Ollama on Mac: Setup and Optimization Guide (2026)

February 26, 2026 · by Mark Bartlett

Download this guide as PDF

Quick Answer: Install with brew install ollama or the .dmg from ollama.com. Pull your first model based on RAM: 8GB gets Llama 3.2 3B, 16GB gets Qwen 3 8B, 24-32GB gets Qwen 3 14B, 48GB+ gets Qwen 3 32B. Run ollama ps to confirm the Processor column says GPU, not CPU. Set env vars with launchctl setenv on Mac, not .zshrc. Enable flash attention (OLLAMA_FLASH_ATTENTION=1) and set OLLAMA_KEEP_ALIVE=-1 to keep models in memory. Memory bandwidth determines your speed more than chip generation -- an M3 Max is faster than an M4 Pro for token generation.

📚 **More on this topic:** Best Local LLMs for Mac 2026 · Running LLMs on Mac M-Series · Ollama Troubleshooting Guide · llama.cpp vs Ollama vs vLLM

Ollama is the fastest path from "I want to try local AI" to a model running on your Mac. One install, one command, and you're talking to a model. No Python, no Docker, no CUDA drivers.

The generic Ollama docs work fine for getting started. What they skip is the Mac-specific stuff: how unified memory changes the rules, why your environment variables aren't taking effect, which models fit your RAM, and how to confirm the GPU is actually being used.

Install Ollama and run your first model

Homebrew vs direct download

Both work. The differences are minor:

Method	Command	Auto-updates	Menubar app	Service management
Homebrew	`brew install ollama`	`brew upgrade`	No	`brew services start ollama`
Direct download	ollama.com/download	Yes (automatic)	Yes	Built-in

The direct download gives you a menubar app that auto-updates and auto-starts at login. Homebrew gives you CLI-only control and explicit version management.

Pick direct download if you want it running in the background without thinking about it. **Pick Homebrew** if you want to control exactly when it updates or if you manage everything through Homebrew already.

Either way, Metal GPU acceleration is compiled into the binary. No extra setup.

Pull your first model

After installing, open Terminal and pull a model matched to your RAM:

Your RAM	Model	Pull command	Size on disk
8 GB	Llama 3.2 3B	`ollama pull llama3.2`	2.0 GB
16 GB	Qwen 3 8B	`ollama pull qwen3:8b`	4.9 GB
24 GB	Qwen 3 14B	`ollama pull qwen3:14b`	9.0 GB
32 GB	Qwen 3 32B	`ollama pull qwen3:32b`	19 GB
48 GB+	Qwen 3 32B (Q6_K)	`ollama pull qwen3:32b-q6_K`	24 GB
64 GB+	Llama 3.3 70B	`ollama pull llama3.3:70b`	40 GB

Then run it:

```
ollama run qwen3:8b
```

You should see output within a few seconds. If it takes 30+ seconds before the first token, something's wrong (probably CPU fallback; see troubleshooting below).

Verify Metal GPU is working

Most people skip this. Don't. Run:

```
ollama ps
```

Check the **Processor** column:

- `100% GPU` means Metal is active. You're good.
- `100% CPU` means it's using system RAM only. Something is wrong.
- `48%/52% CPU/GPU` means the model is partially offloaded. Expected if the model barely fits.

If you see CPU when you expected GPU, jump to the troubleshooting section below.

For more detail, enable debug logging:

```
OLLAMA_DEBUG=1 ollama serve
```

Then pull up the logs. You'll see messages about Metal device detection, memory allocation, and layer offloading.

How Ollama uses your Mac's hardware

Unified memory changes the rules

On a PC, your GPU has dedicated VRAM (usually 8-24GB). Models that don't fit in VRAM either won't load or run at 2-3 tok/s on CPU fallback.

On Apple Silicon, there's no separate GPU memory. Your entire RAM pool is shared between CPU and GPU. This means a 48GB Mac Mini can load a 32B model that would require a $700+ RTX 3090 on PC.

The downside: that same 48GB also runs macOS, your browser, and everything else. You don't get the full amount for models.

The 66%/75% GPU memory rule

macOS caps how much unified memory the GPU can use:

- **36GB RAM or less:** GPU gets about 66% (~21GB on a 32GB machine)
- **More than 36GB:** GPU gets about 75% (~48GB on a 64GB machine)

These are system-enforced limits. Anything beyond that spills to CPU processing, which is slower but still works. Ollama handles the split automatically.

What this means in practice:

Your RAM	GPU allocation	Largest model (Q4)
8 GB	~5 GB	3B comfortably, 7B tight
16 GB	~10 GB	8B comfortably
24 GB	~16 GB	14B comfortably
32 GB	~21 GB	14B-20B range
48 GB	~36 GB	32B comfortably
64 GB	~48 GB	32B-70B range
96 GB	~72 GB	70B comfortably
128 GB	~96 GB	70B at high quant, or 2-3 models

These numbers assume some headroom for macOS and apps. Close Chrome before loading large models. (Seriously. Browsers eat 4-8GB on their own.)

Memory bandwidth determines speed

An M3 Max generates tokens faster than an M4 Pro, even though the M4 Pro is a newer chip. Token generation speed is bottlenecked by memory bandwidth, not compute. This surprises people.

Chip	Memory bandwidth	Rough 8B Q4 speed
M1	68 GB/s	~15 tok/s
M1 Pro	200 GB/s	~25 tok/s
M2	100 GB/s	~18 tok/s
M2 Pro	200 GB/s	~28 tok/s
M3	100 GB/s	~20 tok/s
M3 Pro	150 GB/s	~25 tok/s
M3 Max	400 GB/s	~45 tok/s
M4	120 GB/s	~22 tok/s
M4 Pro	273 GB/s	~35 tok/s
M4 Max	546 GB/s	~58 tok/s

These are approximations for 8B Q4 models. Larger models will be slower. The pattern holds: bandwidth predicts speed better than chip generation.

What Activity Monitor should show

Open Activity Monitor while running a model and check:

1. **Memory tab:** Memory Pressure should be green or yellow. Red means you're swapping, and performance will tank.
2. **GPU tab (if visible):** You should see Ollama or `ollama_llama_server` using GPU. If it only appears in the CPU tab, Metal isn't active.
3. **Memory Used:** Compare against your model size. A 4.9GB model should increase memory usage by roughly that amount.

If Memory Pressure hits red, you need a smaller model or lower quantization. There's no way to force more memory; macOS manages the split.

Mac-specific optimization

Environment variables that matter

Ollama's behavior is controlled through environment variables. The ones that matter most on Mac:

Variable	Default	Recommended for Mac	What it does
`OLLAMA_KEEP_ALIVE`	5m	`-1` (infinite) or `24h`	How long a model stays loaded after last request
`OLLAMA_FLASH_ATTENTION`	0	`1`	Reduces memory usage, slight speed boost
`OLLAMA_KV_CACHE_TYPE`	`f16`	`q8_0`	Halves KV cache memory with minimal quality loss
`OLLAMA_NUM_PARALLEL`	auto (1-4)	Leave default	Max parallel requests per model
`OLLAMA_MAX_LOADED_MODELS`	3	1-2 on 8-16GB	Max models in memory simultaneously

Variable	Default	Recommended for Mac	What it does
`OLLAMA_HOST`	127.0.0.1	`0.0.0.0` if serving LAN	Network bind address
`OLLAMA_CONTEXT_LENGTH`	4096	Depends on task	Token window size

Setting environment variables on Mac (not .zshrc)

This trips up almost everyone. If you installed Ollama via the .dmg (direct download), it runs as a macOS app, not from your shell. Setting variables in `.zshrc` or `.bash_profile` does nothing because the app doesn't read shell config files.

For the macOS app (direct download):

```
launchctl setenv OLLAMA_KEEP_ALIVE "-1"
launchctl setenv OLLAMA_FLASH_ATTENTION "1"
launchctl setenv OLLAMA_KV_CACHE_TYPE "q8_0"
```

Then restart the Ollama app (click the menubar icon, select "Quit Ollama", reopen it).

For Homebrew (`brew services`):

Create or edit `~/.ollama/env` (or set them before starting the service):

```
# In your shell config (.zshrc):
export OLLAMA_KEEP_ALIVE=-1
export OLLAMA_FLASH_ATTENTION=1
export OLLAMA_KV_CACHE_TYPE=q8_0
```

Then restart:

```
brew services restart ollama
```

To verify a variable is set:

```
# Check what the running Ollama process sees
launchctl getenv OLLAMA_KEEP_ALIVE
```

Keep models in memory

By default, Ollama unloads models after 5 minutes of inactivity. Every time you come back, there's a 2-10 second reload delay (depending on model size and disk speed).

Set `OLLAMA_KEEP_ALIVE=-1` to keep models loaded indefinitely. On a Mac you use primarily for AI work, this is worth the memory tradeoff. The model sits in RAM ready to respond instantly.

To unload manually when you need the memory back:

```
ollama stop qwen3:8b
```

Flash attention and KV cache quantization

These two settings reduce memory usage with minimal quality impact:

`OLLAMA_FLASH_ATTENTION=1` optimizes the attention computation to use less memory. No quality downside. Enable it unconditionally.

`OLLAMA_KV_CACHE_TYPE=q8_0` quantizes the KV cache, which stores context during generation. At default FP16, a 32K context window can eat 2-4GB of memory. Setting it to `q8_0` halves that with negligible quality impact. Setting it to `q4_0` cuts it to a quarter but has a noticeable quality reduction on long conversations.

Combined, these two settings can free 2-6GB of memory depending on your context length. That can be the difference between a model fitting in GPU allocation or spilling to CPU.

One caveat: there have been reports of slight performance regressions with KV cache quantization on Apple's Metal backend specifically. Test with your model. If you notice slower generation after enabling `q8_0`, revert to `f16`.

Best models by Mac configuration

For detailed model recommendations with tok/s benchmarks, see our Best Local LLMs for Mac 2026 guide. Here's the quick reference:

Mac config	RAM	Best general model	Best coding model	Best reasoning model
MacBook Air M1/M2	8 GB	Llama 3.2 3B	Qwen 2.5 Coder 3B	Phi-4 Mini 3.8B
MacBook Pro M2/M3 Pro	16 GB	Qwen 3 8B	Qwen 2.5 Coder 7B	DeepSeek-R1-Distill-8B
MacBook Pro M3/M4 Pro	24 GB	Qwen 3 14B	Qwen 2.5 Coder 14B	DeepSeek-R1-Distill-14B
Mac Mini/Studio M4 Pro	32 GB	Qwen 3 14B (Q6)	Qwen 2.5 Coder 14B (Q6)	DeepSeek-R1-Distill-14B (Q6)
Mac Mini/Studio M4 Pro	48 GB	Qwen 3 32B	Qwen 2.5 Coder 32B	QwQ 32B
Mac Studio M4 Max	64 GB	Qwen 3 32B (Q8)	Qwen 2.5 Coder 32B (Q8)	QwQ 32B (Q8)
Mac Studio M3/M4 Ultra	96 GB	Llama 3.3 70B	Qwen 2.5 Coder 32B + second model	DeepSeek-R1-Distill-70B
Mac Studio Ultra	128 GB	Llama 3.3 70B (Q6)	Multi-model setup	DeepSeek-R1-Distill-70B (Q6)

The 48GB tier is the price/performance sweet spot on Mac. A Mac Mini M4 Pro with 48GB ($1,199) runs 32B models that need a $700+ GPU on PC, and does it with zero fan noise.

At 96-128GB, you're running 70B models that require $3,000+ in multi-GPU setups on PC. No other consumer platform gives you that much memory for AI at this price.

Troubleshooting Mac-specific issues

For the full troubleshooting guide covering all platforms, see our Ollama Troubleshooting Guide.

“ollama ps” shows CPU instead of GPU

The most common Mac problem. Symptoms: slow generation (3-5 tok/s when you expected 20+), high CPU usage in Activity Monitor, no GPU activity.

First, is the model too large for GPU allocation?

```
ollama ps
```

If it shows a CPU/GPU split like `23%/77%`, that’s normal. The model partially fits in GPU memory, the rest runs on CPU. If it shows `100% CPU` for a model that should fit, keep going.

Second, are you running through Docker? Docker on Mac has no access to Metal GPU acceleration. This is an Apple/Docker limitation, not an Ollama bug. If Ollama is inside Docker, it will always use CPU. Run it natively instead. The performance difference is 5-6x.

Third, did a recent Ollama update break things? Version 0.12.9 had a known regression where CPU fallback logic accidentally triggered on Apple Silicon. Some users went from 53 tok/s to 7 tok/s overnight. Update to the latest version. If that doesn’t fix it, remove any custom `num_gpu` parameters from Modelfiles, as restrictive GPU layer settings can force CPU fallback.

Memory pressure warnings

If Activity Monitor shows red memory pressure while running a model:

1. Close browsers and heavy apps. Chrome alone can use 4-8GB.
2. Use a smaller quantization. Switch from Q6 to Q4: `ollama pull qwen3:14b-q4_K_M` instead of the default.
3. Reduce context length. Set `OLLAMA_CONTEXT_LENGTH=2048` if you don’t need long conversations.
4. Use a smaller model. Going from 14B to 8B frees several GB immediately.

Yellow memory pressure is usually fine. The model might swap slightly, but generation speed stays reasonable. Red means active swapping and the model will feel unusable.

Performance drops after updates

Both macOS updates and Ollama updates can affect performance.

After a macOS update, Metal drivers sometimes change behavior. Restart your Mac (not just sleep/wake) after major updates. Check `ollama ps` to confirm GPU is still active.

After an Ollama update, check the Ollama releases page for known issues. If performance dropped, you can pin a specific version via Homebrew:

```
brew install ollama@0.15.4  # whatever version worked
```

Or download a specific .dmg from the GitHub releases page.

Where Mac excels and where it falls short

Mac loads what NVIDIA can't

A 32B model needs roughly 18-20GB of memory at Q4. On PC, that requires an RTX 3090 ($700+ used) or RTX 4090 ($1,800+). On Mac, a $1,199 Mac Mini with 48GB handles it with room to spare.

A 70B model needs roughly 40GB at Q4. On PC, you need two GPUs and a system that supports multi-GPU inference. On Mac, a Mac Studio with 96GB ($3,000-4,000) runs it natively.

No other consumer platform lets you load models this large at these price points.

The tradeoff: token speed per dollar

For models that fit in 24GB, an RTX 3090 is faster. A 7B model runs at 80-100+ tok/s on an RTX 3090 vs 15-45 tok/s on Apple Silicon (depending on your chip). CUDA's memory bandwidth advantage is real.

If your workload stays within 24GB of VRAM, a used RTX 3090 ($450-550) plus a cheap PC build delivers more tok/s per dollar than any Mac.

What only Mac can do at consumer prices

Capability	Mac cost	PC equivalent cost
Run 32B models	$1,199 (Mac Mini 48GB)	$700-1,800 (3090/4090)
Run 70B models	$3,000-4,000 (Mac Studio 96GB)	$3,000+ (dual GPU + board)

Capability	Mac cost	PC equivalent cost
Run 70B at high quant	$4,000-5,000 (Mac Studio 128GB)	Not practical on consumer PC
Silent operation	Built-in	Aftermarket cooling needed
Multi-model serving	96GB+ covers 2-3 models	Each model needs its own GPU

The 96-128GB tier is where Mac has no PC equivalent at a similar price. Running two 32B models simultaneously, or one 70B model at Q6 quality, simply requires more memory than any single consumer GPU offers.

For a deeper comparison of Mac vs PC tradeoffs, see our Mac M-Series guide.

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/ollama-mac-setup-optimization/

Free guides for running AI locally