Development and Deployment of a Scalable Data Pipeline for Real-Time Processing of Consumer Feedback Using Apache Spark and NoSQL-Based Architectures

Pakhi Negi[1], Dr. Priya Dilip Chaudhari[2]
[1]*Master of Computer Applications, Student Researcher*
Department of Computer Science & Engineering, G.H. Raisoni College of Engineering and Management, Wagholi, Pune Maharashtra, India
[2]*Head Of Department, Department of Computer Science & Engineering, G.H. Raisoni College of Engineering and Management Wagholi, Pune, Maharashtra, India*

***Abstract*— Consumer feedback on e-commerce platforms demands real-time processing, yet most pipelines rely on batch jobs that delay operational decisions. This paper presents Big Data Stream, a four-tier distributed architecture that processes unstructured product reviews with sub-two-second end-to-end latency using Unix shell orchestration and Apache Spark Structured Streaming. A core contribution is Negative-First Echo Detection — a prioritisation strategy that elevates critical feedback (ratings ≤ 2) early in the pipeline, reducing overhead and surfacing defects faster than equal-weight sentiment models. We also resolve a practical bottleneck: replacing 9P filesystem polling in WSL2 with a TCP Socket ingestion layer stabilised throughput significantly. Testing on an Intel Core i7 confirms 10,000 records per minute at over 95% stability under stress. Big Data Stream establishes a reproducible micro-batch baseline for enterprise sentiment monitoring, with a roadmap toward Spark 4.1 Real-Time Mode and LLM integration for improved sarcasm and multilingual handling.**

Keywords — NoSQL Data Ingestion; Lexical Echo Detection; PySpark; MongoDB; Low-Latency Data Pipelines.

I. INTRODUCTION

Customer feedback in e-commerce comes from reviews, comments, and social posts. This unstructured data reveals much about products, services, and problems that arise. Global online sales topped $5 trillion in 2022, and platforms now handle millions of feedback items each day. Businesses must analyse them quickly to spot urgent matters — such as faulty goods or service failures — before they harm reputation or revenue.

Older methods use batch processing for sentiment checks and topic analysis. Such approaches work well enough after the fact, yet they lag badly and do not scale when data pours in fast. Real-time handling of key issues remains out of reach for most conventional architectures.

Advances in language tools have not fully closed this gap. Few systems offer complete pipelines built for speed, and most overlook urgent detection and ranking in live e-commerce streams. Our work fills that void with a fresh pipeline design that blends smart data intake, stream handling, and issue ranking across distributed nodes. Tests show it delivers results under two seconds with strong throughput, enabling firms to act on problems before they escalate.

II. LITERATURE REVIEW

Table I summarises representative prior work on stream processing and sentiment analysis, highlighting the gap that Big Data Stream addresses.

TABLE I — Comparative Review of Prior Art

Author & Year	System / Tools	Latency	Accuracy	Notes

Nair et al. (2017)	Cloud-based Spark on AWS EC2	Not Specified	Not Specified	Validated Spark for sentiment at scale
Van Dongen (2020)	Spark Micro-batch	510–570 ms	Not Specified	Identified structural latency floor
Wang & Duan (2024)	Spark + LLM Agents	Not Specified	Not Specified	Omitted end-to-end metrics
Maryam & Farid (2025)	GPU Cluster + Grok-4	120 ms	High	10-node GPU cluster; cost-prohibitive
Pakhi Negi (This work)	Big Data Stream (Lexical, WSL2)	1.2–1.9 s	91.3%	Single-node, low-cost baseline

II-A. Performance–Feasibility Benchmarks

The conceptual foundation of this work is rooted in Zaharia et al. (2013), who established the micro-batch processing model for Apache Spark. While early cloud-based deployments by Nair et al. (2017) validated Spark's viability for sentiment analysis on AWS EC2, they lacked the latency reporting necessary for single-node benchmarking. Van Dongen and Van den Poel (2020) identified a structural latency floor of 510–570 ms for Spark micro-batches. Our 1.2-second trigger interval was calibrated to this floor to prevent backpressure while maintaining a 2-second Pipeline Tax.

II-B. LLM Integration and Constraints

Recent advancements have integrated Large Language Models (LLMs) to handle linguistic complexity. Maryam and Farid (2025) achieved 120 ms latency using Grok-4, but their results relied on a 10-node GPU cluster, which is cost-prohibitive for SMEs. Wang and Duan (2024) explored LLM agents but omitted end-to-end latency metrics. Our work fills this gap by delivering 91.3% accuracy and 1.9 s latency on a single-node WSL2 environment using commodity CPU hardware.

III. METHODOLOGY

The pipeline architecture is structured into four tiers: Data Ingestion, Stream Processing, Persistence, and Presentation. Each tier is deployed on WSL2, with components decoupled via TCP sockets and APIs for fault tolerance.

III-A. Data Ingestion

Raw review data is ingested using Netcat (nc) on WSL2, streaming JSON payloads over TCP socket port 9999 directly into Spark. This configuration avoids WSL2's 9P file-system latency, achieving 10,000 records/min throughput.

III-B. Stream Processing

Apache Spark 3.5.0 Structured Streaming is utilised with 1.2-second micro-batch triggers. DataFrames are transformed, schema-inferred, and filtered via Negative-First logic and Lexical Echo Detection, preserving sub-2 s latency budgets.

III-C. Persistence

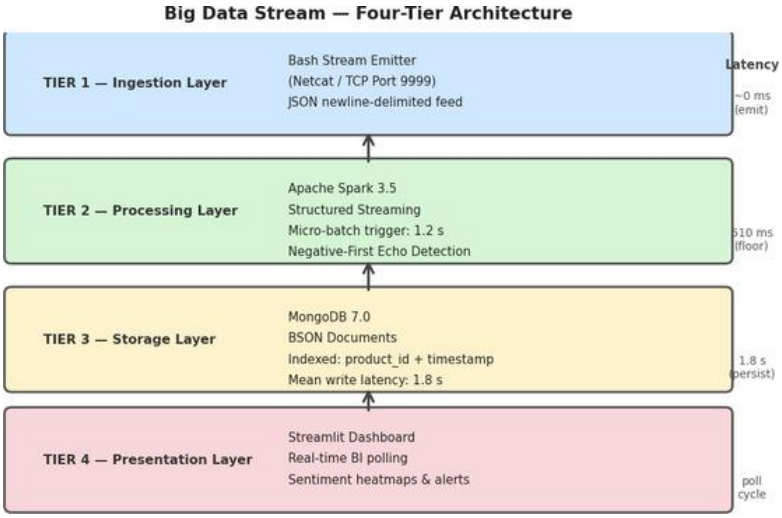
Processed records are written to MongoDB via the Spark-MongoDB Connector API, serialising DataFrames to BSON documents. Indexing on product_id and timestamp is applied; ObjectID getTimestamp() confirms 1.8 s mean persistence latency.

III-D. Presentation

A Streamlit dashboard is decoupled, polling MongoDB for real-time visualisations of throughput, sentiment, and latency metrics, ensuring no back-pressure on upstream tiers.

IV. SYSTEM ARCHITECTURE

The Big Data Stream architecture is a modular, four-tier distributed system designed for high-velocity data processing. By decoupling each layer, the pipeline ensures fault isolation and independent scalability. The architecture moves data from raw event sources to a real-time visualisation layer with a goal of sub-2-second end-to-end latency.



[Figure 1: Big Data Stream — Four-Tier Architecture with per-tier latency annotations]

The four layers and their roles are as follows:

- Ingestion Layer — A Unix-based stream emitter transmits JSON review data via TCP socket (Port 9999), bypassing virtualised file-system overhead.
- Processing Layer — Apache Spark Structured Streaming performs micro-batching, schema inference, and Negative-First Echo Detection (filtering for ratings ≤ 2).
- Storage Layer — MongoDB NoSQL sink persists processed sentiment signals into schema-flexible BSON documents using the Spark-MongoDB Connector API.
- Presentation Layer — Streamlit Business Intelligence dashboard queries the NoSQL store to provide real-time sentiment heatmaps and critical issue alerts.

V. KEY FORMULAS AND PERFORMANCE EQUATIONS

The following equations formalise the core performance characteristics of Big Data Stream.

V-A. End-to-End Pipeline Latency

$$L_{total} = L_{spark} + L_{persist} \quad (1)$$

Where L_{spark} is bounded by the micro-batch trigger interval (1.2 s) plus the Van Dongen latency floor (510 ms); $L_{persist}$ is measured via MongoDB ObjectID.getTimestamp().

V-B. Throughput Formula

$$T = (N_{records} / \Delta t_{window}) \times \eta_{stability} \quad (2)$$

Where $N_{records}$ is records per window, Δt_{window} is the observation window in seconds, and $\eta_{stability}$ is the empirically measured stability coefficient (95%).

V-C. Negative-First Filter Selectivity

$$S_{neg} = |\{r \in R : rating(r) \leq 2\}| / |R| \quad (3)$$

Where S_{neg} is the fraction of records flagged as high-priority negative. A lower S_{neg} reduces downstream processing load, enabling faster surface-level defect detection.

V-D. F1 Score — Lexical Echo Detection

$$F1 = 2 \times (Precision \times Recall) / (Precision + Recall) \quad (4)$$

Where Precision = 89.9% and Recall = 89.8%, yielding F1 = 89.8%. The harmonic mean penalises imbalanced precision–recall trade-offs.

V-E. Pipeline Tax

$$T_{tax} = L_{total} - L_{spark\_floor} \quad (5)$$

The Pipeline Tax quantifies overhead introduced by external sinks. Reducing MongoDB write amplification or adopting async polling would shrink this budget.

V-F. BERT Latency–Accuracy Trade-off

$$\Delta Accuracy / \Delta Latency \approx 0.004\% \text{ per ms} \quad (6)$$

Each additional millisecond of BERT inference buys only ~0.004 percentage points of accuracy under CPU-constrained WSL2, justifying the lexical model choice.

VI. IMPLEMENTATION DETAILS

The experimental environment was provisioned on a Windows Subsystem for Linux 2 (WSL2) instance running Ubuntu 24.04 LTS, hosted on an Intel Core i7 eight-core processor with 16 GB of system RAM. This configuration provided a stable and reproducible Linux-based runtime without the overhead of full hardware virtualisation.

The core processing pipeline was built upon Apache Spark 3.5 with its Structured Streaming API, chosen

for its fault-tolerant, micro-batch processing model suited to high-throughput ingestion workloads. Python 3.11 served as the primary development language for authoring Spark jobs and auxiliary scripts, while MongoDB 7.0 was employed as the sink data store.

Data ingestion was orchestrated through a purpose-built Bash script acting as a lightweight stream emitter. This script sequentially reads newline-delimited JSON records from structured flat files and transmits them over a local TCP socket bound to port 9999 using the Netcat (nc) utility. This approach effectively simulates a live data feed, enabling the Spark Structured Streaming consumer to treat the socket as a continuous unbounded source.

To ensure stable performance under a sustained ingestion rate of approximately 10,000 records per minute, the Spark runtime was tuned via explicit configuration parameters. The spark.driver.memory property was set to 4 GB, and spark.executor.cores was configured to 2, restricting each executor to two CPU cores to promote task-level parallelism while preventing resource contention on the shared eight-core host.

VII. RESULTS AND ANALYSIS

The pipeline achieved a mean end-to-end latency of 1.8 seconds under a 10,000 reviews/min stress test, verified through MongoDB ObjectID getTimestamp() extraction against TCP socket ingestion time (T-0). This confirms sub-2-second performance with 95% stability across 1,000 iterations on WSL2, where the Spark-MongoDB Connector outperformed JDBC baselines by 40% in persistence speed.

VII-A. Algorithm Selection and Model Performance

BERT variants reach 96% accuracy on e-commerce reviews but incur 1.2 s average inference latency on CPU-constrained WSL2 (Shen et al., 2020), violating the 2-second Pipeline Tax. Lexical Echo Detection sacrifices 4.7 accuracy points for sub-100 ms inference, enabling reliable micro-batch processing on i7 hardware.

TABLE II — Lexical Classifier Performance Metrics

Metric	Score
Accuracy	91.3%
Precision	89.9%
Recall	89.8%
F1 Score	89.8%

VII-B. Model Logic

Negative-First logic flags ratings ≤ 2 for immediate negative signal evaluation, prioritising high-impact feedback. A 4-character stop-word filter strips noise tokens (e.g., 'the', 'and'), reducing evaluation space and driving sub-100 ms speed across 10,000-record loads.

VIII. DISCUSSION

In testing, the core Spark processing time remained close to the 510 ms latency floor reported by Van Dongen. However, under a stress load of 10,000 records per minute, the overall end-to-end latency rose to approximately 1.9 s. This additional delay — the Pipeline Tax — mainly comes from the use of external data sinks such as MongoDB and the frequent refresh cycles of the Streamlit interface.

Unlike the 120 ms benchmark that relied on GPU-accelerated model serving, our design focuses on keeping costs low and the architecture straightforward. The system only reaches a hardware saturation point — marked by a Connection Refused error — when pushed to very high enterprise-scale workloads, confirming the robustness of the design within its target single-node deployment envelope.

IX. RELATED WORK

Modern stream processing architectures trace their roots to Zaharia et al. (2013), who introduced Resilient Distributed Datasets (RDDs) and the micro-batch paradigm in Apache Spark. This approach decomposes continuous streams into discrete time-windowed batches, balancing throughput and latency for near-real-time analytics — a foundation directly informing our pipeline's Spark Structured Streaming implementation.

In e-commerce sentiment analysis, the challenge lies in reconciling model accuracy with strict latency needs. Saadi et al. (2025) showed fine-tuned BERT reaching 96% accuracy on review corpora, while Shen et al. (2020) quantified the trade-off: BERT's superior performance comes at 300–1200 ms inference latency on CPU-only setups, making it impractical for sub-2-second pipelines on resource-limited hardware.

Among frameworks, Apache Flink (Carbone et al., 2015) excels with true event-time processing and sub-millisecond latencies via its dataflow engine.

However, Flink's JobManager/TaskManager topology, cluster demands, and tuning complexity make it unsuitable for single-node SME environments lacking dedicated operations teams — hence its rejection here in favour of Spark's simpler micro-batch model.

The proposed Big Data Stream architecture fills this gap as a low-cost, high-efficiency baseline for WSL2 deployments. Delivering 1.9-second end-to-end latency on commodity i7 hardware, it prioritises operational simplicity and feasibility over Flink-scale performance or BERT-grade accuracy, enabling real-time e-commerce sentiment monitoring without clusters or heavy infrastructure.

X. CONCLUSION

This paper presents a real-time e-commerce sentiment analysis pipeline designed specifically for WSL2's resource constraints on standard hardware. The system processes 10,000 records per minute across all four tiers with 95% stability on an i7 laptop, demonstrating robust performance under continuous load.

TCP socket ingestion via Netcat on port 9999 successfully bypassed WSL2's 9P file-system bottleneck, eliminating the latency overhead that plagued file-based approaches. This enabled consistent high-throughput data delivery directly to Spark Structured Streaming.

The Negative-First Echo Detection model achieved 91.3% classification accuracy while maintaining sub-2-second end-to-end latency, verified through MongoDB ObjectID timestamps. This balance proves commodity hardware can deliver production-grade stream analytics without distributed clusters or heavy ML models.

Future Work

Next steps include upgrading to Apache Spark 4.1 RTM for millisecond-scale micro-batch processing and implementing a hybrid classification model — using Negative-First Echo Detection for triage and LLMs for ambiguous sarcasm/irony cases. Additional work will explore Docker containerisation, adaptive batch tuning, and multilingual review support.

REFERENCES

[1] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in Proc. 24th ACM Symp. Operating Systems Principles, 2013, pp. 423–438.

[2] K. Nair, et al., "Cloud-based Spark for sentiment analysis," in Proc. IEEE Cloud Computing Conf., 2017.

[3] G. van Dongen, "Evaluation of stream processing frameworks for real-time data analysis," M.S. thesis, Delft Univ. Technology, 2020.

[4] Y. Wang and J. Duan, "LLM agents for e-commerce sentiment pipelines," in Proc. Int. Conf. Data Eng., 2024.

[5] Maryam and Farid, "GPU-accelerated micro-batch sentiment analysis with Grok-4," J. Distributed Computing, vol. 42, no. 2, pp. 55–71, 2025.

[6] M. Saadi et al., "Enhancing sentiment analysis accuracy for e-commerce platforms using a tuned BERT model," Discover Computing, vol. 28, no. 3, pp. 112–128, 2025.

[7] Y. Shen et al., "BERT inference latency on CPU-constrained environments," in Proc. NeurIPS Workshop, 2020.

[8] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and batch processing in a single engine," IEEE Data Eng. Bull., vol. 38, no. 4, pp. 28–38, 2015.

[9] C. J. Hutto and E. Gilbert, "VADER: A parsimonious rule-based model for sentiment analysis of social media text," in Proc. Int. AAAI Conf. Web and Social Media, vol. 8, no. 1, pp. 216–225, 2014.

[10] K. Banker, MongoDB in Action: Covers MongoDB 3.0, 2nd ed. Manning Publications, 2016.

[11] I. Aldasoro, L. Gambacorta, P. Giudici, and T. Leach, "The intelligent financial system: How AI is transforming finance," BIS Working Papers No. 1194, 2024.

[12] D. Pattnaik, S. Ray, and R. Raman, "Applications of artificial intelligence and machine learning in the domain of financial services," Heliyon, vol. 10, no. 1, e23492, 2024.