-to-first-token from 30-90 seconds to 1-3 seconds on long contexts. Ollama recalculates them every time.

vllm-mlx exposes both OpenAI ( `/v1/chat/completions` ) and Anthropic ( `/v1/messages` ) endpoints. You can configure it as a provider in `openclaw.json` pointing at the local endpoint.

The discussion on the llm-mlx GitHub thread has setup details and benchmarks. It's still early. Ollama is the safer default. But if you're on an M-series Mac and running OpenClaw as a daily driver, vllm-mlx is worth the extra setup.

---

## Memory math: what fits on your Mac

OpenClaw's gateway uses roughly 200-500MB plus whatever the connected model needs. The model is the expensive part. All of this comes from the same unified memory pool. There's no separate GPU VRAM on Mac.

| Your Mac | RAM | What fits | Notes |
|---|---|---|---|
| M1/M2 base | 8GB | OpenClaw + 3B model | Tight. Close browsers. Close Slack. |
| M1/M2/M3 Pro | 16GB | OpenClaw + 7B-14B Q4 | Comfortable for most tasks |
| M4 Pro | 24GB | OpenClaw + 14B Q6 or 32B Q4 | Good coding agent setup |
| M3/M4 Max | 32-48GB | OpenClaw + 32B Q8 | Room for long contexts |
| M4 Max | 64GB | OpenClaw + 70B Q4 or 122B MoE | The Mac Mini sweet spot |
| M2-M4 Ultra | 128GB+ | OpenClaw + anything | Multiple models concurrently |

On 8GB: the gateway plus a 7B Q4 model (~5GB) plus macOS overhead (~2-3GB) adds up to more than 8GB. You'll swap to SSD. Drop to a 3B model, or accept slow generation. See our 8GB Apple Silicon guide for the full picture.

On 16GB: the sweet spot is a 7B-8B model. You can load a 14B Q4 if you close other apps. Check Activity Monitor's Memory Pressure graph — green means you're fine, yellow or red means the model doesn't fit. Our Ollama Mac troubleshooting guide covers this in detail.

On 64GB: this is where OpenClaw gets interesting. You can run a 30B coding model with vllm-mlx's prefix caching and the experience feels fast. A Mac Mini M4 Pro with 64GB running OpenClaw + Ollama + Telegram is, frankly, a better personal AI server than most cloud setups.

---

## The gateway: what it does and why it's confusing

If you're coming from a simple Ollama setup where you `ollama run model` and start chatting, OpenClaw's gateway is the part that trips you up.

The gateway is a WebSocket server (port 18789) that sits between you and the model. When you message OpenClaw on Telegram, the gateway receives the message, sends it to the model, runs whatever tools the model asks for, and returns the response. It also tracks sessions and manages which channel maps to which agent. Our architecture guide covers this in depth.

On Mac, the gateway runs as a launchd service. After install, check it:

```
openclaw status
```

If the gateway isn't running:

```
openclaw gateway start
```

To see the control UI in your browser:

```
openclaw dashboard
```

For local-only use (no external messaging, just you on this Mac), set the gateway mode to local. If you're connecting from your phone via Telegram, the gateway needs to reach the internet, but Telegram uses long-polling by default on Mac, so you don't need to open any inbound ports. Your bot polls Telegram's servers. This is the setup most Mac users want.

---

## launchd: making it stick across reboots

The `--install-daemon` flag during onboarding creates a launchd service labeled `ai.openclaw.gateway` (earlier versions used `bot.molt`). This starts the gateway at login and restarts it if the process crashes.

Check the service status:

```
launchctl list | grep openclaw
```

Restart it:

```
launchctl kickstart -k gui/$UID/ai.openclaw.gateway
```

Stop it entirely:

```
launchctl bootout gui/$UID/ai.openclaw.gateway
```

### Where logs go

Gateway logs write to `~/.openclaw/logs/` by default. On some installs, they end up in `~/Library/Logs/openclaw/`. Check both if you can't find them. The `openclaw doctor` command will tell you the correct path for your install.

For live log tailing:

```
tail -f ~/.openclaw/logs/gateway.log
```

### After updates

When you update OpenClaw (`brew upgrade openclaw-cli` or `openclaw update`), the launchd service config might be stale. Run:

```
openclaw doctor --fix
```

This checks the supervisor config against current defaults and rewrites it if needed. Without this, I've had the gateway fail to start after updates because the launchd plist pointed at a different binary path.

---

## macOS gotchas

These are the things that ate my time so they don't have to eat yours.

### Environment variables don't work the way you think

On Linux, you set env vars in a systemd service file or your shell profile. On Mac, neither works for the OpenClaw gateway because it runs as a launchd service, not a shell process.

Setting `OLLAMA_API_KEY` in your `.zshrc` does nothing for the gateway. It runs in its own context. Use `launchctl setenv` instead:

```
launchctl setenv OLLAMA_API_KEY "ollama-local"
```

Then restart the gateway. These reset on reboot. For persistence, add them to `~/.zprofile` (which runs at login, before launchd services fully initialize) or create a LaunchAgent plist.

The same applies to Ollama's env vars. If you need `OLLAMA_FLASH_ATTENTION=1`, set it via `launchctl setenv`, not your shell. Our Ollama Mac setup guide has the full explanation.

### Spotlight indexing eats CPU during model downloads

When Ollama downloads a multi-GB model to `~/.ollama/models/`, Spotlight tries to index it. On an 8GB Mac, Spotlight's indexing plus the download plus the existing model can push you into swap territory.

Exclude the models directory:

System Settings → Siri & Spotlight → Spotlight Privacy → add `~/.ollama/models/`

Do the same for `~/.openclaw/` if you store any large files there.

### Time Machine backs up model files

Time Machine doesn't know that a 4.5GB GGUF file is a downloaded artifact you can re-pull. It'll back it up, eating disk space.

Exclude `~/.ollama/models/` from Time Machine backups in System Settings → Time Machine → Options.

### Prevent sleep for always-on setups

If you're running OpenClaw on a Mac Mini as a server, disable sleep:

```
sudo pmset -a sleep 0 displaysleep 0 disksleep 0
```

Set auto-restart after power failure:

```
sudo pmset -a autorestart 1
```

Without these, your Mac will sleep after the display timeout and your OpenClaw agent goes dark. Amphetamine (free on the App Store) is a more granular alternative if you want sleep prevention only while OpenClaw is running.

### macOS firewall and messaging channels

If you're using webhook mode for Telegram or Discord (instead of long-polling), you need to allow incoming connections. Go to System Settings → Network → Firewall and either disable it or add an exception for the OpenClaw gateway.

For most home setups, you don't need this. Telegram long-polling works behind NAT with zero inbound ports. You only need firewall changes if you're running webhooks or exposing the gateway to other devices on your network.

### Credential storage

OpenClaw stores API keys in `~/.openclaw/openclaw.json` as plaintext. On Mac, you can use the macOS Keychain instead. The config supports a `$KEYCHAIN:secret-name` syntax. At startup, the config loader resolves these references via `security find-generic-password`.

To store a key in Keychain:

```
security add-generic-password -a openclaw -s "anthropic-api-key" -w "sk-ant-your-key-here"
```

Then in your config:

```
{
  "apiKey": "$KEYCHAIN:anthropic-api-key"
}
```

Better than a plaintext config file, especially if you use iCloud Keychain (though be aware that syncs the credential to all your Apple devices).

---

## Practical setups people run

### Mac Mini as always-on server

The most popular OpenClaw Mac setup by far. A Mac Mini M4 Pro (24GB or 64GB) running headless:

- OpenClaw gateway as a launchd daemon
- Ollama serving a coding model (Qwen 2.5 Coder 32B on 64GB, 14B on 24GB)
- Telegram long-polling — chat with your agent from your phone
- SSH enabled for remote management
- Sleep disabled, auto-restart on power loss
- No inbound ports open, no Cloudflare Tunnel needed

Total power draw: about 10-15W idle. This is a $20/year AI server that runs 24/7.

The openclaw-hardware guide covers the hardware decision in more detail.

### MacBook Pro as portable dev assistant

OpenClaw + Ollama on a laptop works fine when the laptop is open. The gateway runs, you chat via the terminal or a local web UI. When you close the lid, the gateway stops (sleep), and picks back up when you open it.

Set `OLLAMA_KEEP_ALIVE=-1` via `launchctl setenv` so the model stays loaded between queries instead of unloading after 5 minutes. On a 16GB MacBook, keep the model at 7B-8B to leave room for your IDE and browser.

### Mac Studio as multi-model workstation

With 64-192GB of unified memory, a Mac Studio can run OpenClaw with multiple models simultaneously. A small model handles quick triage and routing, while a 32B or 70B coding model takes the actual development work. OpenClaw's model routing handles when to use which. See our model routing guide for that configuration.

---

## Security on Mac

Two recent updates matter:

**v2026.2.14** patched an SSRF vulnerability (CVE-2026-26322) where the gateway accepted tool-supplied URLs without restrictions, letting an attacker make the gateway connect to arbitrary targets. Make sure you're on 2026.2.14 or later.

**v2026.2.23** hardened the browser sandbox and added credential redaction in logs so API keys don't leak into debug output. The sandbox media path fixes specifically affect macOS users. Earlier versions had issues with symlink chains in media paths that could escape the sandbox boundary.

Run the security audit:

```
openclaw security audit
openclaw security audit --deep    # includes live gateway probe
openclaw security audit --fix     # auto-remediate where possible
```

This flags the usual problems: missing gateway auth, open DM policies without allowlists, loose SSRF configuration, and wrong file permissions.

Mac-specific: keep `~/.openclaw/` permissions tight:

```
chmod 700 ~/.openclaw
chmod 600 ~/.openclaw/openclaw.json
```

The `openclaw doctor` command checks for world-readable config files and warns you. If you skipped doctor after install, run it now.

Read our full security guide before connecting any real accounts. Use throwaway credentials for testing.

---

## Troubleshooting quick reference

| Problem | Check | Fix |
|---|---|---|
| Gateway won't start | `openclaw status`, check port 18789 | `openclaw doctor --fix`, kill conflicting process |
| Ollama not detected | `curl http://localhost:11434/api/tags` | Start Ollama, set `OLLAMA_API_KEY` via launchctl |
| Slow generation | Activity Monitor → Memory Pressure | Close browsers, smaller model, lower quant |
| Crashes mid-conversation | Console.app → search "jetsam" | Model doesn't fit in RAM — downsize |
| Gateway dies after reboot | `launchctl list \| grep openclaw` | `openclaw onboard --install-daemon` |
| Env vars ignored | Check if set via launchctl, not .zshrc | `launchctl setenv VAR value`, restart gateway |
| Permission denied on config | `ls -la ~/.openclaw/openclaw.json` | `chmod 600 ~/.openclaw/openclaw.json` |

If your model is running slowly, the problem is almost always memory pressure. The diagnostic steps are the same as for Ollama alone — `ollama ps` to check GPU vs CPU, Activity Monitor for memory pressure, Console.app for jetsam kills. Our Ollama Mac troubleshooting guide walks through all of it.

---

## Related guides

- OpenClaw Setup Guide
- OpenClaw Security Guide
- How OpenClaw Works
- Best Models for OpenClaw
- OpenClaw Hardware: Mac Mini vs VPS vs PC
- OpenClaw Model Routing
- Ollama on Mac: Setup & Optimization
- Ollama Mac Troubleshooting
- Best Local LLMs for Mac in 2026
- Running LLMs on Mac M-Series
- 8GB Apple Silicon Local AI

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/openclaw-mac-setup-guide/

Free guides for running AI locally