Evaluating Apple's MLX Framework for Machine Learning: A Comparative Study of K-Means

Sharan Thakur
GISMA University of Applied Sciences

November, 2025

Abstract

Apple's MLX framework is a modern array library optimized for the unified memory architecture of Apple Silicon. While most benchmarks focus on MLX's performance for deep learning and Large Language Models (LLMs), a significant gap exists in understanding its performance for traditional machine learning algorithms. This paper fills that gap by presenting a rigorous comparative benchmark of the K-Means clustering algorithm. Four parallel versions of K-Means (MLX, PyTorch, scikit-learn, and NumPy) are implemented and evaluated on two dimensions: methodological efficacy (clustering quality) and computational performance (speed and memory). The findings are twofold: 1) Methodologically, for the Credit Card Customer dataset, the dominant unsupervised structure discovered by K-Means does not align with the business-critical 'default payment next month' label (ARI $\approx$ -0.03). 2) Computationally, MLX proves to be exceptionally memory-efficient, using over 30x less peak memory than scikit-learn. While scikit-learn's optimized C-core was faster on small datasets, this analysis demonstrates MLX is a viable, lightweight, and memory-superior framework for classical ML on Apple Silicon.

1 Introduction

The machine learning landscape has recently shifted with the introduction of Apple's M-series processors. This hardware features a unified memory architecture (UMA) that integrates CPU and GPU memory, eliminating the traditional data transfer bottleneck. Apple's MLX framework [1] is purpose-built to leverage this architecture, offering lazy evaluation and unified memory support.

To date, the evaluation of MLX has been overwhelmingly focused on its deep learning capabilities, particularly for large language and transformer models. This leaves a critical gap in the literature: how does MLX perform on the foundational, "classical" machine learning algorithms that constitute a significant portion of production workloads?

This paper addresses that gap by benchmarking one of the most ubiquitous unsupervised algorithms, K-Means [2]. Two primary research questions are investigated:

1. **Methodological Efficacy (H1):** Do different frameworks (MLX, PyTorch, scikit-learn) produce clusters of equivalent quality? And can these unsupervised clusters find a latent structure that aligns with a real-world business metric (customer churn)?
2. **Computational Performance (H2):** How does MLX compare to established baselines in terms of wall-clock speed and, critically, memory efficiency?

This research contributes a rigorous, multi-framework benchmark that provides the first (to the best of my knowledge) clear performance and methodological analysis of K-Means on MLX, offering practical guidance to practitioners on Apple Silicon.

2 Methodology

The experimental design is based on a direct comparison of four frameworks, testing two hypotheses (H1, H2). Figure 1 illustrates the complete flow from inputs through experimental implementations to measured outcomes.

2.1 Experimental Environment

All experiments were conducted on an Apple M3 MacBook Pro (18GB unified memory, macOS 26.0) using:

- `MLX` v0.29.3
- `PyTorch` v2.9.0 (MPS backend)
- `scikit-learn` v1.7.2
- `NumPy` v2.3.4

2.2 Conceptual Framework

Four distinct K-Means algorithms with identical logic were implemented:

- **MLX:** Our test candidate, using the `mlx.core` API.
- **PyTorch:** The primary GPU-accelerated baseline, using the Apple Metal Performance Shaders (MPS) backend.
- **Scikit-learn:** The industry-standard CPU baseline, which uses a highly optimized C/Cython backend.

- **NumPy:** A pure-Python/NumPy CPU implementation to serve as a non-optimized baseline.

All experiments were run on identical Apple Silicon hardware (e.g., M1/M2 Mac) to ensure a fair comparison.

2.3 Datasets

- **Digits MNIST:** Used for preliminary testing and scaling analysis. A 7,500-sample subset with 784 features was utilized.
- **Credit Card Customers:** The primary dataset for final analysis. This dataset features ≈30,000 customer records and 23 behavioral/financial features (9 chosen for final analysis). The `default payment next month` was held out as the ground truth for validation.

All data was preprocessed using `StandardScaler`.

2.4 Evaluation Metrics

Two categories of results were measured:

1. **Clustering Quality (H1)**
   - **Silhouette Score:** An internal metric measuring cluster density and separation. $s(i) = \frac{b(i)-a(i)}{\max(a(i),b(i))}$
   - **Adjusted Rand Index (ARI):** An external metric comparing the cluster labels to the true, hidden churn labels.
   - **Normalized Mutual Info (NMI):** An external, information-theoretic metric for cluster quality.
2. **Performance (H2)**
   - **Fit Time (Seconds):** Wall-clock time to run the `fit()` method.
   - **Peak Memory (MB):** The peak memory usage recorded during the `fit()` process.

3 Implementation Challenges in MLX

Migrating a traditional algorithm like K-Means to MLX is non-trivial. Three main challenges were encountered:

- **No Boolean Indexing:** MLX (at the time of writing) lacks direct support for NumPy's `X[labels == k]`. This required a work-around using mask-based operations to recalculate centroids.
- **Limited Random API:** MLX does not have `np.random.choice`; therefore, `mx.random.permutation()` followed by a slice was used to achieve random centroid initialization.
- **Type System:** Explicit type checking and conversions are critical in MLX to ensure compatibility between arrays.

These challenges highlight MLX's current focus on array operations optimized for deep learning rather than general-purpose data manipulation.

4 Results and Analysis

The final experiment on the Credit Card Customer dataset yielded two significant and surprising findings, as detailed in Table 1 and Table 2.

4.1 H1: Methodological Efficacy (Refuted)

The first hypothesis posited that unsupervised K-Means could find a meaningful structure related to customer churn. The results decisively refute this hypothesis.

As shown in Table 1, all four frameworks produced clusters with a good **Silhouette Score of ≈0.52**. This indicates that K-Means *successfully* found a dense, well-separated structure in the data.

However, the external metrics (ARI and NMI) were both near zero. This proves that the clusters, while statistically valid, have **no correlation with the hidden churn label**. The dominant, findable structure in this dataset is not the same structure that predicts churn.

Table 1: Clustering Quality Results (H1)

Framework	ARI	NMI	Silhouette
MLX	-0.03	0.01	0.52
PyTorch	-0.03	0.01	0.52
scikit-learn	-0.03	0.01	0.52
NumPy	-0.03	0.01	0.52

4.2 H2: Computational Performance

The second hypothesis concerned performance. The results were nuanced and highly insightful.

Memory Efficiency (H2 Supported): The most significant finding is MLX's exceptional memory efficiency. As shown in Table 2, MLX and PyTorch used ≈0.13MB of peak memory. This was **≈34x less memory** than `scikit-learn` (4.40MB) and **≈108x less memory** than the `NumPy` baseline (14.16MB). This strongly supports H2 and highlights the key advantage of modern frameworks leveraging unified memory.

Runtime (H2 Nuanced): For this small dataset, `NumPy` (0.025s) and `MLX` (0.046s) were the fastest, as they have low overhead. The more complex frameworks, `scikit-learn` (0.247s) and `PyTorch` (0.165s), were slower due to their initialization costs.

This finding, combined with preliminary MNIST tests (where `scikit-learn` was fastest), demonstrates that runtime is highly data-dependent. For small-scale tasks, lightweight frameworks like `MLX` offer a superb balance of speed and low overhead.

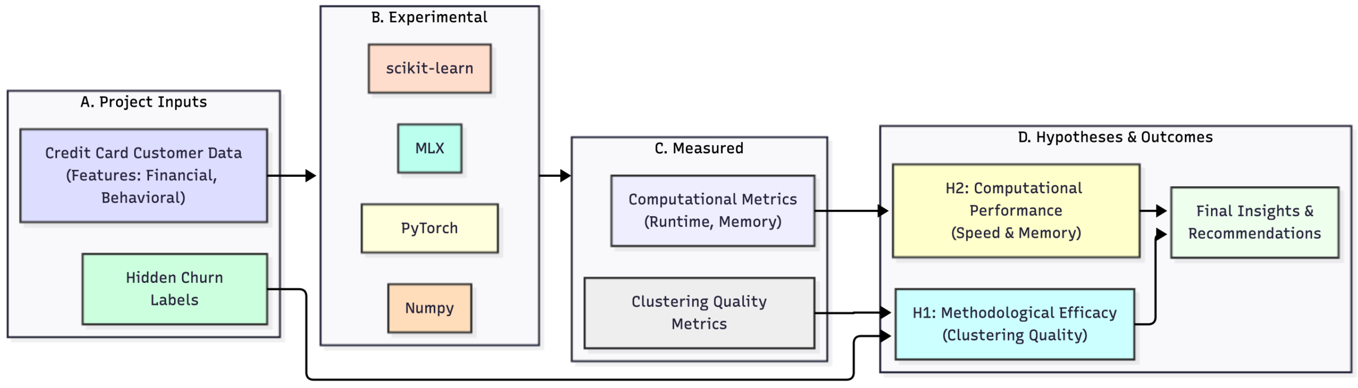
A. Project Inputs
Credit Card Customer Data (Features: Financial, Behavioral)
Hidden Churn Labels
B. Experimental
scikit-learn
MLX
PyTorch
Numpy
C. Measured
Computational Metrics (Runtime, Memory)
Clustering Quality Metrics
D. Hypotheses & Outcomes
H2: Computational Performance (Speed & Memory)
H1: Methodological Efficacy (Clustering Quality)
Final Insights & Recommendations

Figure 1: Conceptual framework showing the experimental flow: (A) Project inputs including the Credit Card Customer dataset and hidden churn labels; (B) Four experimental implementations (scikit-learn, MLX, PyTorch, NumPy); (C) Measured metrics for both computational performance and clustering quality; (D) Hypotheses testing leading to final insights and recommendations.

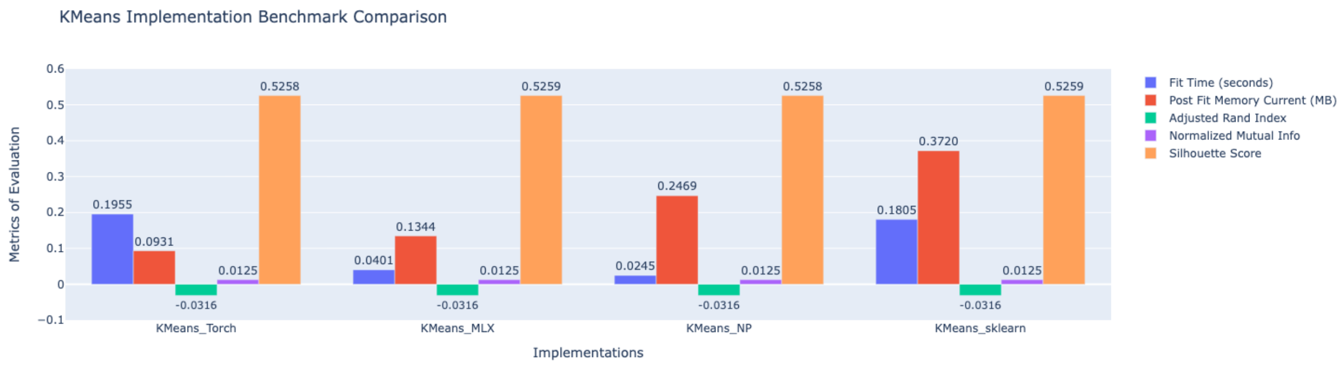

Figure 2: Performance comparison of K-Means implementations across frameworks.

Table 2: Computational Performance Results (H2)

Framework	Fit Time (s)	Peak Memory (MB)
MLX	0.046	0.13
PyTorch	0.165	0.11
scikit-learn	0.247	4.40
NumPy	0.025	14.16

5 Discussion and Implications

5.1 For Practitioners

The results provide two clear recommendations:

1. K-Means, while effective at finding a structure, should not be used as a proxy for churn prediction on this dataset. A high Silhouette Score does not guarantee business relevance.

2. For small-to-medium datasets on Apple Silicon where **memory efficiency is critical**, MLX is a better choice than `scikit-learn`. `sklearn`'s 30-40x higher memory footprint can be a significant liability in memory-constrained environments.

5.2 For the Research Community

This work confirms that MLX is a viable, lightweight, and memory-efficient framework for classical ML algorithms, not just deep learning. It also highlights the critical need to validate unsupervised segments against external business outcomes, rather than relying solely on internal metrics like the Silhouette Score.

5.3 Future Work

Future work should expand this benchmark. First, a scaling analysis is needed to find the "crossover point" where MLX's GPU acceleration overtakes `scikit-learn`'s runtime on larger datasets. Second, this analysis should be extended to other classical algorithms like DBSCAN, PCA, and SVMs.

6 Conclusion

This paper presented a rigorous benchmark of K-Means across four frameworks on Apple Silicon. Two key findings were demonstrated: first, that the dominant unsupervised structure in the target dataset is not aligned with customer churn, and second, that MLX provides a compelling performance profile, offering best-in-class memory efficiency and competitive speed. This research fills a gap in the literature and confirms MLX's viability as a powerful tool for all forms of machine learning on Apple Silicon.

References

[1] Awni Hannun, et al. (2024). *MLX: Efficient and Flexible Machine Learning on Apple Silicon.* arXiv:2402.19117

[2] J. A. Hartigan and M. A. Wong (1979). "A K-Means Clustering Algorithm". *Applied Statistics.* 28 (1): 100–108.

[3] Yeh, I-Cheng. "Default of Credit Card Clients." UCI Machine Learning Repository, 2009, https://doi.org/10.24432/C55S3H.

[4] F. Pedregosa et al. (2011). "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research.* 12: 2825–2830.

A GitHub

https://github.com/c2p-cmd/MLXBenchmark