Ollama on Mac Not Working? Fix Metal, Memory Pressure, and Slow Performance

February 26, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Run ollama ps. If the Processor column says CPU instead of GPU, Ollama isn't using Metal -- reinstall the native ARM build from ollama.com (not Homebrew's x86 version under Rosetta). If it says GPU but generation is slow, open Activity Monitor and check Memory Pressure. Yellow or red means your model doesn't fit -- close browsers, drop to a smaller model, or use Q4 quantization. If Ollama crashes mid-sentence with no error, check Console.app for 'jetsam' -- macOS killed it for using too much memory. Rule of thumb: your model's file size plus 2-3GB for macOS overhead is the minimum RAM you need.

More on this topic: Ollama on Mac: Setup & Optimization | Best Local LLMs for Mac | Running LLMs on Mac M-Series | Ollama Troubleshooting (all platforms) | 8GB Apple Silicon Local AI

Ollama on Mac mostly just works. Install it, pull a model, start chatting. But when it doesn't work, the failure modes are different from Windows and Linux because macOS handles GPU memory, process management, and environment variables differently. Generic Ollama troubleshooting guides skip these differences.

This guide covers every Mac-specific problem I've seen: Metal GPU not being used, slow generation from memory pressure, models crashing mid-sentence, and the Activity Monitor metrics that tell you what's actually going wrong.

Metal GPU not being used (running on CPU)

This is the problem that wastes the most time because nothing looks wrong. Ollama loads your model, you start chatting, everything seems to work – but it's running on CPU at a fraction of the speed it should.

How to check

```
ollama ps
```

Look at the Processor column:

```
NAME              ID           SIZE     PROCESSOR    UNTIL
llama3.2:3b       a80c4f17acd5 2.0 GB   100% GPU     4 minutes from now
```

If it says `100% GPU`, Metal is working. If it says `100% CPU` or a split like `48%/52% CPU/GPU`, something is wrong.

Cause 1: running the x86 binary under Rosetta

This is the one that wastes the most time because there's no obvious error message. On some M1 Macs, if you installed Ollama through an older method or an incorrect Homebrew formula, you might be running the Intel x86 binary through Rosetta translation instead of the native ARM build. Rosetta can't access Metal GPU acceleration, so Ollama falls back to CPU silently.

Check your binary architecture:

```
file $(which ollama)
```

You want to see `arm64`. If you see `x86_64`, that's the problem.

Fix: Uninstall and reinstall the native ARM build:

```
# Remove old install
brew uninstall ollama  # if installed via Homebrew

# Download native ARM .dmg from ollama.com
# Or reinstall via Homebrew (current formula is ARM-native):
brew install ollama
```

After reinstalling, verify with `ollama ps` that the Processor column shows GPU.

Cause 2: Ollama version regression

Certain Ollama versions have shipped with GPU-to-CPU fallback bugs on Apple Silicon. Version 0.12.9 introduced a regression where a fix for "CPU-only systems" accidentally triggered CPU fallback on unified memory Macs. Users reported going from 53 tok/s on GPU to 7 tok/s overnight after an auto-update.

Fix: If `ollama ps` suddenly shows CPU after an update, downgrade:

```
# Check your current version
ollama --version

# If on a broken version, download the last known-good .dmg from:
# https://github.com/ollama/ollama/releases
```

Version 0.12.5 is confirmed working for Metal. Check the release notes before upgrading past that.

Cause 3: debug it yourself

If the above don't apply, enable debug logging:

```
OLLAMA_DEBUG=1 ollama serve
```

Look for lines containing `library=cpu` or `library=metal` in the output. If you see `library=cpu`, Ollama isn't detecting your Metal GPU. File a bug with your `ollama --version`, macOS version, and chip model.

Slow token generation

The model loads, Metal is confirmed via `ollama ps`, but generation is painfully slow. On Mac, this almost always comes down to one of two things: memory bandwidth or memory pressure.

Memory bandwidth is your speed limit

Token generation speed on Apple Silicon is bottlenecked by memory bandwidth, not compute cores. The chip can do the math faster than it can read the model weights from memory. This is a hardware limit – no software setting changes it.

Chip	Memory bandwidth	Llama 3.1 8B Q4 (approx)
M1 base	68 GB/s	12-15 tok/s
M2 base	100 GB/s	18-22 tok/s

Chip	Memory bandwidth	Llama 3.1 8B Q4 (approx)
M3 base	100 GB/s	18-22 tok/s
M4 base	120 GB/s	22-28 tok/s
M1/M2/M3 Pro	200 GB/s	28-35 tok/s
M4 Pro	273 GB/s	35-42 tok/s
M1/M2/M3 Max	400 GB/s	45-55 tok/s
M4 Max	546 GB/s	55-65 tok/s
M2/M3 Ultra	800 GB/s	70-80+ tok/s

If your tok/s is roughly in line with these numbers, your Mac is performing normally. A base M1 at 12-15 tok/s isn't broken – it's bandwidth-limited. The only fix is a higher-bandwidth chip.

If your tok/s is significantly below these numbers, keep reading.

Memory pressure: why your Mac is secretly swapping to disk

This is the Mac-specific issue that trips up the most people. Your model loads, Metal is active, but generation runs at 2-5 tok/s instead of 25+. The model technically fits, but macOS is swapping to SSD behind your back.

How to check:

1. Open Activity Monitor (Applications → Utilities → Activity Monitor)
2. Click the Memory tab
3. Look at two things:
    - **Memory Pressure graph**: Green is fine. Yellow means the system is compressing memory. Red means active swapping.
    - **Swap Used**: Any number above 0 while running a model means your model doesn't actually fit.

When macOS runs out of physical RAM, it writes data to your SSD as "swap." Apple Silicon SSDs are fast for storage, but they're still 100x slower than unified memory for the random access patterns LLM inference needs. This is why generation drops from 25 tok/s to 2 tok/s – the model weights are being read from disk instead of memory.

The fix is always the same: make the model fit.

- Close Safari, Chrome, Slack, and VS Code. Browsers alone eat 2-4GB.

- Drop to a smaller model (3B instead of 7B, 7B instead of 14B)
- Use more aggressive quantization (Q4_K_M instead of Q6_K or Q8)
- Reduce context length: `ollama run llama3.2 --ctx-size 2048`

Memory pressure and swap death

How macOS unified memory actually works

On a PC with a discrete GPU, VRAM and system RAM are separate pools. On Mac, everything shares one pool: macOS, your apps, and the model all compete for the same memory. There's no separate "GPU memory" to fill.

When you load a model in Ollama, it claims a chunk of unified memory. macOS sees this the same way it sees Safari tabs or Xcode – just another memory consumer. If the total demand exceeds physical RAM, macOS starts compressing pages and swapping to SSD. It doesn't warn you, and it doesn't tell Ollama.

The rule of thumb

Model file size + 2-3GB for macOS overhead = minimum RAM needed.

A 4.5GB GGUF file (like Qwen 2.5 7B Q4_K_M) needs about 5.5GB of actual memory once you account for KV cache and framework overhead. On an 8GB Mac, that leaves 2.5GB for macOS – which is right at the edge. On a 16GB Mac, it's comfortable.

Your RAM	Model file size ceiling	What fits
8GB	~3-4GB	3B-7B Q4, expect pressure on 7B
16GB	~10-12GB	7B-14B Q4 comfortably
24GB	~18-20GB	14B Q6, 32B Q4 tight
32GB	~26-28GB	32B comfortably
64GB	~56-58GB	70B Q4

"But I have 6GB free according to Activity Monitor"

macOS reports memory in a confusing way. "Memory Used" includes cache that the OS can reclaim. "App Memory" is closer to what's actually occupied. But neither tells the full story.

The only metric that matters is the Memory Pressure graph. If it's green, you're fine. If it's yellow, you're on the edge. If it's red, you're swapping and your model is running at a fraction of its potential speed.

Don't look at the numbers. Look at the color.

Models crashing or getting killed mid-sentence

Symptom: generation stops, no error

You're chatting with a model. Mid-response, it just… stops. Run `ollama ps` and nothing is loaded. No error message, no crash dialog.

Cause: macOS jetsam

macOS has a process killer called jetsam. When memory pressure exceeds a threshold, jetsam terminates the highest-memory process to protect the system. Ollama, holding several gigabytes of model weights, is usually the biggest target.

The error Ollama shows (if it shows anything) is:

```
Error: llama runner process has terminated: signal: killed
```

That "signal: killed" is jetsam. macOS decided your model was threatening system stability and killed it.

How to confirm

Open Console.app (Applications → Utilities → Console). Search for `jetsam`. If you see entries timestamped around when your model died, that's your answer.

You can also check from terminal:

```
log show --predicate 'eventMessage contains "jetsam"' --last 1h
```

Fix

Jetsam kills happen because the model doesn't fit. The solutions are the same as the memory pressure section above: smaller model, lower quantization, fewer background apps. There's no way to tell macOS "don't kill Ollama" – jetsam is a kernel-level mechanism with no user override.

If you're repeatedly hitting jetsam on a model that should fit based on file size alone, check whether context length is expanding the KV cache beyond what you expect. A 7B model at Q4 might use 5GB at 2K context but 7GB at 8K context. Set `--ctx-size` explicitly.

Mac-specific optimizations

Environment variables on macOS

I've seen this one in at least a dozen GitHub issues. On Linux, you set Ollama env vars in a systemd service file or your shell profile. On Mac, neither works because Ollama runs as a macOS application, not a shell process.

The correct way on Mac:

```
launchctl setenv OLLAMA_FLASH_ATTENTION 1
launchctl setenv OLLAMA_KEEP_ALIVE -1
launchctl setenv OLLAMA_NUM_PARALLEL 1
```

Then restart Ollama (quit from the menubar and reopen, or `brew services restart ollama`).

These settings reset on reboot. For persistence, create a LaunchAgent plist in `~/Library/LaunchAgents/`, or add the `launchctl setenv` commands to your `.zprofile` (which runs at login, before Ollama starts).

Setting env vars in `.zshrc` or exporting them in terminal does nothing for the Ollama background service – it runs in its own context, not your shell.

Recommended settings by RAM tier

RAM	OLLAMA_NUM_PARALLEL	OLLAMA_KEEP_ALIVE	Notes
8GB	1	5m (default)	Don't waste memory on parallelism. Let models unload.

RAM	OLLAMA_NUM_PARALLEL	OLLAMA_KEEP_ALIVE	Notes
16GB	1-2	-1 (keep loaded)	Keep model warm to avoid reload time
24GB+	2-4	-1	Can handle parallel requests
32GB+	4	-1	Comfortable with multi-model serving

On 8GB, `OLLAMA_NUM_PARALLEL=1` matters more than any other setting. Each parallel slot reserves additional KV cache memory. On a machine where every megabyte counts, cutting from the default to 1 frees 500MB-1GB.

Close browsers before running large models

I keep saying this because it keeps being the fix. Safari with 10 tabs: 1-2GB. Chrome with 10 tabs: 2-4GB. That's the difference between a model running in memory and a model swapping to disk on 8-16GB machines.

Before launching a large model, quit (not minimize) your browser. Check Activity Monitor to confirm memory pressure is green. Then load the model.

Quick reference: what fits on your Mac

RAM	Best model	Max model	Expect
8GB	Llama 3.2 3B Q4	7B Q4 (tight, will swap)	12-35 tok/s depending on chip
16GB	Qwen 3 8B Q4	14B Q4 (some pressure)	18-42 tok/s
24GB	Qwen 3 14B Q6	32B Q4 (tight)	22-55 tok/s
32GB	Qwen 3 32B Q4	32B Q6	25-65 tok/s
48GB	Qwen 3 32B Q8	70B Q3 (degraded)	28-70 tok/s
64GB+	Llama 3.1 70B Q4	70B Q6	30-80+ tok/s

These are comfortable fits – models that run without memory pressure, leaving room for macOS and light background tasks. "Max model" means it technically loads but you'll be close to the edge.

If your numbers are significantly below the tok/s ranges above, work through this guide from the top: check `ollama ps` for CPU vs GPU, check Activity Monitor for memory pressure, check Console.app for jetsam kills. One of those three will explain the problem.

Related guides

- Ollama on Mac: Setup & Optimization
- Best Local LLMs for Mac in 2026
- Running LLMs on Mac M-Series
- Ollama Troubleshooting (all platforms)
- 8GB Apple Silicon Local AI

Get notified when we publish new guides.

Subscribe — free, no spam