Architectural Characteristics of Apple Silicon for Machine Learning Applications

TAO GROVES*, University of Virginia, USA

A study is presented comparing the performance of Apple Silicon and NVIDIA GPUs across a range of machine learning workloads. A range of models and operation are measured, and conclusions are drawn about the priorities of each architecture.

CCS Concepts: • **Hardware** → **Emerging architectures**; • **Computing methodologies** → **Neural networks**; • **Applied computing** → *Service-oriented architectures.*

Additional Key Words and Phrases: Apple Silicon, Machine Learning, CUDA, Language Model, Convolution, High-Performance Computing, Computer Architecture, GPU

ACM Reference Format:

Tao Groves. 2025. Architectural Characteristics of Apple Silicon for Machine Learning Applications. 1, 1 (May 2025), 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Abstract

Since the introduction of Apple Silicon, its applicability to machine learning has been a topic of growing interest, particularly in light of its distinct GPU architecture and unified memory design. In this paper, we systematically benchmark Apple Silicon GPUs against NVIDIA CUDA across a range of fundamental operations, including matrix-matrix multiplication, matrix-vector multiplication, and convolution. We also compare the performance and developer experience of ML frameworks optimized for Apple hardware, namely PyTorch with Metal Performance Shaders (MPS) and Apple's MLX. Our results reveal not only the performance trade-offs between platforms but also provide insight into which operations Apple Silicon appears to be optimized for at the hardware level. We hypothesize how Apple's architectural choices—such as high memory bandwidth, large shared caches, and tightly integrated CPU-GPU pipelines—shape its performance profile and influence its suitability for different classes of machine learning workloads.

2 Introduction

Apple's M-series SoCs integrate CPU, GPU, Neural Engine, and I/O onto a single chip, sharing a unified pool of high-bandwidth RAM. [6] [5] For example, the M3 Ultra can deliver 800GB/s of system memory bandwidth [1] , which, while slower than modern dedicated VRAM, is not similarly bottlenecked by a PCIe bus and allows memory to be instantly shared between the CPU and GPU. Architecturally, Apple-designed GPUs scale from 7–8 cores (in low-end M1) up to 64 (M1 Ultra/M4) or 80 (M3 Ultra) cores, each split into multiple Execution Units (EUs). . The latest 80-core GPU (in M3 Ultra) reaches about 115 TFLOPs for dense FP16 matrix operations, significantly lower than the 209.5 TFLOPs achieved by a comparable NVIDIA GPU (RTX 5090). [2] While raw performance is significantly lower, Apple's unified design has some key advantages. Because tensors do not have to be copied to VRAM and back, there is much less downtime

Author's Contact Information: Tao Groves, groves@virginia.edu, University of Virginia, Charlottesville, Virginia, USA.

Manuscript submitted to ACM

between computations. For smaller tensors or interactive workloads, this unified-memory zero-copy model can reduce overhead and latency compared to discrete-GPU systems. [4] This advantage is much more meaningful when working with small models or when models cannot be held in memory long-term (as is often the case on consumer devices, which Apple Silicon is primarily designed for). Additionally, its newer ARM-based architecture is significantly lighter and more energy-efficient. However, the fact remains that Apple Silicon is not currently competitive for most HPC applications, where performance per dollar is more important than performance per watt. This paper investigates the cause of, reasons for, and implications of this performance gap, focusing on image- and text-generative models.

3 Method

To evaluate machine learning performance across architectures, we conducted a series of benchmarks on three GPUs:

- Apple M1 Max with 32-core GPU, 32GB unified memory, 400GB/s bandwidth
- NVIDIA RTX 2080 with 8GB VRAM, 448GB/s bandwidth
- NVIDIA Quadro A6000 with 48GB VRAM, 768GB/s bandwidth

All benchmarks were run under comparable conditions using native PyTorch backends to simulate a realistic environment (Metal Performance Shaders on Apple; CUDA/cuDNN on NVIDIA), with thermal throttling and background processes minimized to aid reproducibility.

We first measured end-to-end training performance on three representative ML workloads: ResNet-50, U-Net (in multiple parameter configurations), and Stable Diffusion 2.0. For each model, we recorded peak memory usage, forward-pass latency, and backward-pass latency across a range of batch sizes and input resolutions. These metrics capture both raw computational throughput and memory efficiency, and enable comparisons of training viability on-device.

To isolate architectural and kernel-level performance differences, we next benchmarked low-level linear algebra operations—specifically, matrix–vector products and vector–Jacobian products—across varying input sizes, data distributions, floating-point precisions (FP32, FP16), and iteration counts. These experiments were performed only on the M1 Max and Quadro A6000, chosen to contrast a unified-memory GPU with a high-end discrete GPU. Timing results were averaged over multiple warm-started iterations, and memory transfer times were excluded where zero-copy access was supported.

Finally, we analyzed convolutional kernels and other common ML operations to infer which workloads are most optimized for on Apple Silicon. This involved measuring isolated layer-level performance in convolution, normalization, and activation pipelines, enabling identification of operations where Apple's GPU/compiler stack (e.g., Metal or MLX) demonstrates either bottlenecks or unusually high throughput.

Together, these methods provide a multi-scale view of machine learning performance on Apple Silicon relative to CUDA, from low-level numerical ops to full-model training.

4 Results

4.1 End-To-End Training (Comparative)

4.1.1 Memory Usage. Stable Diffusion 2.0 (SD) was the most memory-intensive model on all devices, due to its large input sizes and complex architecture. Notably, the M1 Max reached its maximum available memory (28GB) during SD training, while the Quadro A6000 utilized up to 47.5GB. The RTX 2080 was unable to load SD at all due to insufficient VRAM.

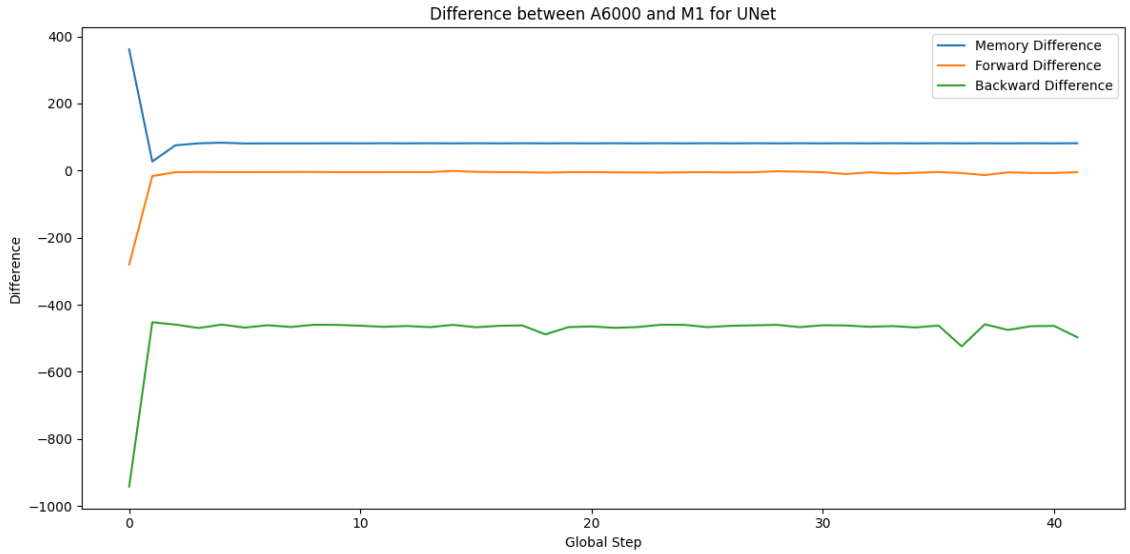

Fig. 1. Memory usage and latency for UNet on M1 Max vs. A6000

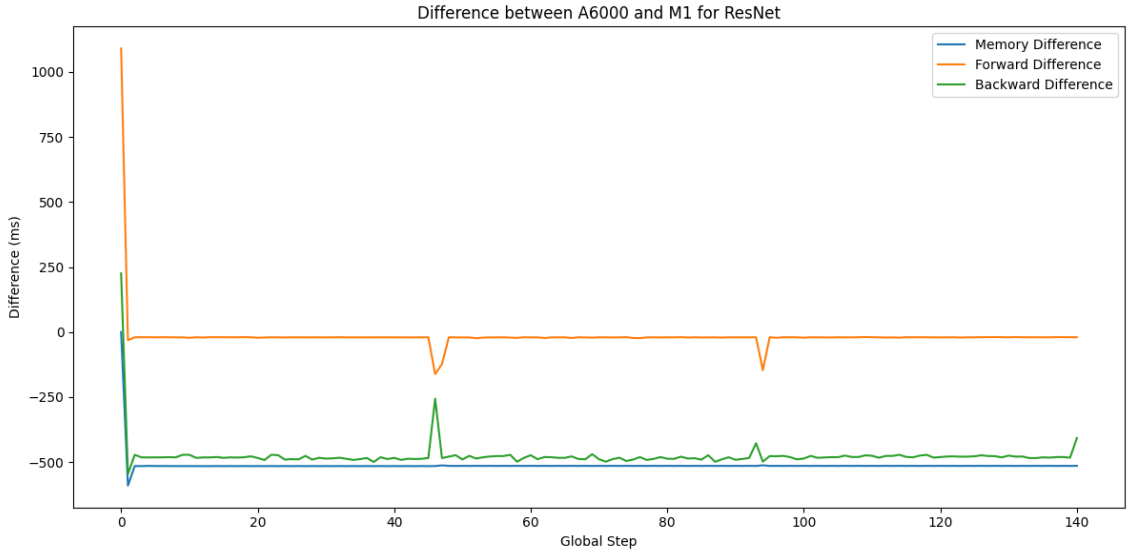

Fig. 2. Memory usage and latency for ResNet-50 on M1 Max vs. A6000

For smaller models, ResNet-50 and U-Net, memory usage patterns diverged across GPUs. On Apple Silicon, ResNet consumed slightly more memory than U-Net. Conversely, on NVIDIA GPUs, U-Net required approximately 2× more memory than ResNet. Interestingly, the M1 Max's overall memory usage was inconsistent—requiring up to 25% more

memory than the Quadro A6000 for some models, but slightly less for others. For example, during U-Net training, the M1 used 50MB less memory on average than the other GPUs.

4.1.2 Forward and Backward Latency. Forward-pass latency remained relatively consistent across all models and GPUs. The M1 Max exhibited slightly higher latency on average, but the difference was marginal—approximately 15ms slower for SD, and negligible for smaller models.

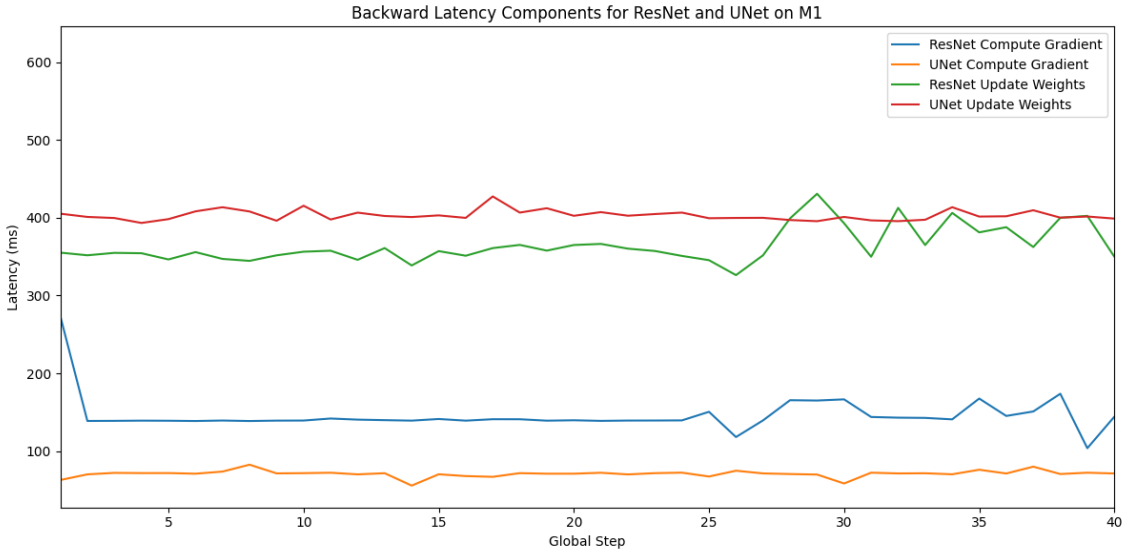

Fig. 3. Contribution of each backward pass step to overall backward latency for M1 Max

Backward-pass latency revealed much sharper differences. As model complexity increased, the M1 Max showed significantly degraded performance relative to the NVIDIA GPUs. For instance, during U-Net training, the Quadro A6000 completed backpropagation 480ms faster than the M1, while during SD training it was 600ms faster. Notably, profiling revealed that the latency bottleneck on the M1 occurred entirely during the weight update step, rather than during gradient computation. On the RTX 2080 and A6000, backward-pass components were closely matched in duration, suggesting better kernel-level optimization and scheduling.

In terms of overall training throughput, the RTX 2080 was $\tilde{4}\times$ faster and the Quadro A6000 $\tilde{8}\times$ faster than the M1 Max for smaller models. This performance gap widened dramatically for SD, where the A6000 achieved a 32× speedup, though these results were less reliable due to memory constraints and dataset limitations on the M1.

4.1.3 Mixed-Precision Training. One significant downside to current Apple Silicon is its lack of support for mixed-precision training, which can significantly speed up computation. Mixed-precision was tested on the NVIDIA GPUs to demonstrate. Across all GPUs, mixed-precision increased both forward and backward latency slightly (by $\tilde{2}$.5ms forward, and up to 40ms backward on SD), but roughly doubled overall training throughput. This discrepancy likely stems from improved GPU occupancy and lower memory bandwidth pressure, despite minor overheads in casting and managing lower-precision data. Mixed-precision did not reduce memory usage on any GPU.

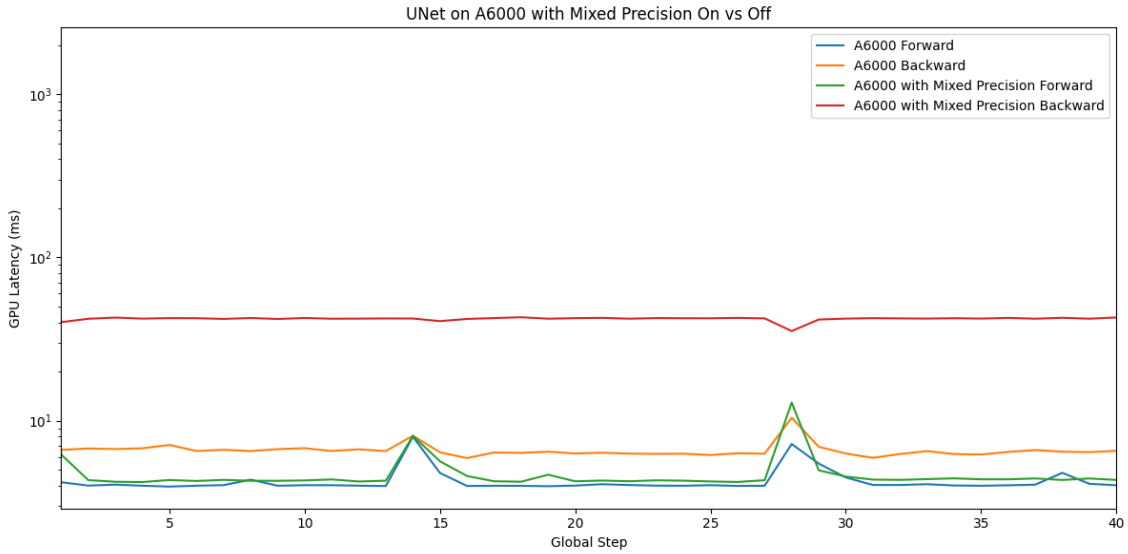

Fig. 4. UNet forward/backward latency on A6000 with mixed-precision on vs. off

In addition to specifying the *template style* to be used in formatting your work, there are a number of *template parameters* which modify some part of the applied template style. A complete list of these parameters can be found in the *LaTeX User's Guide.*

4.2 Matrix-Vector and Vector-Jacobian Products (Comparative)

4.2 Matrix–Vector and Vector–Jacobian Product Performance To evaluate low-level linear algebra throughput, we benchmarked matrix–vector and vector–Jacobian multiplications across a variety of configurations using MLX on Apple Silicon (M1 Max) and CUDA on the Quadro A6000. The test setup involved multiplying a matrix of shape $[B \times T \times D]$ with a vector of shape $[B \times D \times 1]$, where B=1 corresponds to a 2D matrix–vector product. We varied matrix size, floating-point precision (FP32 vs. FP16), data value range, and iteration count to capture performance across realistic machine learning workloads.

4.2.1 Performance Scaling and Architectural Characteristics. Our results show clear architectural tendencies. CUDA consistently outperformed MLX, and the performance gap widened with increasing matrix size. This indicates that NVIDIA's tensor cores and memory pipeline are heavily optimized for large, throughput-oriented matrix operations—consistent with their intended deployment in large-batch deep learning workloads.

In contrast, MLX was more competitive with CUDA on small matrices, particularly in the 32×32 to 64×64 range. In some cases, especially for matrices commonly seen in convolutional layers, MLX even outperformed CUDA when using FP32, although this advantage diminished quickly as matrix size increased. This pattern supports the hypothesis that Apple's GPU architecture is optimized for many small, fast operations rather than large batched workloads.

4.2.2 Precision Effects: FP16 vs. FP32.

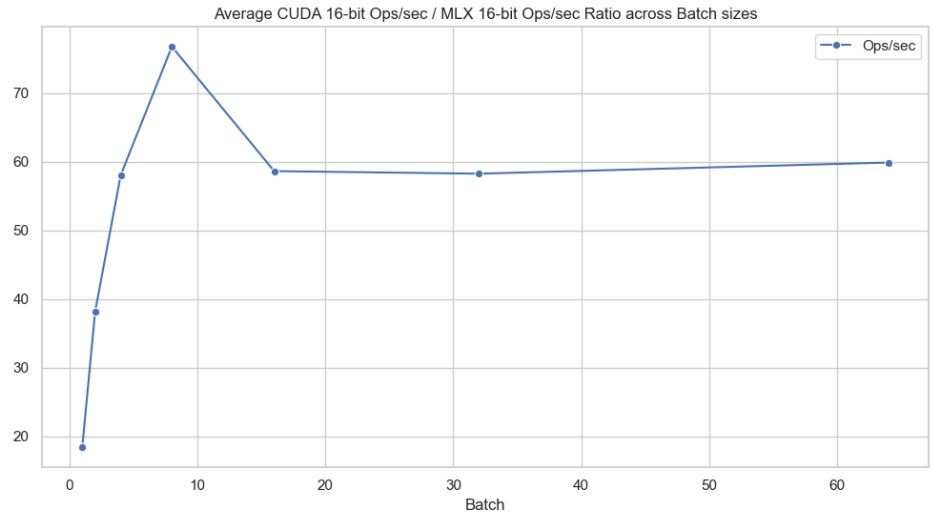

Fig. 5. Performance ratio of CUDA/MLX for FP16 multiplication

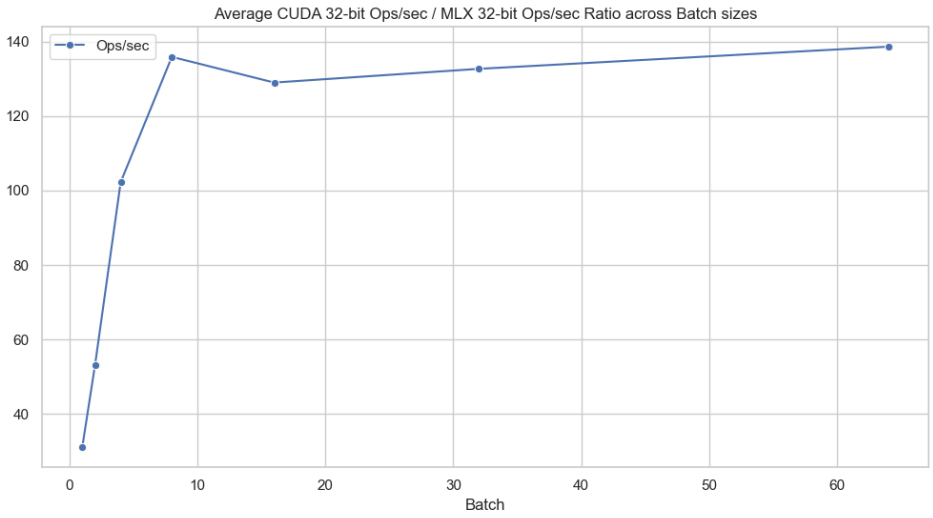

Fig. 6. Performance ratio of CUDA/MLX for FP32 multiplication

4.2.3 Precision Effects: FP16 vs. FP32. Precision had different effects on each platform. On CUDA, FP16 operations were consistently 1.25× faster than FP32, with little variation as matrix size changed. However, on MLX, the performance delta between FP16 and FP32 increased dramatically with matrix size—small matrices saw little benefit or even worse

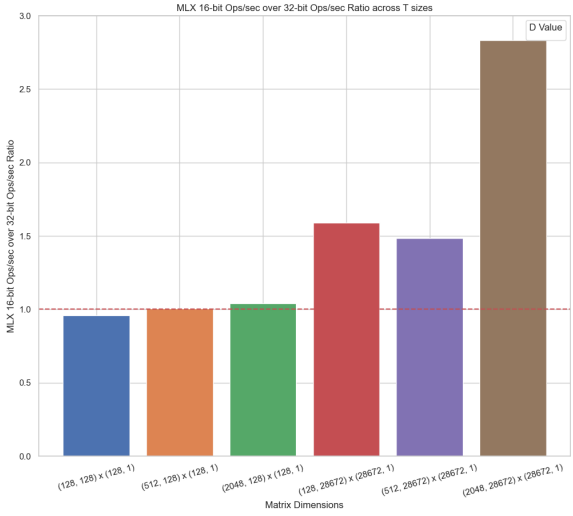

Fig. 7. MLX performance ratio of FP16/FP32 across matrix sizes

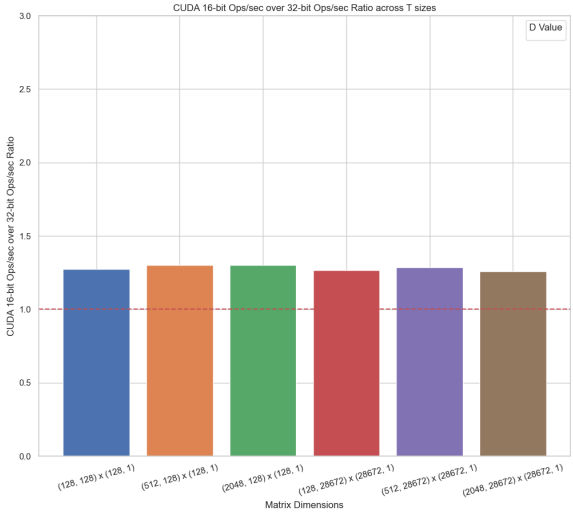

Fig. 8. CUDA performance ratio of FP16/FP32 across matrix sizes

performance with FP16, while large matrices saw 2× or more speedup. This nonlinearity suggests that MLX benefits from FP16 only once the overhead of precision conversion and operation setup is amortized over enough compute.

Notably, for certain very small matrix sizes—often used in convolution kernels—FP32 was faster than FP16 on MLX. This suggests that Apple's compiler and hardware may fuse or vectorize FP32 operations more effectively in these cases, or that FP16 incurs overhead at small sizes due to memory alignment or kernel dispatch constraints.

4.2.4 Data Value Effects.

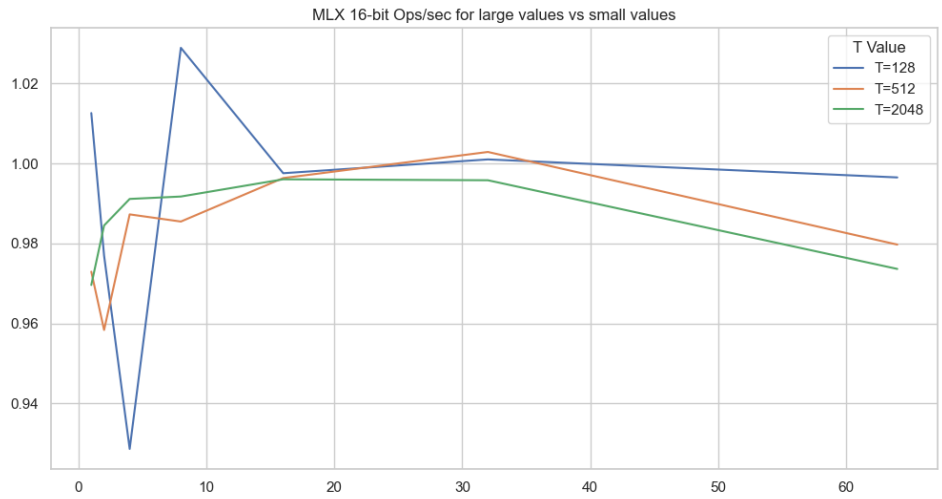

Fig. 9. MLX performance ratio of large/small matrix values across matrix sizes

4.2.5 Precision Effects: FP16 vs. FP32. We also tested how the range of matrix values influenced performance. While CUDA showed negligible variation, MLX ran marginally faster on matrices with values in $[0, 1]$ compared to larger values. This effect became more pronounced at larger matrix sizes, but the absolute performance delta remained small. Without detailed knowledge of Apple's internal floating-point representation, it's difficult to determine whether this behavior stems from numerical encoding optimizations, quantization artifacts, or memory alignment effects. Nonetheless, the observation suggests that operand magnitude has a non-negligible influence on Apple's kernel execution time, particularly for large inputs.

4.2.6 Operation Setup Overhead. Profiling revealed that a significant portion of total execution time on MLX was spent preparing operations, rather than performing arithmetic. For small matrices, operation setup and kernel dispatch took up a disproportionate amount of time, leading to reduced throughput. Throughput (ops/sec) increased nonlinearly with matrix size, particularly when the larger dimension was increased. This scaling behavior implies that Apple's pipeline incurs fixed per-operation overhead, which becomes less impactful as matrix size—and therefore arithmetic intensity—increases. CUDA showed similar throughput scaling, meaning the theoretical benefit of shared memory on Apple Silicon was not realized in our tests.

4.2.7 Matrix–Matrix Multiplication (MLX Only). To further investigate scaling, we performed matrix–matrix multiplication benchmarks on MLX using matrices of size $(a, b)(b, c)$, varying all dimensions. Interestingly, ops/sec grew

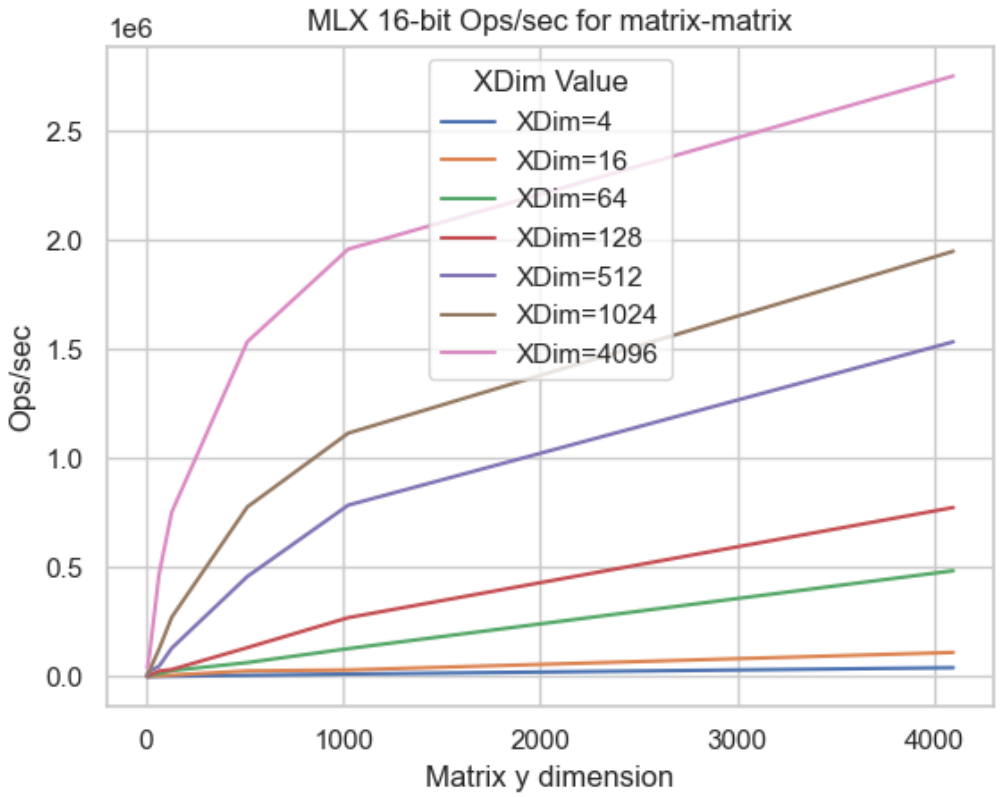

Fig. 10. MLX performance for matrix-matrix multiplication across matrix dimension variations

logarithmically with matrix size, especially when increasing the smaller dimension b. This behavior suggests that MLX's performance scales better when matrices are more square, which aligns with the theoretical properties of modern block matrix multiplication algorithms. However, another likely contributing factor is that operation setup dominates execution for smaller or highly skewed matrices, and increasing size reduces the proportional cost of kernel overhead.

4.3 Convolution and Theories

Unlike the general matrix–vector and matrix–matrix multiplication benchmarks, convolutional operations presented an interesting inversion in performance dynamics: MLX on Apple Silicon consistently outperformed CUDA on NVIDIA GPUs across a range of kernel sizes and input shapes, often by a significant margin, and even when mixed-precision was used on the NVIDIA GPUs. This finding was robust across both FP32 and FP16 precision and held for input tensors corresponding to typical convolutional layers used in image processing and computer vision models.

MLX achieved faster execution times for standard 2D convolutions with small to moderate kernel sizes (e.g., 3×3 and 5×5). CUDA consistently trailed behind MLX in raw throughput for equivalent convolution workloads. This performance edge is consistent with all findings in showing that Apple Silicon performs much better when the majority

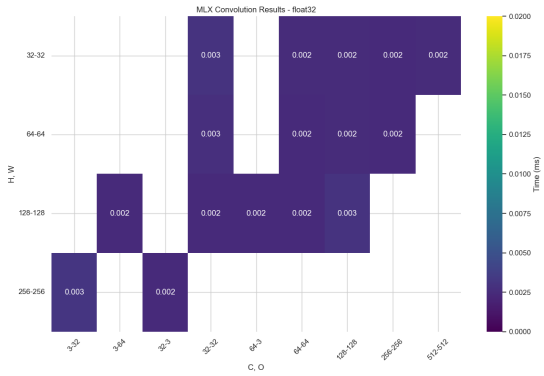

Fig. 11. Heatmap of MLX computation time for FP32 5x5 convolution (lower is better)

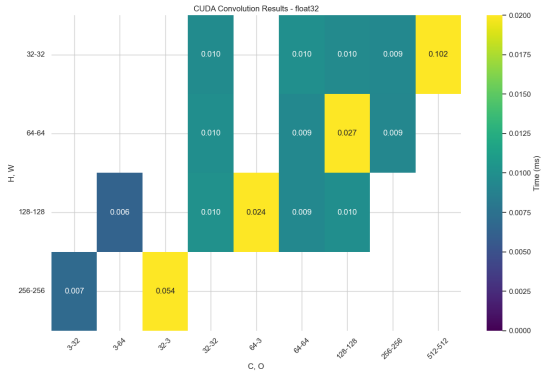

Fig. 12. Heatmap of CUDA computation time for FP32 5x5 convolution (lower is better)

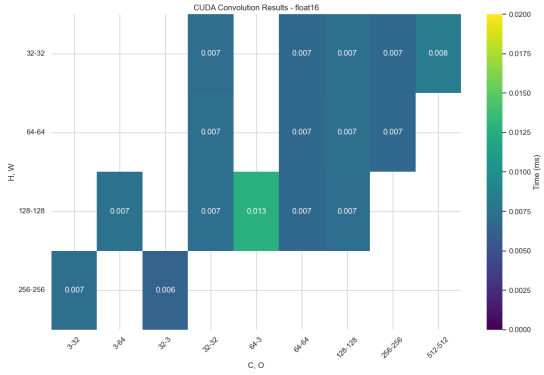

Fig. 13. Heatmap of CUDA computation time for FP32 5x5 convolution (lower is better)

of computation is many small, square matrix operations. As established in the previous section, CUDA held a substantial advantage for larger computations, particularly outside of vision-oriented layers. The reversal observed here suggests

that Apple Silicon has been explicitly tuned at the hardware level for convolutional workloads—a likely consequence of its intended use cases and design priorities.

4.3.1 Architectural Implications. This finding is consistent with Apple's vertical integration strategy and the company's emphasis on real-time image processing, computer vision, and on-device machine learning. Apple's GPUs are designed not for general-purpose training of large language models but for efficient execution of a wide range of image manipulation and inference tasks—from Photo app filters to real-time object segmentation in AR and the Camera app.

In the context of Apple Intelligence and similar on-device AI initiatives, convolution remains a foundational operation. From edge detection and segmentation to style transfer and facial recognition, convolutions underpin the vast majority of compute-intensive image transformations. It follows that Apple would prioritize optimizations for convolution kernels—potentially including:

- Specialized convolution accelerators or fused compute pipelines
- Optimized memory access patterns for common kernel shapes
- Lower dispatch and setup latency for tiled convolutions
- Tighter integration with unified memory for small image tiles

These architectural choices may also be influenced by power and thermal constraints. Accelerating convolutions efficiently enables high-throughput real-time processing within the tight thermal envelope of mobile devices. This is less of a priority for CUDA, which targets large data center GPUs where maximizing throughput for large-scale training takes precedence over thermal efficiency.

These results also align with prior research about Apple Silicon. [3] Specifically:

- The MLX framework shows performance benefits on small, vision-oriented tensor operations, suggesting hardware–software co-design.
- Apple's Neural Engine, while not directly benchmarked here, is known to offload certain convolution operations in inference scenarios, which may influence how the GPU pipeline is optimized for training workloads that resemble inference patterns.
- Convolutions are inherently local operations with predictable memory access patterns, which map well to tiled memory hierarchies like those on Apple GPUs.

5 Conclusion

This study set out to investigate how Apple Silicon compares to traditional NVIDIA CUDA-based GPUs for machine learning workloads, with a focus on understanding architectural trade-offs and performance characteristics. By benchmarking a range of models alongside fundamental operations, we aimed to assess both high-level training performance and low-level computational behavior on Apple Silicon.

Our results confirm a significant gap in training performance between Apple Silicon and NVIDIA GPUs, especially as model size and memory requirements increase. M1 Max exhibited notably higher backward pass latency, primarily due to overhead in applying gradients, and lagged behind by factors ranging from 4× to 32× depending on the model. However, Apple Silicon demonstrated surprising strengths in specific areas: MLX performed competitively—sometimes outperforming CUDA—on small matrix operations and consistently outpaced CUDA in convolutional workloads. These findings point to a likely optimization of Apple Silicon for many small, local operations rather than large batched matrix multiplications.

We theorize that these design decisions stem from Apple's focus on on-device intelligence, image processing, and mobile-first use cases, where low-latency convolutional performance and power efficiency are critical. By contrast, CUDA and NVIDIA's architecture are built for large-scale, parallel workloads in datacenter training environments. Together, these results underscore the importance of architectural context in ML performance and offer insight into how and why Apple may be tuning its silicon differently for emerging AI use cases.

Acknowledgments

Professor Felix Lin, for mentoring and advising me throughout this project.

References

[1] 2023. Apple introduces M2 Ultra. https://www.apple.com/newsroom/2023/06/apple-introduces-m2-ultra/
[2] 2025. NVIDIA Blackwell Architecture Technical Brief.
[3] Dahua Feng, Zhiming Xu, Rongxiang Wang, and Felix Xiaozhu Lin. 2025. Profiling Apple Silicon Performance for ML Training. doi:10.48550/arXiv.2501.14925 arXiv:2501.14925 [cs].
[4] hoakley. 2024. Apple silicon: 5 Memory and internal storage. https://eclecticlight.co/2024/03/06/apple-silicon-memory-and-internal-storage/
[5] Paul Hübner, Andong Hu, Ivy Peng, and Stefano Markidis. 2025. Apple vs. Oranges: Evaluating the Apple Silicon M-Series SoCs for HPC Performance and Efficiency. doi:10.48550/arXiv.2502.05317 arXiv:2502.05317 [cs] version: 1.
[6] Apple Inc. [n. d.]. Explore the new system architecture of Apple silicon Macs - WWDC20 - Videos. https://developer.apple.com/videos/play/wwdc2020/10686/