Best Way to Run Qwen 3.5 on Mac: MLX vs Ollama Speed Test

February 26, 2026 · Updated: April 29, 2026

Download this guide as PDF

> **Quick Answer:** On Apple Silicon, running Qwen 3.5 through MLX (via LM Studio or mlx_lm) is roughly 2x faster for token generation and up to 5x faster for prompt processing compared to Ollama's llama.cpp backend. The 35B-A3B at Q4 hits 60-70+ tok/s on an M4 Max via MLX vs ~35 tok/s through Ollama. MLX also uses about half the memory for the same model. Use LM Studio with the MLX backend for the fastest Mac experience. Use Ollama if you need an always-on API for apps like Open WebUI or Continue.dev. On 16GB, the 35B-A3B at Q4 fits comfortably with MLX's lower memory footprint. On 64GB+, the 122B-A10B is the model that justifies your hardware.

More on this topic: Qwen 3.5 Local AI Guide | LM Studio vs Ollama on Mac | Best Local LLMs for Mac | Ollama on Mac: Setup & Optimization | Running LLMs on Mac M-Series

Qwen 3.5 dropped on February 24, 2026, and Mac users finally have a model family built around the thing Apple Silicon is best at: feeding large models from unified memory without a discrete GPU. The 35B-A3B only activates 3 billion parameters per token despite having 35 billion total, which means it runs at small-model speeds with large-model quality. On Mac, that speed depends entirely on which backend you choose.

MLX, Apple's native ML framework, runs the 35B-A3B at roughly twice the speed of Ollama on the same hardware. I've been running both side by side on the same Mac and the gap is consistent: 60-70+ tok/s via MLX vs ~35 tok/s via Ollama on an M4 Max. The prompt processing gap is even wider – 5x in some tests. This isn't a small optimization. It changes whether the model feels fast or feels like you're waiting.

Both routes work. Both are free. This guide covers when each one makes sense, how to set them up, and which Qwen 3.5 model to run on your Mac.

Qwen 3.6 update — MLX vs Ollama on April 2026 (April 29)

Qwen 3.6-35B-A3B is now active and r/LocalLLaMA's daily-driver pick. Same MoE shape as 3.5-35B-A3B (3B active per token, 256K context, Apache 2.0), better numbers on SWE-Bench,

Terminal-Bench, and tool-use — see the Qwen 3.5 Local AI Guide for the head-to-head. Two wrinkles for Mac users.

baa-ai 3-bit MLX quants for RAM-poor Macs. baa-ai's 3-bit MLX builds bring the 35B-A3B down to roughly 12-14 GB resident — 16 GB M1/M2 MacBook Airs that previously had to swap (or skip it entirely) can now load it cleanly. Quality cost vs Q4 is real but not catastrophic on chat and short-context coding. If you've got 24 GB+ unified memory, stay on Q4_K_M MLX. The 3-bit path is for Macs that otherwise can't play.

DFlash is Linux/CUDA only — Mac users don't get the 2× speedup. The DFlash speculative-decoding draft for Qwen 3.5-27B (and the 3.6 draft once it lands) ships as a CUDA-only path through llama.cpp. No Metal kernel for it yet, so the 2-3× decode speedup we wrote up for the RTX 3090 does not apply on Apple Silicon. The MLX-vs-Ollama gap measured below for 3.5 carries over to 3.6 — MLX is still the lever.

The speed gap: MLX vs Ollama on Apple Silicon

Why the difference exists

Ollama uses llama.cpp with Metal Performance Shaders (MPS) for GPU acceleration on Mac. MPS works – it's a solid, well-tested backend – but it's a generic GPU abstraction layer. It wasn't built for Apple's unified memory architecture specifically.

MLX was built by Apple, for Apple Silicon. It targets unified memory directly, avoids the copy overhead between CPU and GPU address spaces (they're the same memory on Apple Silicon, but llama.cpp's abstractions don't fully exploit this), and uses optimized Metal compute shaders tuned for Apple's GPU cores.

The result: MLX consistently generates tokens 25-50% faster than llama.cpp on smaller models, and the gap widens to roughly 2x on MoE models like the 35B-A3B. MoE routing is particularly expensive through generic backends, and MLX handles it more efficiently.

Benchmark comparison

These numbers combine community benchmarks and data from our LM Studio vs Ollama on Mac testing with similar-architecture models, extrapolated for Qwen 3.5's MoE workload:

Chip	Qwen3.5-35B-A3B Q4 via Ollama	Qwen3.5-35B-A3B Q4 via MLX	MLX advantage
M1 Pro (16GB)	~15-18 tok/s	~25-35 tok/s	~1.5-2x

Chip	Qwen3.5-35B-A3B Q4 via Ollama	Qwen3.5-35B-A3B Q4 via MLX	MLX advantage
M3 Pro (18GB)	~18-22 tok/s	~35-45 tok/s	~1.8-2x
M4 Pro (24GB)	~25-30 tok/s	~45-55 tok/s	~1.8x
M4 Max (64GB)	~30-38 tok/s	~60-75 tok/s	~2x
M3 Ultra (192GB)	~35-45 tok/s	~70-90 tok/s	~2x

Prompt processing is where the gap gets extreme. MLX processes input tokens 3-5x faster than Ollama's llama.cpp backend. If you paste a 10-page document and ask a question, MLX starts generating in 2-3 seconds where Ollama might take 10-15 seconds to chew through the prompt.

Memory usage is the other half of the story. MLX uses roughly 50% less memory than Ollama for the same model at the same quantization. On a 16GB Mac, that's the difference between the 35B-A3B running cleanly or swapping to disk.

Setup: the MLX route (fastest)

Option 1: LM Studio (recommended)

LM Studio is the easiest way to run MLX on Mac. It detects your hardware, offers MLX-format models alongside GGUF, and handles everything through a GUI.

1. Download LM Studio from lmstudio.ai
2. Search for "Qwen3.5-35B-A3B"
3. Pick the MLX variant (look for "MLX" in the format tag, not "GGUF")
4. Choose Q4 for 16-24GB Macs, Q8 for 32GB+
5. Load and chat

LM Studio also exposes an OpenAI-compatible API on port 1234 and an Anthropic-compatible API, so you can point coding tools at it when you need MLX speed for development work.

The downside: LM Studio needs to be running as an app. It's not an always-on background service like Ollama.

Option 2: mlx_lm CLI (for scripting)

If you want terminal-level control or need to integrate Qwen 3.5 into a Python pipeline:

```
pip install mlx-lm

# Run the 35B-A3B directly from HuggingFace
mlx_lm.generate \
  --model mlx-community/Qwen3.5-35B-A3B-4bit \
  --prompt "Explain the difference between MoE and dense models" \
  --max-tokens 500
```

The mlx-community org on HuggingFace has pre-quantized MLX weights for every Qwen 3.5 model. Available variants include 4-bit, 8-bit, and nvfp4 (a format that preserves MoE router gates at 8-bit precision for better routing accuracy).

For a server setup:

```
mlx_lm.server --model mlx-community/Qwen3.5-35B-A3B-4bit --port 8080
```

This gives you an OpenAI-compatible endpoint. Not as polished as Ollama's server, but it runs on MLX.

Which MLX quant to download

Your RAM	Recommended MLX model	File size	Notes
16GB	Qwen3.5-35B-A3B-4bit	~22 GB	Tight. Close browsers first.
24GB	Qwen3.5-35B-A3B-4bit	~22 GB	Comfortable, room for context
32GB	Qwen3.5-35B-A3B-8bit	~38 GB	Higher quality, fits well
64GB	Qwen3.5-122B-A10B-4bit	~70 GB	The Mac power user pick
128GB+	Qwen3.5-397B-A17B-4bit	~214 GB	If you have an Ultra

Setup: the Ollama route (simpler ecosystem)

Ollama is slower on Mac, but it's still the right choice for certain workflows.

```
ollama pull qwen3.5:35b-a3b
ollama run qwen3.5:35b-a3b
```

That's it. The model downloads, Metal acceleration kicks in, and you're chatting. No format decisions, no HuggingFace browsing, no Python.

When Ollama wins despite the speed gap

- **Always-on API**: Ollama runs as a background service via launchd, starting at boot. Your apps hit its API 24/7 without launching anything. LM Studio needs to be open.
- **Ecosystem**: Open WebUI, Continue.dev, Aider, and dozens of other tools speak Ollama natively. Some support LM Studio's API too, but Ollama compatibility is more universal.
- **Multi-model switching**: Ollama manages multiple models and swaps between them automatically. Pull both `qwen3.5:35b-a3b` and `qwen3.5:27b` and route between them.
- **Simplicity**: One command to pull, one command to run. No thinking about MLX vs GGUF formats.

For chatting and light coding help, Ollama's speed on Mac is fine. 18-35 tok/s reads comfortably. The speed gap matters when you're processing long documents, running batch jobs, or sitting through repeated model calls in a pipeline.

Mac-specific Ollama tips

Set environment variables with `launchctl`, not your shell profile:

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

Then restart Ollama. See our Ollama Mac setup guide for the full config.

Which Qwen 3.5 model on which Mac

The Qwen 3.5 family has four models. Three are MoE (only a fraction of parameters activate per token), one is dense (all parameters activate).

Model	Total	Active	Architecture	Best for
35B-A3B	35B	3B	MoE	Speed, tool use, general tasks
27B	27B	27B	Dense	Coding, deep reasoning
122B-A10B	122B	10B	MoE	Everything, if you have the RAM
397B-A17B	397B	17B	MoE	Frontier-class, 128GB+ only

Model picks by memory tier

Your Mac	RAM	Model	Quant	What to expect
M1/M2 base	8GB	35B-A3B	Q3	Tight. Expect memory pressure. Close everything else.
M1/M2/M3 Pro	16GB	35B-A3B	Q4	Comfortable with MLX. Swap risk with Ollama.
M4 Pro	24GB	35B-A3B Q8 or 27B Q4	Q4-Q8	Both models fit. Pick by use case.
M3/M4 Max	32-48GB	27B Q6-Q8	Q6-Q8	Coding sweet spot. Room for long context.
M4 Max	64GB	122B-A10B	Q4	The model that justifies the hardware.
M2/M3/M4 Ultra	128GB+	122B Q8 or 397B Q4	Q4-Q8	Frontier-class, entirely local.

On 8GB, the 35B-A3B barely fits because it only activates 3B parameters per token. But the full 35B of weights still need to live in memory. With MLX's lower memory footprint, it's possible at Q3. With Ollama, you'll likely swap. See our 8GB Apple Silicon guide for more on working within 8GB limits.

On 16GB, the 35B-A3B at Q4 (~22GB GGUF, but MLX uses roughly half that effective memory) is the pick. This is the tier where MLX vs Ollama matters most – MLX's memory efficiency can mean the difference between running cleanly and swapping to disk.

On 64GB, skip the 35B-A3B and run the 122B-A10B at Q4. It beats GPT-5 mini by 30% on tool use (BFCL-V4: 72.2 vs 55.5) and matches it on SWE-bench. With only 10B active parameters, generation speed is good despite the massive total parameter count.

Memory bandwidth is your speed limit

The tok/s numbers above assume the model fits in memory. If it does, your speed is determined by memory bandwidth – how fast the chip can read model weights from unified memory.

Chip tier	Memory bandwidth	35B-A3B speed (MLX, Q4)
M1/M2 base	68-100 GB/s	20-30 tok/s
M1/M2/M3 Pro	200 GB/s	35-50 tok/s
M4 Pro	273 GB/s	45-60 tok/s
M1/M2/M3 Max	400 GB/s	55-75 tok/s
M4 Max	546 GB/s	65-85 tok/s
M2/M3 Ultra	800 GB/s	80-100+ tok/s

If your speed is well below these ranges, you're probably swapping. Open Activity Monitor, check Memory Pressure (green = fine, yellow/red = swapping), and either close apps or drop to a smaller quant. Our Ollama Mac troubleshooting guide walks through the full diagnostic.

Thinking mode on Mac

Qwen 3.5 supports thinking/non-thinking modes. In thinking mode, the model generates an internal chain-of-thought before responding. This improves reasoning quality on hard problems but generates many more tokens before you see output.

On a base M2 at 25 tok/s, a thinking response that generates 500 internal tokens before the visible answer starts means 20 seconds of waiting with nothing on screen. On an M4 Max at 70 tok/s via MLX, the same thinking process takes 7 seconds.

Disable thinking for conversational use:

```
# Ollama
ollama run qwen3.5:35b-a3b "/no_think What's the best coffee shop in Portland?"

# mlx_lm
mlx_lm.generate --model mlx-community/Qwen3.5-35B-A3B-4bit \
  --prompt "/no_think Summarize this paragraph:" --max-tokens 200
```

Enable thinking when you actually need it – a tricky bug, a math proof, anything where you'd want a human to stop and think before answering. The quality jump is real on those tasks.

The bottom line

For speed on Mac: use MLX. LM Studio with the MLX backend is the easiest path, mlx_lm is the scriptable one. You'll get roughly 2x the generation speed and 5x faster prompt processing compared to Ollama, plus lower memory usage.

For ecosystem on Mac: use Ollama. If your workflow depends on always-on API access, Open WebUI, or coding tools that speak Ollama, the speed trade-off is worth the integration convenience.

For the model itself: the 35B-A3B is the default pick for most Macs. It's the fastest Qwen 3.5 variant, strong on benchmarks despite only activating 3B parameters, and fits on 16GB at Q4. If you code seriously and have 24GB+, add the 27B dense model for its stronger SWE-bench scores. If you have 64GB, the 122B-A10B is the reason you bought that much RAM.

Related guides

- Qwen 3.5 for Local AI: Which Model, Which Quant, Which GPU
- LM Studio vs Ollama on Mac
- Best Local LLMs for Mac in 2026
- Ollama on Mac: Setup & Optimization
- 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/qwen35-mac-mlx-vs-ollama/

Free guides for running AI locally