Apple Neural Engine for LLM Inference: What Actually Works

March 5, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** The Apple Neural Engine is a dedicated AI accelerator in every M-series chip. It uses almost no power (2 watts vs 20 watts for the GPU) and is extremely efficient per operation. But most LLM tools like MLX and llama.cpp ignore it entirely and run on the GPU instead. Why? The ANE was designed for small, fixed-size neural networks like image classification, not variable-length text generation. Projects like ANEMLL are making it work for LLMs, achieving 47-62 tok/s on 1B models and ~9 tok/s on 8B models, at a fraction of the power draw. But MLX on the GPU does 93+ tok/s on the same 8B model. ANE inference is worth watching if you care about battery life or want to run a small model in the background all day. For raw speed on a Mac, MLX on the GPU still wins.

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

Every M-series Mac has a dedicated AI chip that most LLM users never touch. The Apple Neural Engine sits on the die, draws almost no power, and handles Apple Intelligence features like image segmentation, voice recognition, and on-device Siri processing. It's fast at those things.

For LLMs? It's complicated. The ANE wasn't designed for text generation, the software stack is opaque, and Apple hasn't made it easy to use for third-party inference. But people are making it work anyway, and the results are interesting enough to pay attention to.

What the Neural Engine is

The ANE is a fixed-function accelerator on Apple Silicon, separate from the CPU and GPU. Apple markets it by TOPS (trillions of operations per second): 38 TOPS on the M4, for example. But that number is misleading.

Reverse-engineering work by Manjeet Singh measured the M4 ANE directly through private APIs, bypassing CoreML. The real numbers:

Metric	M4 ANE
Marketed performance	38 TOPS
Actual FP16 throughput	19 TFLOPS
INT8 throughput	Same as FP16 (dequantized internally)
Peak power draw	~2.8 watts
Idle power	Zero (complete power gating)
Efficiency	~6.6 TFLOPS/W
On-chip SRAM	~32 MB

The “38 TOPS” counts INT8 operations at double the rate because each takes half the bits. In practice, INT8 and FP16 deliver the same compute throughput because the ANE converts INT8 weights to FP16 before doing the math.

The efficiency numbers are real though. At 6.6 TFLOPS per watt, the ANE is roughly 80x more efficient per operation than an A100 datacenter GPU. Zero watts at idle. That’s why Apple uses it for always-on features like “Hey Siri” detection.

Why LLM tools don’t use it

When you run a model with MLX, llama.cpp, Ollama, or LM Studio on a Mac, inference runs on the GPU through Metal. The ANE sits idle. There are real reasons for this.

The ANE was designed for convolutions, not text generation. It’s built around fixed-size neural network operations: image classification, segmentation, audio processing. LLMs need variable-length sequences and autoregressive token generation, which is a poor fit for the ANE’s static computation model.

CoreML adds overhead. The official way to use the ANE is through Apple’s CoreML framework. But CoreML adds 2-4x overhead on small operations compared to direct hardware access. For LLM token decoding, where each step generates a single token, that overhead kills throughput.

The ANE has a 32MB SRAM cache cliff. Singh’s benchmarks show that 2048x2048 matrix operations (24MB working set) hit 5.7 TFLOPS. Jump to 4096x4096 (96MB working set) and throughput drops 30%. LLM weight matrices regularly exceed this cache, causing performance to fall off.

Context windows are limited. ANE inference typically caps at 512-2048 tokens, with some implementations reaching 4096. Compare that to MLX or llama.cpp where 32K-128K context is routine.

Apple keeps the interfaces private. There's no public API for direct ANE access. CoreML is the approved path, and it's not optimized for LLM workloads. Researchers who've made progress have done so by reverse-engineering private `_ANEClient` APIs, which could break with any macOS update.

What works today

Two projects have made real progress running LLMs on the ANE.

ANEMLL

ANEMLL (Artificial Neural Engine Machine Learning Library) is the most practical option. It's an open-source pipeline that converts Hugging Face models to ANE-optimized CoreML format and runs inference on the Neural Engine.

Supported models: Llama 3.1/3.2 (1B, 8B), Qwen 2.5/3 (0.6B-8B), Gemma 3 (270M-4B), DeepSeek R1 8B, DeepHermes 3B/8B.

Current version: 0.3.5 Beta. Usable but still developing.

Requirements: macOS Sequoia, Apple Silicon, 16GB RAM minimum (32GB for 8B models), Python 3.9-3.11.

Installation:

```
brew install uv
./create_uv_env.sh
source env-anemll/bin/activate
./install_dependencies.sh
```

The conversion pipeline handles chunking, quantization, and the operator adaptations needed to fit transformer architectures onto the ANE. It's not a one-click process, but it works.

Maderix ANE

Manjeet Singh's ANE project goes deeper. It bypasses CoreML entirely, talking directly to the ANE hardware through reverse-engineered private APIs. Singh achieved something Apple said wasn't possible: training a neural network on hardware designed exclusively for inference.

This is research-grade work. The README says "a research project, not a production framework." Current utilization sits at 5-9% of peak ANE capacity, with many operations falling back to CPU. But it proved that the hardware barrier is artificial. The ANE can do more than Apple lets on.

Performance: ANE vs GPU

Here's where the tradeoffs get concrete. All numbers from community benchmarks on M4-series hardware.

Model	ANE (ANEMLL)	GPU (MLX, M4 Max)	ANE Power	GPU Power
Llama 3.2 1B	47-62 tok/s	~204 tok/s	~2W	~20W
DeepSeek R1 8B	~9.3 tok/s	~50 tok/s	~2W	~20W
Qwen3 8B	Not tested	~93 tok/s	-	~20W
Memory (8B model)	~500 MB	~8 GB	-	-

The GPU is 2-5x faster in raw token generation. That's the headline number and it's why most people should stick with MLX.

But look at the other columns. The ANE uses roughly one-tenth the power. And for the 8B model, ANEMLL used 500MB of memory while MLX used 8GB. Those numbers matter for specific use cases.

The M5 changes things (sort of)

The M5 introduced "Neural Accelerators" in the GPU, which is different from the ANE. These are dedicated matrix multiplication units inside the GPU pipeline that Apple's MLX team benchmarked at 3.3-4x faster time-to-first-token compared to M4, with 19-27% faster token generation.

This matters because it means Apple is bringing neural-network-style acceleration to the GPU rather than making the ANE better for LLMs. The M5's token generation improvement (19-27%) maps closely to its 28% higher memory bandwidth (153 GB/s vs 120 GB/s), confirming that generation speed is still memory-bound.

For LLM users, the M5's GPU Neural Accelerators through MLX are the practical path. The ANE remains a separate, lower-power option for specific workloads.

When ANE inference makes sense

Battery life matters more than speed. If you want a small model running in the background all day on a MacBook, the ANE's 2-watt draw vs the GPU's 20 watts means hours of extra battery. A 1B model at 50 tok/s on ANE is fast enough for many tasks.

Memory is tight. On a 16GB Mac running other applications, freeing 8GB by moving inference off the GPU to the ANE could be the difference between usable and swapping.

You're building always-on features. A background agent that monitors messages or summarizes notifications doesn't need 93 tok/s. It needs low power and no interference with the user's actual work. The ANE fits that.

You want to experiment. The ANE is interesting hardware. If you enjoy tinkering with low-level optimizations and don't mind beta software, ANEMLL is a real project doing real work.

When to stick with the GPU

You want speed. MLX on the GPU is 2-5x faster for token generation. For interactive chat or coding assistance, the GPU wins.

You need long context. ANE implementations cap around 2048-4096 tokens. MLX and llama.cpp handle 32K+ without issue.

You want stability. MLX is mature and updated by Apple's ML team. ANEMLL is beta software built partly on reverse-engineered APIs. One macOS update could break things.

You're running models larger than 8B. ANEMLL's tested range tops out at 8B parameters. For 14B, 27B, or larger models, the GPU through MLX is the only realistic path on Mac today.

Where this is headed

Apple has deliberately kept the ANE locked down for third-party LLM use. The efficiency numbers are real: 80x more efficient per FLOP than datacenter GPUs, zero idle power, tiny memory footprint. Apple uses it heavily for their own features.

For LLMs, it's early. ANEMLL works but it's beta, limited to 8B and below, and constrained to short context windows. The GPU through MLX is faster and better supported. Most Mac users running local models should use MLX and not think about the ANE at all.

The power efficiency gap is hard to ignore though. As local AI moves toward always-on background agents and on-device intelligence, a 2-watt inference engine gets more interesting than a 20-watt one. Apple's direction with the M5 Neural Accelerators suggests they see this too, even if they're bringing the acceleration to the GPU side rather than opening up the ANE.

If you have a Mac and want to experiment, install ANEMLL, convert a 1B model, and see what 50 tok/s on 2 watts feels like. It won't replace your MLX setup for daily use. But it might change how you think about where local inference is headed.

For getting the most out of your Mac's GPU right now, see our Mac M-series guide and best local LLMs for Mac.