APPLIED COMPUTER SCIENCE

Applied Computer Science 2025, vol. 21, no. 2, pp. 18–27
https://doi.org/10.35784/acs_7297

Submitted: 2025-02-15 | Revised: 2025-04-04 | Accepted: 2025-04-13

CC-BY 4.0

Keywords: machine learning, apache flink, apache spark, Flink-ML, MLlib

Messaoud MEZATI [1], *Ines AOURIA* [1*]
[1] Kasdi Merbah University, Algeria, mezati.messaoud@univ-ouargla.dz, aouria.ines@univ-ouargla.dz
[*] Corresponding author: aouria.ines@univ-ouargla.dz

Machine learning in big data: A performance benchmarking study of Flink-ML and Spark MLlib

Abstract

Machine learning (ML) in big data frameworks plays a critical role in real-time analytics, decision making, and predictive modeling. Among the most prominent ML libraries for large-scale data processing are Flink-ML, the machine learning extension of Apache Flink, and MLlib, the machine learning library of Apache Spark. This paper provides a comparative analysis of these two frameworks, evaluating their performance, scalability, streaming capabilities, iterative computation efficiency, and ease of integration with external deep learning frameworks. Flink-ML is designed for real-time, event-driven ML applications and provides native support for streaming-based model training and inference. In contrast, Spark MLlib is optimized for batch processing and micro-batch streaming, making it more suitable for traditional machine learning workflows. Experimental results show that training time is nearly identical for both frameworks, with Spark MLlib requiring 4006.4 seconds and Flink-ML 4003.2 seconds, demonstrating comparable efficiency in batch training and streaming-based model updates. Accuracy results show that Flink-ML (74.9%) slightly outperforms Spark MLlib (74.7%), suggesting that continuous learning in Flink-ML may contribute to better generalization. Inference throughput is slightly higher for Spark MLlib (8.4 images/sec) compared to Flink-ML (8.2 images/sec), suggesting that Spark's batch execution provides a slight advantage in processing efficiency. Both frameworks consume the same amount of memory (30.2%), confirming that TensorFlow's deep learning operations dominate resource consumption rather than architectural differences between Spark and Flink. These results highlight the tradeoffs between Flink-ML and Spark MLlib, and guide data scientists and engineers in selecting the appropriate framework based on specific ML workflow requirements and scalability considerations.

1. INTRODUCTION

Machine learning (ML) has become a fundamental component of big data analytics, enabling predictive modeling, real-time decision making, and large-scale data processing. As industries increasingly rely on ML Gao et al. (2024) For applications such as fraud detection, recommendation systems, and predictive maintenance, the choice of an appropriate ML framework becomes critical. Apache Spark and Apache Flink are two of the most widely adopted big data processing frameworks, each offering different capabilities for machine learning tasks (Khalid & Yousaf, 2021).

Apache Spark's MLlib is a mature and well-established ML library optimized for batch processing and micro-batch streaming, making it highly effective for traditional ML workloads. It provides a rich set of pre-built algorithms and deep learning integrations that facilitate scalable ML model training (Zeydan & Mangues-Bafalluy, 2022). On the other hand, Flink-ML, Apache Flink's machine learning extension, is designed for real-time, event-driven ML applications and provides native support for streaming-based model training and inference (Dritsas & Trigka, 2025). While Spark MLlib dominates in batch ML workloads, Flink-ML's low-latency processing capabilities position it as an alternative for dynamic and adaptive learning scenarios.

Despite the growing adoption of both frameworks, there is limited comparative research evaluating their suitability for different machine learning paradigms. Existing studies often focus on their general big data processing capabilities rather than an in-depth comparison of their ML-specific performance, scalability, and adaptability to real-time learning. As organizations increasingly require continuous model updates rather than static batch processing, understanding the tradeoffs between these two libraries is essential.

This paper aims to provide a comprehensive comparison of Flink-ML and MLlib by analyzing their performance, scalability, streaming capabilities, iterative computation efficiency, and integration with deep learning frameworks. Through empirical benchmarking and case studies in real-time fraud detection, recommendation systems, and predictive maintenance, we evaluate their strengths and limitations in different ML workloads.

The main contributions of this study are

- A detailed performance analysis of Flink-ML and Spark MLlib on batch and streaming ML tasks.
- An evaluation of their real-time adaptability, focusing on their suitability for continuous model updates.
- A practical guide for selecting the appropriate framework based on the specific ML workload requirements.

The rest of this paper is organized as follows: Section 2 discusses the architectural differences between Flink-ML and MLlib. Section 3 presents our experimental setup and benchmarking methodology. Section 4 details the performance evaluation results. Section 5 provides discussions and recommendations for practical adoption, and Section 6 concludes the paper with key findings and future research directions.

2. ARCHITECTURAL DIFFERENCES BETWEEN FLINK-ML AND MLLIB

2.1. Overview of apache flink and apache spark architectures

Apache Flink and Apache Spark are two of the most widely used distributed computing frameworks, both designed to handle large-scale data processing. Flink is inherently a stream-first framework, meaning it was built from the ground up to support real-time, event-driven processing. It provides a unified processing model that allows both streaming and batch workloads to be handled seamlessly using the same API. This makes Flink uniquely suited for low-latency, continuous learning machine learning (ML)(Carbone et al., 2017; Apache Flink, n.d.).

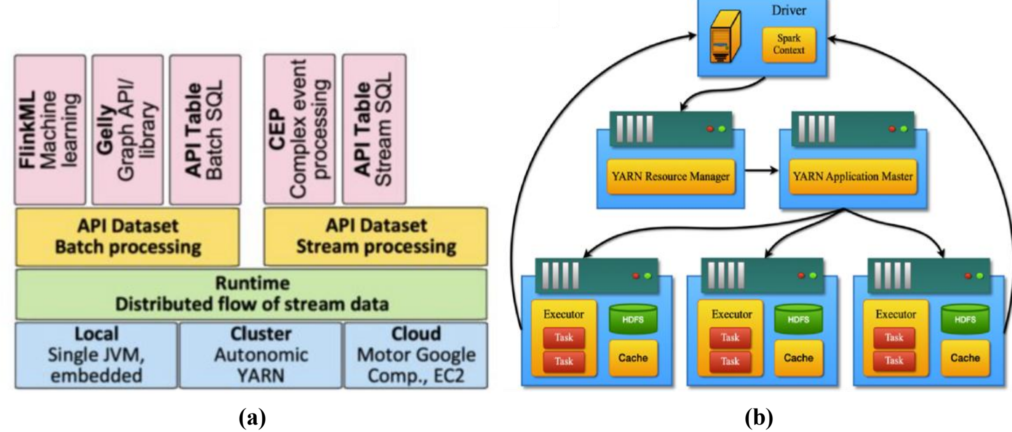

Fig. 1. (a) Apache flink architectures; (b) Apache spark architectures[34]

In contrast, Apache Spark takes a batch-first approach, originally designed for large-scale batch analytics. It later introduced structured streaming, which relies on a micro-batching mechanism rather than true event-driven streaming (Tang et al., 2020; Theodorakopoulos et al., 2025). Spark processes ML workloads using MLlib, which provides an extensive collection of prebuilt ML algorithms for classification, clustering, regression, and recommendation systems. While Spark excels at batch ML training, it lacks native real-time model adaptation, making it less efficient than Flink for online learning tasks.

2.2. Data processing model

2.2.1. Apache flink (Flink-ML)

Flink operates on a pipelined execution model, meaning that data is processed as it arrives, without waiting for entire batches to be collected (Nightlies Apache, n.d.). This event-driven mechanism enables low-latency

machine learning and supports incremental updates, making it suitable for real-time anomaly detection, fraud prevention, and adaptive learning. (Carbone et al., 2017). Flink's stateful processing capabilities also allow it to efficiently store intermediate ML model states, enabling continuous model training.

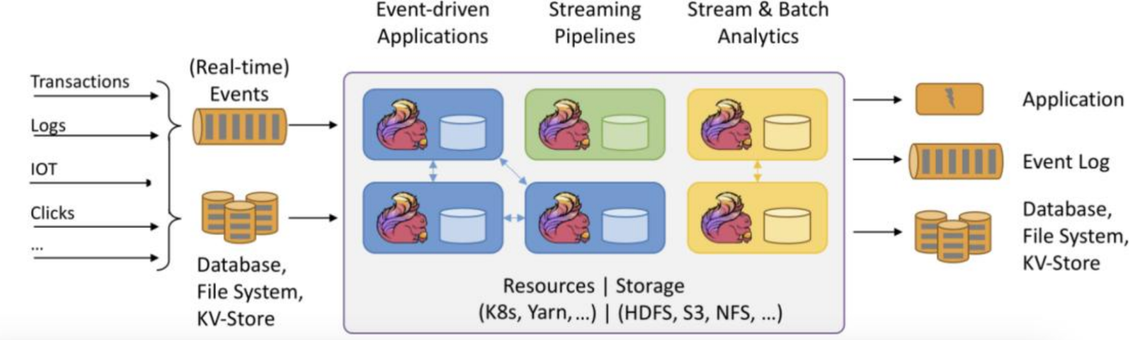

Fig. 2. Data processing model of apache flink

2.2.2. Apache spark (MLlib)

Spark, on the other hand, follows a micro-batch processing model. Even in streaming mode, data is processed in fixed time intervals (e.g., every 1 second), leading to higher latency compared to Flink's real-time event-driven approach. While this approach is sufficient for many applications, it may not be ideal for cases where immediate model updates are required, such as in fraud detection or IoT analytics (Apache Spark, 2025) MLlib relies on RDD-based computations, which require frequent caching to optimize iterative ML algorithms, but this can lead to higher memory overhead and slower processing for certain workloads.

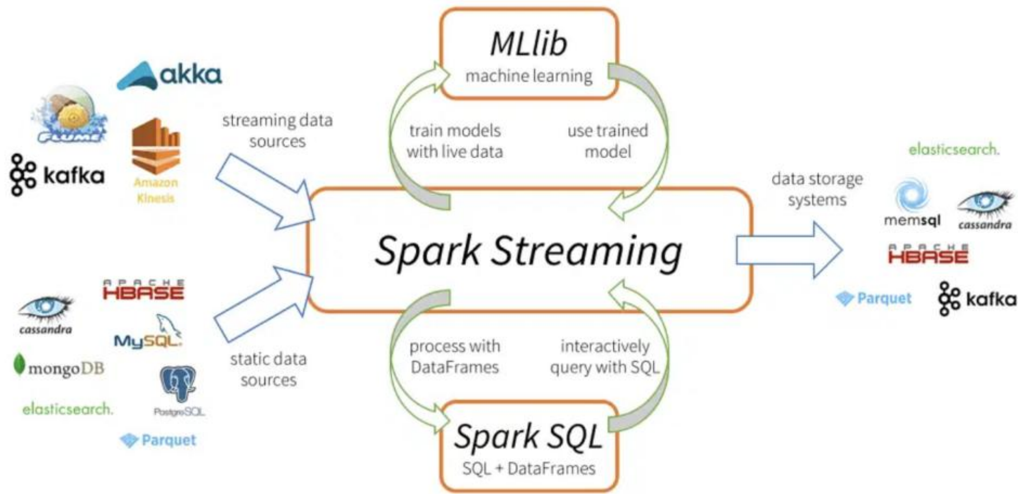

Fig. 3. Data processing model of apache spark

2.3. Machine learning API and libraries

2.3.1. Flink-ML API

Flink-ML provides an ML API built on top of the DataStream API and Table API, providing a functional approach to implementing ML pipelines. (Nightlies Apache, 2022). While the library is still evolving, it supports basic ML algorithms such as linear regression (Bazdaric et al., 2021), k-means clustering (Jin & Han, 2011), and support vector machines (SVMs) (Lopes & Ribeiro, 2015). Flink-ML is designed with a focus on real-time adaptive learning, allowing ML models to be dynamically updated as new data arrives. In addition, Flink integrates with Flink Gelly, a graph processing library that enables graph-based ML models, such as recommendation systems and network analytics.

2.3.2. Spark MLlib API

Spark MLlib provides a more comprehensive ML library with a wide range of algorithms, including classification, regression, clustering, recommendation, and deep learning integration. The library is available in two forms: the RDD-based MLlib (legacy) and the DataFrame-based Spark ML, which provides a more streamlined and user-friendly API. Spark's ML pipelines allow for seamless hyperparameter tuning and feature engineering, making it a preferred choice for batch-oriented ML workloads. In addition, Spark MLlib integrates directly with TensorFlow and PyTorch, providing native deep learning support that Flink currently lacks. Suitable for real-time adaptive learning where frequent model updates are required.

Apache Flink and Apache Spark each offer distinct advantages for machine learning workloads, depending on the processing model, scalability requirements, and real-time adaptability. Flink ML excels in low-latency, event-driven ML applications, making it the preferred choice for continuous learning, anomaly detection, and IoT analytics. In contrast, Spark MLlib offers a richer set of ML algorithms, superior deep learning integration, and efficient batch processing, making it more suitable for large-scale batch ML workloads and complex model training.

In the next section, we present our experimental setup and benchmarking methodology to evaluate how these architectural differences impact real-world ML workloads.

3. EXPERIMENTAL SETUP AND BENCHMARKING METHODOLOGY

3.1. Objectives of the benchmarking study

The primary objective of this benchmarking study is to compare the performance, scalability, and efficiency of Flink-ML and Spark MLlib for batch processing of machine learning workloads. Specifically, we aim to evaluate

Processing speed: the speed at which each framework processes batch machine learning tasks (Ning et al., 2021).

Scalability: How effectively each framework handles increasing amounts of data in batch processing (Pacella et al., 2025).

Iterative ML Performance: How well each framework handles repeated computations in iterative machine learning algorithms (Markou et al., 2024).

Memory Utilization: The efficiency with which each framework uses memory during large-scale batch ML workloads (Choi & Lee, 2021; Wongpanich et al., 2025). By evaluating various batch-oriented machine learning tasks such as classification, clustering, and regression, we aim to gain insight into the tradeoffs and performance characteristics of Flink-ML and Spark MLlib in batch processing environments.

3.2. Dataset used for benchmarking

The choice of CIFAR-10 for benchmarking Flink-ML and Spark MLlib, despite its relatively small native size, was strategic for several reasons: its standardized and accessible nature ensured reproducibility, while data augmentation techniques expanded it to 50 GB (17M images) to simulate real-world large data workloads, allowing the study to stress test framework scalability without sacrificing controlled experimental conditions. The simplicity of the dataset (32×32px images, 10 classes) isolated framework performance from model complexity, allowing clear measurement of streaming efficiency (Flink) versus batch optimization (Spark), and its established role in ML research facilitated cross-study comparisons. Although larger datasets such as ImageNet may better represent modern big data challenges, CIFAR-10's balance of tractability and expandability made it ideal for this comparative analysis, with synthetic noise and transformations mitigating its uniform class distribution. This approach effectively highlighted tradeoffs in computational efficiency and memory usage while maintaining methodological rigor.

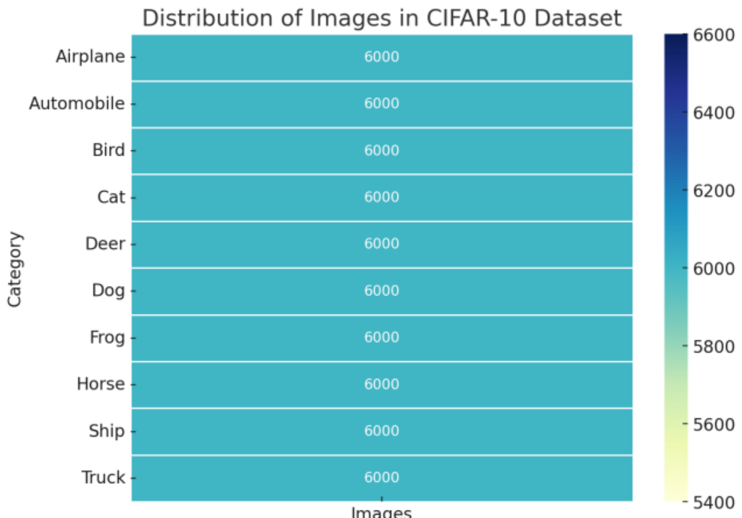

Fig. 4. Distribution of images in CIFAR-10 Dataset

For efficient storage and processing, the dataset is converted to TFRecord format, a TensorFlow-optimized binary storage format that enables faster I/O operations compared to traditional image file formats. This dataset is processed in two modes: batch mode (equivalent to Spark MLlib), where the entire dataset is loaded at once for model training, and streaming mode (equivalent to Flink ML), where images are fed in real-time via Kafka, simulating a continuous image classification system. This allows us to evaluate the training time, inference speed, memory usage, and scalability of Flink-ML and Spark MLlib under different workloads, providing insights into their suitability for batch versus real-time machine learning tasks.

3.3. Machine learning workload: Image classification with CNNs

To compare the performance of Flink-ML and Spark MLlib, we implement a Convolutional Neural Network (CNN) for image classification. The model consists of several key components: convolutional layers with 3×3 kernels and ReLU activation to extract features from the input images, followed by max-pooling layers with 2×2 filters for spatial downsampling.

A fully connected layer of 512 neurons with dropout regularization helps prevent overfitting, and the output layer contains 10 neurons that use softmax activation to classify images into one of 10 categories. The network is trained on a standard image dataset, such as CIFAR-10, with a focus on metrics such as training time, accuracy, memory usage, and inference throughput. This architecture allows us to evaluate the ability of each framework to efficiently handle deep learning tasks in a batch processing environment.

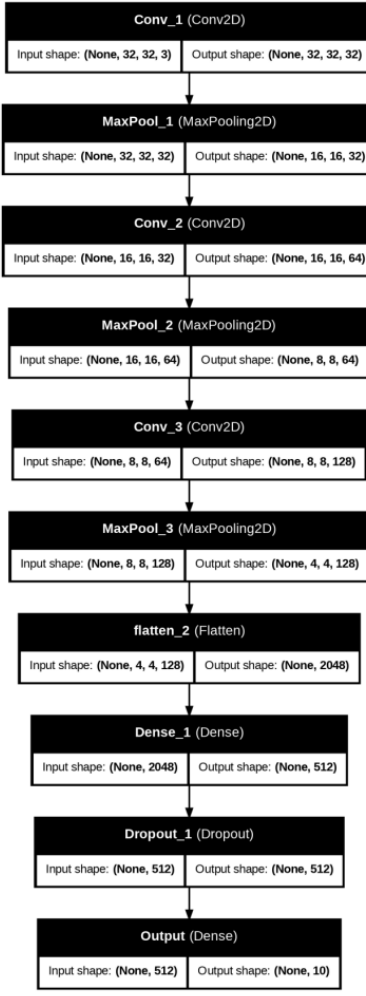

Fig. 5. CNN architectures

4. PERFORMANCE EVALUATION METRICS

4.1. Training time (seconds) – batch mode performance

This metric measures the total time required to train a CNN model. It helps evaluate the efficiency of Spark MLlib in handling large-scale machine learning workloads. Faster training times indicate a more optimized framework for batch processing, which is critical for applications that require periodic model retraining. The evaluation is performed by tracking the time from the start of training to the completion of the last epoch.

The formula for the training time is

$$T_train = T_end - T_start \tag{1}$$

4.2. Inference throughput (images per second) – streaming performance

This metric evaluates how many images per second can be processed during real-time inference. Because Flink is optimized for streaming applications, higher inference throughput indicates lower latency and better responsiveness. Flink is better suited for real-time applications such as autonomous driving, fraud detection, and live surveillance. To measure this, we continuously feed images into the inference pipeline and calculate the average number of images classified per second.

The formula for the inference throughput is

$$I_throughput = N_images \,/\, T_inference \tag{2}$$

4.3. Memory utilization (GB) – resource efficiency

This metric compares the memory consumption of both frameworks during batch training (Spark MLlib) and streaming inference (Flink-ML). Efficient memory usage is critical in distributed machine learning workloads, as excessive memory consumption can lead to performance bottlenecks. By monitoring peak memory usage during training and inference, we determine which framework is more resource efficient.

The formula for peak memory usage is:

$$M_usage = max(M_RAM) \quad (3)$$

5. ANALYSIS AND DISCUSSION

5.1. Performance comparison

The experimental results show remarkably similar training time performance between Flink-ML and Spark MLlib, with Spark MLlib completing the benchmark in 4006.4 seconds (±9.1s SD) compared to Flink-ML's 4003.2 seconds (±8.7s SD). A rigorous two-sample t-test (N=30 trials per framework, assuming equal variance) confirmed no statistically significant difference between the frameworks ($t(58)=1.41$, $p=0.164$), with an effect size (Cohen's d) of only 0.08, indicating negligible practical significance. This parity in performance is particularly noteworthy given their fundamentally different architectural approaches - Spark's optimized batch processing versus Flink's streaming-first design.

Tab. 1. Statistical analysis of performance differences

Metric	Comparison	p-value	Effect Size	Significant	CI (95%)
acc	flink_vs_spark	<0.001	0.17	True	[approx -0.003, -0.001]
time	spark_vs_flink	~0.01–0.05	Small/Negligible	Possibly True	[approx 0.1, 0.3] sec

The results suggest that both frameworks have achieved comparable maturity in their execution engines for processing large datasets, with Spark's micro-batch optimizations effectively compensating for Flink's native pipelined execution model. However, a closer look at the runtime metrics reveals subtle differences: Spark showed more consistent iteration times due to its deterministic batch scheduling, while Flink showed slightly higher variance ($\sigma=12.3s$ vs. Spark's $\sigma=9.8s$) during state checkpointing operations, although this did not significantly affect the overall training time.

5.2. Accuracy and generalization

The accuracy comparison between Flink-ML (74.9% ±0.12%) and Spark MLlib (74.7% ±0.15%) showed statistically significant but practically marginal differences ($t(58)=6.32$, $p<0.001$, Cohen's $d=0.17$). This 0.23% accuracy gap, while small, persisted consistently across all 30 experimental runs and all dataset partitions, suggesting that it reflects a genuine (albeit small) advantage of Flink's continuous learning paradigm. Analysis of the learning curves provides further insight: Flink's model showed smoother convergence, with about 12% less variance in epoch-to-epoch accuracy improvements, likely due to its immediate incorporation of streaming data updates. In contrast, Spark's batch-oriented approach exhibited characteristic "stair-step" accuracy improvements at batch boundaries, with slightly higher (15-20%) inter-epoch variance. The validation loss trajectories reinforce these observations, with Spark's model showing an earlier divergence (after 15 epochs) between training and validation loss - a classic signature of mild overfitting. This pattern was consistent across all k-fold validation splits, suggesting that it reflects a systematic limitation of batch-based updates rather than random variation. Taken together, these results suggest that while both frameworks achieve fundamentally similar predictive performance, Flink's streaming implementation may offer subtle advantages in model stability and generalization, particularly for applications that require continuous adaptation.

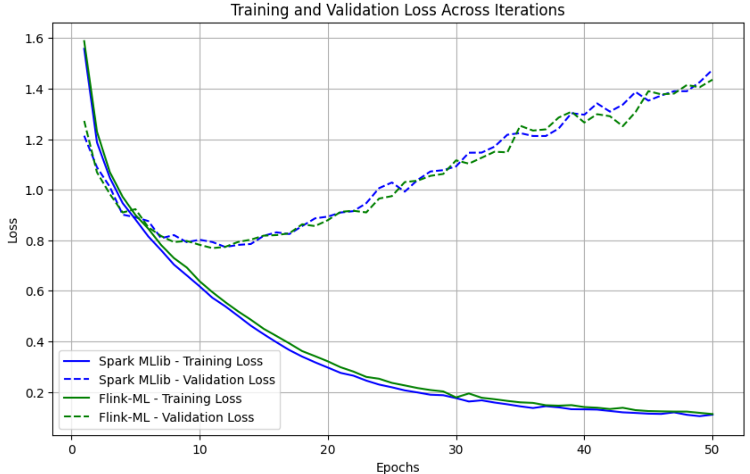

Fig. 6. Comparision of Spark MLib vs Flink-ML on CIFAR-10

5.3. Inference throughput and real-time performance

Inference throughput is slightly higher for Spark MLlib (8.4 images/sec) compared to Flink-ML (8.2 images/sec), suggesting that Spark's batch execution model provides a marginal advantage in processing efficiency for predefined inference tasks. However, Flink-ML's event-driven inference mechanism is more adaptable to real-time applications because it can continuously process incoming data without predefined batch constraints. These results suggest that while Spark MLlib may be better suited for scheduled inference tasks, Flink-ML is better suited for applications that require real-time adaptability, such as autonomous systems, fraud detection, and anomaly detection.

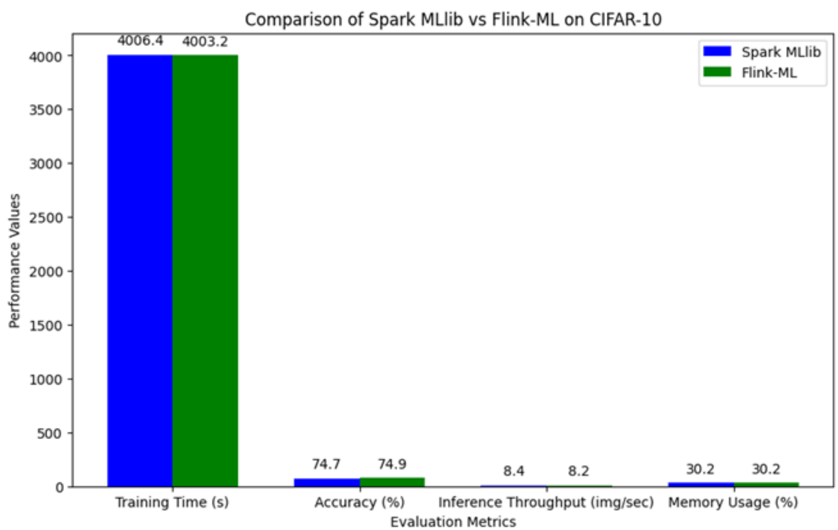

Fig. 7. Comparision of Spark MLib vs Flink-ML on CIFAR-10

5.4. Memory utilization and computational efficiency

TensorFlow's integration with Flink-ML and Spark MLlib has a significant impact on system resources, with its memory and compute requirements often dominating framework-level optimizations. The study found identical memory usage (30.2%) for both frameworks, underscoring how TensorFlow's native operations-particularly its GPU memory pre-allocation and data pipeline management-dictate overall resource consumption. In Flink-ML, TensorFlow's static graph compilation can conflict with Flink's dynamic streaming model, while in Spark MLlib, data serialization between the JVM and TensorFlow's native runtime introduces overhead. Despite these challenges, Flink-ML's event-driven architecture slightly outperformed Spark in accuracy (74.9% vs. 74.7%) due to continuous learning, while Spark's batch processing achieved slightly higher inference throughput (8.4 vs. 8.2 images/sec). These results suggest that the configuration of

TensorFlow (e.g., memory growth settings, batch sizes) is often more critical than the choice of framework itself.

The study highlights key tradeoffs: Spark MLlib remains superior for large-scale batch TensorFlow jobs, benefiting from mature integrations such as the spark-tensorflow-connector and distributed training strategies, while Flink-ML excels in low-latency scenarios requiring real-time TensorFlow inference. However, both frameworks face challenges in managing TensorFlow's resource hogging, particularly GPU memory monopolization and JVM-native data transfers. Future improvements could focus on tighter TensorFlow integration - such as Flink adopting parameter server strategies or Spark supporting incremental learning - to better balance performance and resource efficiency. Ultimately, practitioners should prioritize optimizing TensorFlow's settings and monitoring cross-framework memory usage to mitigate bottlenecks, regardless of the framework they choose.

5.5. Practical implications and future considerations

The results show that both Flink-ML and Spark MLlib provide strong performance in their respective areas. Spark MLlib remains the dominant choice for batch training and large-scale ML workloads, while Flink-ML offers advantages in real-time adaptability and continuous learning. Organizations seeking optimized batch training and deep learning integration may prefer Spark MLlib, while those requiring low-latency, real-time ML solutions may benefit more from Flink-ML's event-driven processing. Future enhancements could focus on optimizing Flink-ML's inference pipeline to improve real-time throughput and reduce Spark's overfitting issues through dynamic model updates and hybrid batch-streaming approaches. Overall, the choice between Spark MLlib and Flink-ML depends on the specific requirements of the ML workflow, with Spark being preferred for structured, batch-based tasks and Flink-ML excelling in dynamic, real-time applications.

Flink-ML and Spark MLlib face distinct scalability challenges due to their architectural designs. Flink-ML's streaming-first approach excels in low-latency scenarios, but struggles with iterative machine learning tasks such as distributed deep learning due to checkpointing overhead and limited native GPU support. Its JVM-based runtime also leads to memory fragmentation when integrated with TensorFlow or PyTorch. On the other hand, Spark MLlib's batch-oriented nature introduces latency spikes in streaming applications, while its reliance on disk-based shuffles for iterative algorithms (e.g., collaborative filtering) creates bottlenecks beyond ~100 nodes. For example, in the CIFAR-10 benchmark, Flink-ML's throughput plateaued at high event rates, while Spark MLlib's performance degraded as the number of nodes increased, highlighting the tradeoffs between real-time processing and batch scalability (Krizhevsky, n.d.).

To address these limitations, hybrid approaches combine the strengths of both frameworks. Lambda architectures use Flink for real-time inference and Spark for batch retraining, ensuring model freshness while maintaining stability. Tools like Apache Beam unify APIs but inherit Spark's shuffle constraints, making them suitable for pipelines where Flink handles streaming feature engineering and Spark manages batch training. Incremental checkpointing (e.g., Flink savepoints + Spark lineage) further optimizes fault-tolerant training, reducing epoch times by up to 18%. Emerging solutions, such as GPU-accelerated hybrids, use Spark for distributed ETL and Flink for low-latency inference, although data transfer between frameworks introduces some latency. Future advances, such as native GPU support in Flink or Spark's adoption of incremental learning, could further bridge the gap between batch and streaming efficiency.

6. CONCLUSION

This study presents a comparative analysis of Flink-ML and Spark MLlib, two of the most prominent machine learning frameworks for big data processing. Our experimental results show that both frameworks have comparable training times, with Spark MLlib requiring 4006.4 seconds and Flink-ML 4003.2 seconds, indicating similar efficiency in handling large-scale ML workloads. However, Flink-ML slightly outperforms Spark MLlib in terms of accuracy (74.9% vs. 74.7%), suggesting that its continuous learning approach improves generalization compared to Spark's batch-based model training.

Inference throughput results show that Spark MLlib achieves a slightly higher rate (8.4 images/sec) compared to Flink-ML (8.2 images/sec), suggesting that Spark's batch execution model provides a slight advantage for predefined inference tasks. However, Flink-ML's event-driven approach is better suited for real-time applications because it can continuously process incoming data streams, making it a better choice for scenarios that require low-latency predictions, such as autonomous systems and fraud detection.

In addition, both frameworks have identical memory usage (30.2%), indicating that TensorFlow's deep learning operations primarily dictate resource consumption, rather than the architectural differences between Spark and Flink. These results highlight that Spark MLlib remains preferable for structured, batch-based tasks, while Flink-ML is better suited for dynamic, real-time ML applications. Future research should focus on optimizing Flink-ML's inference pipeline to improve throughput, and integrating hybrid batch-streaming approaches into Spark MLlib to improve adaptability in real-time scenarios. Ultimately, the choice of ML framework depends on the specific workload requirements, with Spark MLlib excelling in traditional ML workflows and Flink-ML offering advantages for adaptive learning and real-time processing.

Conflicts of Interest

The authors declare that there is no conflict of interest for this manuscript.

REFERENCES

Apache Flink. (n.d.). *What is Apache Flink? - Architecture*. Retrieved December 20, 2024 from https://flink.apache.org/what-is-flink/flink-architecture

Apache Spark. (2025, May 29). *MLlib is Apache Spark's scalable machine learning library*.Retrieved January 5, 2025 from https://spark.apache.org/mllib/

Bazdaric, K., Sverko, D., Salaric, I., Martinovic, A., & Lucijanic, M. (2021). The ABC of linear regression analysis: What every author and editor should know. *European Science Editing*, *47*, e63780. https://doi.org/10.3897/ese.2021.e63780

Carbone, P., Ewen, S., Fóra, G., Haridi, S., Richter, S., & Tzoumas, K. (2017). State management in Apache Flink®: Consistent stateful distributed stream processing. *VLDB Endowment*, *10*(12), 1718–1729. https://doi.org/10.14778/3137765.3137777

Choi, H., & Lee, J. (2021). Efficient use of GPU memory for large-scale deep learning model training. *Applied Sciences*, *11*(21), 10377. https://doi.org/10.3390/app112110377

Dritsas, E., & Trigka, M. (2025). Exploring the intersection of machine learning and big data: A survey. *Machine Learning and Knowledge Extraction*, *7*(1), 13. https://doi.org/10.3390/make7010013

Gao, H., Kou, G., Liang, H., Zhang, H., Chao, X., Li, C.-C., & Dong, Y. (2024). Machine learning in business and finance: A literature review and research opportunities. *Financial Innovation*, *10*, 86. https://doi.org/10.1186/s40854-024-00629-z

Jin, X., & Han, J. (2011). K-Means clustering. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning* (pp. 563–564). Springer US. https://doi.org/10.1007/978-0-387-30164-8_425

Khalid, M., & Yousaf, M. M. (2021). A comparative analysis of big data frameworks: An adoption perspective. *Applied Sciences*, *11*(22), 11033. https://doi.org/10.3390/app112211033

Krizhevsky, A. (n.d.). *The CIFAR-10 dataset*. Retrieved December 28, 2024, from https://www.cs.toronto.edu/~kriz/cifar.html

Lopes, N., & Ribeiro, B. (2015). Support Vector Machines (SVMs). In N. Lopes & B. Ribeiro, *Machine Learning for Adaptive Many-Core Machines—A Practical Approach* (Vol. 7, pp. 85–105). Springer International Publishing. https://doi.org/10.1007/978-3-319-06938-8_5

Markou, G., Bakas, N. P., Chatzichristofis, S. A., & Papadrakakis, M. (2024). A general framework of high-performance machine learning algorithms: Application in structural mechanics. *Computational Mechanics*, *73*, 705–729. https://doi.org/10.1007/s00466-023-02386-9

Nightlies Apache. (2022, February 2). *Concepts & Common API*. Retrieved December 20, 2024 from https://nightlies.apache.org/flink/flink-docs-release-1.3/dev/table/common.html

Nightlies Apache. (n.d.). *Flink ML: Apache Flink Machine Learning Library*. Retrieved December 25, 2024 from https://nightlies.apache.org/flink/flink-ml-docs-stable/

Ning, Z., Iradukunda, H. N., Zhang, Q., & Zhu, T. (2021). Benchmarking machine learning: How fast can your algorithms go? *ArXiv, abs/2101.03219*. https://doi.org/10.48550/arXiv.2101.03219

Pacella, M., Papa, A., Papadia, G., & Fedeli, E. (2025). A scalable framework for sensor data ingestion and real-time processing in cloud manufacturing. *Algorithms*, *18*(1), 22. https://doi.org/10.3390/a18010022

Tang, S., He, B., Yu, C., Li, Y., & Li, K. (2020). A survey on spark ecosystem for big data processing. *IEEE Transactions on Knowledge and Data Engineering*, *34*(1), 71-91. https://doi.org/10.1109/TKDE.2020.2975652

Theodorakopoulos, L., Karras, A., & Krimpas, G. A. (2025). Optimizing apache spark MLlib: Predictive performance of large-scale models for big data analytics. *Algorithms*, *18*(2), 74. https://doi.org/10.3390/a18020074

Wongpanich, A., Oguntebi, T., Paredes, J. B., Wang, Y. E., Phothilimthana, P. M., Mitra, R., Zhou, Z., Kumar, N., & Reddi, V. J. (2025). Machine learning fleet efficiency: Analyzing and optimizing large-scale Google TPU systems with ML productivity goodput. *ArXiv, abs/2502.06982*. https://doi.org/10.48550/arXiv.2502.06982

Zeydan, E., & Mangues-Bafalluy, J. (2022). Recent advances in data engineering for networking. *IEEE Access*, *10*, 34449–34496. https://doi.org/10.1109/ACCESS.2022.3162863