Featured Highlight

Local and Distributed Large LLM Inference with MLX and RDMA on Mac Studio

October 15, 2026

MLX ↗ on Mac with Apple silicon lets AI researchers and developers experiment locally with the latest models and methods in a private environment. The open-source array framework leverages the unified memory architecture of Apple silicon, enabling anyone to run new inference and fine-tuning methods on their own hardware and explore other AI techniques. In a previous post ›, we highlighted the advantages of the M5 architecture and its GPU Neural Accelerators.

In this post, we show how to run the latest frontier-class model — Muse Spark Open, with nearly a trillion parameters — right from your desk, using the latest version of MLX, released today. On a cluster of four Mac Studios, Muse Spark Open generates over 36 tokens per second (see Figure 1), well above average human reading speed.

Beginning on a single Mac Studio (up to 512GB of unified memory), then scaling out across multiple Macs with the RDMA over Thunderbolt driver, below we share an overview of the range of models you can now run on Mac Studio.

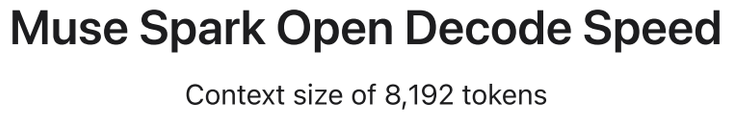

Figure 1: Muse Spark Open decode speed in tokens per second, on Mac Studio with M5 Ultra, at a context size of 8,192 tokens. Higher is better. We report performance for three parallelization schemes: pipeline parallelization over two Mac Studio computers (2 M5 Pipeline), and tensor parallelization over two (2 M5 Tensor) and four Mac Studio (4 M5 Tensor) computers.

Running Large Language Models with MLX

We benchmark several popular open-weight LLMs on Mac Studio ›, using the MLX LM package, as described in our prior post ›. We compare two generations of Mac Studio: the recently released M5 Ultra and the previous generation, M3 Ultra. The M5 generation features the GPU Neural Accelerator, which supports dedicated matrix multiplication operations, essential for many machine learning workloads.

We evaluate three frontier-class models (Table 1) spanning a wide range of sizes (in billions of parameters, B): the newly released Muse Spark Open (816 B), Mistral Medium 3.5 (128 B), and Qwen 3.8 (27 B). Muse Spark Open is a mixture of experts (MoE) model, which keeps only a fraction of its experts — and thus parameters — active, reducing compute at inference, and it is evaluated in NVFP4 precision. Mistral Medium 3.5 and Qwen 3.8 are dense architectures, where all parameters are active at inference, and they are evaluated in their native floating-point precision (FP8 and BF16, respectively).

Model	Total Parameters	Active Parameters	Native Precision	Evaluation Precision
Muse Spark Open	816B	42B	BF16	NVFP4
Mistral Medium 3.5	128B	128B	FP8	FP8
Qwen 3.8	27B	27B	BF16	BF16

Table 1: Language models used in this post. Models ship in various precision formats; we evaluate Qwen 3.8 and Mistral Medium 3.5 in their native precision, and Muse Spark Open in 4-bit (NVFP4) precision.

Our evaluation measures three metrics across a range of context sizes: time to first token (TTFT, in seconds), generation throughput (in tokens/second), and peak power usage (in watts). All the performance numbers in this post were obtained with `mlx_lm.benchmark`. For example, running Qwen 3.8 with 8,192 tokens of context and generating 128 tokens is performed with the command line:

```
mlx_lm.benchmark --model Qwen/Qwen3.8-27B -p 8192 -g 128 -b
```

A popular use of local LLMs on Mac Studio is running agentic experiences in a private environment, including agentic coding applications that often need large contexts. We therefore report performance at both 8,192 and 65,536 context sizes.

Raw TTFT and decode throughput are reported in Table 2 and Table 3, respectively.

Inference Performance on a Single Mac Studio

Qwen 3.8 27B and Mistral Medium 3.5 128B fit comfortably on a single Mac Studio with 512GB, taking up to 68GB and 158GB of RAM in the large context size (65,536 tokens) setup. In Figure 2, we compare their inference performance on a single Mac Studio with 512GB of RAM, both on M3 Ultra and M5 Ultra.

Time to first token (TTFT) is essentially compute bound, and that is where the Neural Accelerators in the M5 Ultra GPU shine: the specialized hardware support for fast matrix-multiplication of the GPU Neural Accelerator directly translates into significant speedups for TTFT, up to 4.6× faster compared to an M3 Ultra (see Figure 2). That makes the Mac Studio with M5 Ultra the desktop of choice when it comes to handling tasks involving very large contexts.

Generation speed (in tokens per second) is memory-bandwidth bound. The M5 Ultra chip delivers 1.2 TB/s of unified memory bandwidth, up from 819 GB/s for the M3 Ultra. Here again, that translates directly into generation speed (see Figure 3), with the Mac Studio with M5 Ultra reporting 1.6× faster tokens/s.

TTFT Speedup

Single Mac Studio, M5 Ultra over an M3 Ultra baseline (1.0×)

Figure 2: Time to first token (TTFT) speedup on M5 Ultra over an M3 Ultra baseline, for Qwen 3.8 and Mistral Medium 3.5, at context sizes of 8,192 and 65,536 tokens. The M5 Ultra GPU Neural Accelerator shines on compute-bound workloads.

Decode Speedup

Single Mac Studio, M5 Ultra over an M3 Ultra baseline (1.0×)

Figure 3: Decode speedup on M5 Ultra over an M3 Ultra baseline, for Qwen 3.8 and Mistral Medium 3.5, at context sizes of 8,192 and 65,536 tokens. Decode is memory-bandwidth bound, and the M5 Ultra's 1.2 TB/s of unified memory bandwidth (vs 819 GB/s on the M3 Ultra) is a key advantage.

The Mac Studio is also energy-efficient for LLM inference (see Figure 4): peak power consumption is about 200W for the Mac Studio with M3 Ultra, and 300W for the Mac Studio with M5 Ultra — a fraction of the power used by a common toaster. More importantly, that modest amount of power consumption means a Mac Studio can be deployed for AI work in any home or office environment.

Peak Power Consumption

Context size of 65,536 tokens

Figure 4: Peak power consumption (in watts) during Qwen 3.8 and Mistral Medium 3.5 inference, at a context size of 65,536 tokens, on Mac Studio with M3 Ultra and with M5 Ultra.

While both the Qwen and Mistral models fit on a single Mac Studio with 512GB of RAM, Muse Spark Open would require more than 1.6TB of RAM in its native precision (BF16) for the model parameters alone. In a 4-bit quantized (NVFP4) format, the model memory footprint is lowered to about 493GB. Add the memory needed for intermediate computations during inference, and the model cannot fit on a single machine. It requires parallelization across multiple Mac Studio computers.

Pipelining Inference Across Two Mac Studio Computers

Very large frontier-class models like Muse Spark Open will not fit on a single machine. It is, however, possible to store half of the model on one Mac Studio and the other half on another. We can then pipeline the model's inference across two computers: the first Mac Studio executes the first half of the model's layers, then sends the output to the second Mac Studio, which executes the second half. This parallelization scheme is called pipeline parallelization (see Figure 5) and is extremely simple to implement, requiring only minimal code changes.

Splitting Large Models Across Multiple Computers

Figure 5: MLX supports two approaches to splitting large models across multiple computers. With pipeline parallelization, a model is split across depth, while with tensor parallelization, a model is split across width.

MLX supports different parallelization schemes, as described in the MLX distributed documentation ↗. These rely on fast inter-machine communication: macOS 26.2 › introduced a low-latency RDMA driver over Thunderbolt 5, and MLX (v0.30.1 and later) builds on top of this driver and provides new communication collectives (`jaccl` , pronounced "jackal") for different parallelization schemes. In our setup, the two Mac Studios are connected with two Thunderbolt cables (to double the link bandwidth), providing the RDMA-over-Thunderbolt link that `jaccl` uses. Once the hosts are set up, MLX provides `mlx.launch` , a helper command to launch jobs across multiple nodes:

```
mlx.launch --hostfile hostfile.json --backend jaccl --env ML
mlx_lm.benchmark --model mistralai/Mistral-Medium-3.5-128B -
```

In this command, `--backend jaccl` specifies the RDMA Thunderbolt driver backend. The `--env MLX_METAL_FAST_SYNCH=1` flag enables an experimental MLX feature for fast CPU-GPU synchronization, reducing communication overhead between the

CPU (which initiates the Thunderbolt communication) and the GPU (which does the compute).

In general, decode inference speed with pipeline parallelization (see Figure 7) is very similar to that of a single machine, since the layers are still evaluated sequentially. Communication overhead is also very low, as only the intermediate outputs between the two sets of layers need to be exchanged.

It is however notable that for very large contexts (65,536 tokens in our benchmark), pipeline parallelization ends up faster than a single node for prefill — up to ~1.8× faster (see Figure 6). This is because MLX segments large contexts into smaller chunks (here, 8,192 tokens, as set by `--prefill-step-size 8192`) and pipelines the prefill across the machines: while the second machine computes the later layers of chunk k, the first machine computes the earlier layers of chunk $k+1$, so consecutive chunks overlap in time. For even better compute parallelization, we next turn to tensor parallelization.

TTFT Speedup

Two Mac Studios with pipeline parallelization, over a single M5 Ultra baseline (1.0×)

Figure 6: Time to first token (TTFT) speedup of two M5 Ultras with pipeline parallelization over a single M5 Ultra baseline, for Qwen 3.8 and Mistral Medium 3.5, at context sizes of 8,192 and 65,536 tokens. Splitting the context (here into 8,192-token chunks) lets MLX overlap computation across the two machines — hence the speedup at 65,536 but not 8,192.

Decode Speedup

Two Mac Studios with pipeline parallelization, over a single M5 Ultra baseline (1.0×)

Figure 7: Decode speedup of two M5 Ultras with pipeline parallelization over a single M5 Ultra baseline, for Qwen 3.8 and Mistral Medium 3.5, at context sizes of 8,192 and 65,536 tokens. Pipeline parallelization leaves decode speed essentially unchanged.

Tensor Parallelization for Faster Inference

Tensor parallelization (see Figure 5) distributes the LLM's width across several computers, in contrast to pipeline parallelization, which distributes the model across its depth. Each machine now holds a fraction of each of the layers of the model, such that per-layer computation is parallelized. When efficiently implemented, tensor parallelization improves inference speed. However, tensor parallelization is more complex to implement than pipeline parallelization: each layer must be parallelized, which increases code complexity. The scheme also involves many small communication operations: for each input token and for each layer, one must broadcast inputs and aggregate outputs. As a result, a very low-latency communication technology like Thunderbolt RDMA, supported by MLX `jaccl`, is critical for this approach to yield any speedup.

With MLX's high-level `mlx.launch` helper, it is easy to switch to tensor parallelization. The Mac Studio computers are connected in a Thunderbolt mesh, with each pair of machines linked peer-to-peer by two Thunderbolt cables. Assuming the hosts are set up, we simply launch the benchmark across multiple machines with:

```
mlx.launch --hostfile hostfile.json --backend jaccl --env ML
mlx_lm.benchmark --model meta-models/Muse-Spark-Open-816B-A4
```

Figure 8 and Figure 9 show the speedups obtained with two or four Mac Studio computers with M5 Ultra, relative to a single-machine baseline (or, for Muse Spark Open, the pipeline parallelization baseline). Notably, dense architectures scale better across multiple computers.

With a small number of machines, the communication overhead of tensor parallelization can offset some of its compute benefits at very large contexts, though this applies to prefill only. This shows up with Muse Spark Open, where separate normalization of the experts' outputs adds some prefill communication, enough that at very large contexts (65,536 tokens), two-machine tensor parallelization lags behind two-machine pipeline parallelization. The approach however pays off as machines are added, as seen with four M5 Ultras. Decode is unaffected: Muse Spark Open sustains a significant decode speedup under tensor parallelization even at large contexts, comparable to dense models like Qwen 3.8 and Mistral Medium 3.5.

The two-cable mesh also pays off during prefill, where the extra link bandwidth is felt. For Muse Spark Open on four M5 Ultras at a 65,536-token context, two cables per link cut TTFT from about 50.5s (one cable) to 40.6s — the two-cable value reported in Table 2, the one-cable run being an additional experiment.

TTFT Speedup

Two or four Mac Studios with tensor parallelization, over a single M5 Ultra baseline (1.0×; Muse Spark Open: two-machine pipeline)

2 machines 4 machines

Figure 8: Time to first token (TTFT) speedup of two or four M5 Ultras with tensor parallelization over a single M5 Ultra baseline, for Qwen 3.8, Mistral Medium 3.5, and Muse Spark Open, at context sizes of 8,192 and 65,536 tokens. Muse Spark Open does not fit on a single machine, so its baseline is the two M5 Ultra pipeline configuration.

Decode Speedup

Two or four Mac Studios with tensor parallelization, over a single M5 Ultra baseline (1.0×; Muse Spark Open: two-machine pipeline)

2 machines 4 machines

Figure 9: Decode speedup of two or four M5 Ultras with tensor parallelization over a single M5 Ultra baseline, for Qwen 3.8, Mistral Medium 3.5, and Muse Spark Open, at context sizes of 8,192 and 65,536 tokens. Muse Spark Open does not fit on a single machine, so its baseline is the two M5 Ultra pipeline configuration.

Time to First Token (seconds)

Lower is better

8,192 tokens 65,536 tokens

Model	M3 Ultra	M5 Ultra	2 M5 Pipeline	2 M5 Tensor	4 M5 Tensor
Muse Spark Open	—	—	8.4	8.2	4.8
Mistral Medium 3.5	95.9	24.6	24.5	14.4	7.5
Qwen 3.8	18.2	4.1	4.1	3.1	1.7

Table 2: Time to first token (TTFT) in seconds for Qwen 3.8, Mistral Medium 3.5, and Muse Spark Open, at context sizes of 8,192 and 65,536 tokens. Lower is better. We report a single M3 Ultra, a single M5 Ultra, two M5 Ultras (pipeline and tensor parallelization), and four M5 Ultras (tensor parallelization).

Decode Speed (tokens per second)

Higher is better

8,192 tokens 65,536 tokens

Model	M3 Ultra	M5 Ultra	2 M5 Pipeline	2 M5 Tensor	4 M5 Tensor
Muse Spark Open	—	—	15.74	24.99	36.73
Mistral Medium 3.5	4.95	7.70	7.70	13.69	23.37
Qwen 3.8	12.46	19.69	19.44	33.27	50.89

Table 3: Decode speed in tokens per second for Qwen 3.8, Mistral Medium 3.5, and Muse Spark Open, at context sizes of 8,192 and 65,536 tokens. Higher is better. We report a single M3 Ultra, a single M5 Ultra, two M5 Ultras (pipeline and tensor parallelization), and four M5 Ultras (tensor parallelization). Decode speed was measured over 128 generated tokens following the context.

Local Agentic AI on Mac

Recently, agentic AI on Mac has gained considerable traction, with agentic coding tools seeing widespread adoption among developers. With MLX, powering these tools locally is as simple as launching a server with the `mlx_lm.server` tool. For example, a server running Muse Spark Open on port 8080 is started with:

```
mlx.launch --hostfile hostfile.json --backend jaccl --env ML
mlx_lm.server --model meta-models/Muse-Spark-Open-816B-A42B-
```

As with `mlx_lm.benchmark`, the `mlx_lm.server` helper can be prefixed with `mlx.launch` to parallelize computation across multiple Mac Studios. From there, agentic AI can be enabled with tools like OpenCode ↗, which can be easily configured to use that local LLM server:

```
{
  "$schema": "https://opencode.ai/config.json",
  "model": "meta-models/Muse-Spark-Open-816B-A42B-NVFP4",
  "provider": {
    "mlx": {
      "npm": "@ai-sdk/openai-compatible",
      "name": "Local MLX",
      "options": {
        "baseURL": "http://127.0.0.1:8080/v1"
      },
      "models": {
        "meta-models/Muse-Spark-Open-816B-A42B-NVFP4": {
          "name": "Muse Spark Open 816B (MLX)"
        }
      }
    }
  }
}
```

Many other local agentic AI experiences can be enabled through a similar approach.

Conclusion

Over the past year, we have worked on several key innovations in MLX to make it easier and more efficient to run state-of-the-art LLMs locally on recent Apple silicon. These include software support for the Neural Accelerators in the GPU of the M5 family of chips, low-latency communication across several Macs using RDMA over Thunderbolt, and efficient implementations of recent popular architectures such as mixture of experts (MoE). Each of these helps on its own, but it is their combination with the latest Apple silicon — Mac Studio with M5 Ultra, Neural Accelerators in the M5 Ultra GPU, 512GB of unified memory, and RDMA over Thunderbolt 5 — that now lets us run models like Muse Spark Open with more than 800B parameters locally, at up to 4.6× the start-up (time-to-first-token) speed of the previous generation of Mac Studio.

Get Started with MLX

- Learn more ↗
- MLX LM on GitHub ↗
- MLX Hugging Face community ↗
- MLX distributed support ↗

Related readings and updates.

November 19, 2025

Exploring LLMs with MLX and the Neural Accelerators in the M5 GPU

April 17, 2026 | General

International Conference on Learning Representations (ICLR) 2026

Apple is presenting new research at the annual International Conference on Learning Representations (ICLR) ↗, which takes place in person in Rio de Janeiro, Brazil, from April 23 to 27. We are proud to again sponsor the conference, which brings together the scientific and industrial research communities focused on deep learning. Below is an overview of Apple's participation at ICLR 2026.

Mac with Apple silicon is increasingly popular among AI developers and researchers interested in using their Mac to experiment with the latest models and techniques. With MLX, users can explore and run LLMs efficiently on Mac. It allows researchers to experiment with new inference or fine-tuning techniques, or investigate AI techniques in a private environment, on their own hardware. MLX works with all Apple silicon systems, and with the latest...

Read more

Discover opportunities in Machine Learning.

Our research in machine learning breaks new ground every day.

Work with us

Privacy Policy Terms of Use Legal

Copyright © 2026 Apple Inc. All rights reserved.