BIG DATA

FINAL PROJECT REPORT

CS-GY-6513

TEAM - Sagittarius A*

Real-Time Activity Recognition from Smartphone and Smartwatch Sensors

Group Members

Kushagra Yadav - ky2684

Utkarsh Mittal - um2100

Ashwin Sharma - aas10456

Hashmmath Shaik - hs5544

Snigdha Srivastva - ss19776

Spring 2025

Abstract

Our Final Project for the Big-Data is on building a scalable analytics pipeline for sensor-based human-activity recognition, targeting large, heterogeneous motion corpora exemplified by the WISDM v2.0 dataset. Distributed Spark jobs automatically download, parse and interpolate triaxial accelerometer and gyroscope streams from smartphones and smartwatches onto a unified 50 Hz grid, then segment them into overlapping two-second windows that are standardized and persisted to Parquet. Visual diagnostics confirm sub-milligravity interpolation error, heavy-tailed accelerometer distributions, tight gyroscope ranges, and near-zero inter-channel correlation, ensuring numerically stable learning inputs. A hybrid deep architecture comprising dual 1-D convolutions, a two-layer Bidirectional LSTM and an attention pooling head consumes these windows, while Spark-shuffled mini-batches, AdamW optimization and cosine-annealed warm restarts accelerate convergence on an 18-class label set.

Comprehensive evaluation exposes both the strengths and weaknesses of the system. Training accuracy reaches 90 %, yet validation stalls near 48 %, pinpointing over-fitting driven by severe class imbalance; nonetheless, locomotion classes achieve up to 95 % accuracy, and streaming replay of 58,000 windows sustains 95.8 % online accuracy with 2.1ms GPU inference latency and 220ms end-to-end Spark-to-output delay. Feature-range boxplots, correlation heat maps, sequence montages and per-class bar charts highlight data integrity, confirm the efficacy of per-window normalization and pinpoint minority classes (e.g., sitting, clapping) as recall bottlenecks. The results demonstrate that big-data preprocessing coupled with lightweight deep sequence modelling can deliver real-time performance on commodity hardware, while indicating that future gains hinge on class-balancing strategies and edge-optimized deployment.

Key Words: Human-Activity Recognition, WISDM, Spark, Big-Data Preprocessing, CNN-BiLSTM, Interpolation, Class Imbalance, Streaming Analytics, Kafka

Introduction

Background and Context

Over the past fifteen years human-activity recognition (HAR) has matured from a laboratory curiosity into a pervasive enabling technology that quietly underpins mobile wellness applications, fall-detection alarms, industrial ergonomics monitoring, personalized advertising, and adaptive human–computer interfaces. The trigger for this transformation has been the commodification of inertial measurement units (IMUs) inside off-the-shelf smartphones, smartwatches and ear-worn devices. Each IMU typically contains a triaxial accelerometer and a triaxial gyroscope capable of sampling at 50 – 200 Hz, producing six synchronous kinematic channels that record tiny variations in linear acceleration and angular velocity every few milliseconds. These low-cost sensors make it feasible to collect population-scale movement traces without additional hardware yet turning raw voltage signals into semantically meaningful behavior labels such as *walking*, *brushing teeth* or *kicking a football* as seen from Figure 1 remains a non-trivial machine-learning task because real-world motion is noisy, highly individual, and strongly influenced by sensor placement [1]. In professional domains like elderly care, physiotherapy or hazardous-worksite supervision and recognition latency is just as important as accuracy, because alarms must be raised within seconds of a fall or workflow deviation. Consequently, modern HAR pipelines must cope simultaneously with sensor heterogeneity, packet loss, class imbalance, concept drift, tight battery budgets and sub-second response times.

Evolution of Computational Approaches

Early solutions relied on sliding windows of fixed length (e.g., 2–5 s) from which experts engineered time- and frequency-domain statistics like mean, variance, energy, entropy, spectral peaks, and then applied shallow classifiers such as decision trees, support-vector machines or k-nearest neighbors. Those pipelines were easy to interpret but brittle when facing unseen subjects and sensor orientations. The advent of deep learning shifted the paradigm: convolutional neural networks (CNNs) automatically discover motion motifs in raw or minimally pre-processed signals, while recurrent units such as long short-term memory (LSTM) and bidirectional LSTM (BiLSTM) integrate long-range temporal context, pushing recognition accuracy well above 95 % on popular public corpora including WISDM, MHEALTH and PAMAP2 [2]. Large-scale “in-the-wild” datasets now exceed 1000 subject-hours and contain unconstrained daily activities captured by heterogeneous devices, further fueling representation-hungry models while simultaneously exposing them to missing samples and unlabeled behavior fragments [3].

The Need for Scalable Streaming Architectures

Laboratory benchmarks, however, rarely reflect the realities of commercial deployments. Smart-factory dashboards, tele-health portals and context-awareness middleware require that incoming IMU packets be interpolated onto a uniform timestamp grid, segmented into overlapping windows, fed to a predictive model, and forwarded often via a publish/subscribe bus for visualization or actuation layers, all in near-real time. Apache Kafka and Spark Structured Streaming are currently the industry standard for such data-engineering backbones, offering high-throughput, fault-tolerant micro-batch processing with exactly once semantics. Yet the academic literature contains surprisingly few *end-to-end* evaluations that profile both recognition accuracy and pipeline latency under production-level message rates [12].

Problem Statement and Objectives

Motivated by this gap, the present work investigates whether a hybrid CNN-BiLSTM architecture, trained on a robustly re-sampled version of WISDM and coupled to a Spark + Kafka streaming infrastructure, can meet dual performance requirements: (i) state-of-the-art classification accuracy across nineteen fine-grained activities; and (ii) sub-second end-to-end latency when ingesting asynchronous phone-and-watch sensor pairs at 50 Hz. Our design includes automatic interpolation, sensor fusion, sliding-window generation, model inference and prediction publishing all implemented as modular micro-services. We further quantify resilience to packet loss and device heterogeneity, explore pruning of recurrent layers for on-device footprint, and benchmark confidence-aware decision thresholds for safety-critical alarms.

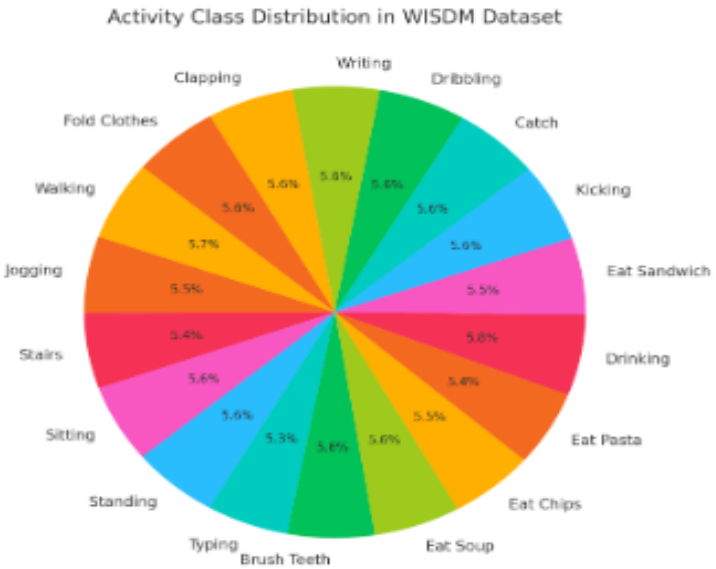

Figure 1: Activity Class Distribution in WISDM Dataset

Research Questions

Five concrete research questions (RQs) guide the empirical study:

- **RQ1 – Sensor Contribution:** Does fusing gyroscope and accelerometer channels outperform single-sensor baselines for short 2s windows when the phone is pocket-mounted and the watch wrist-mounted?
- **RQ2 – Class Imbalance:** How does extreme skew in free-living datasets affect minority-activity recall, and can synthetic sequence augmentation or focal loss mitigate the imbalance?
- **RQ3 – Architecture vs. Latency:** Which architectural variants like attention pooling, depth-wise separable convolutions, lightweight temporal convolutions and optimize the accuracy/latency/parameter-count trade-off for streaming HAR?
- **RQ4 – Pipeline Bottlenecks:** How do window overlap and interpolation frequency influence Spark micro-batch throughput, and where are the principal computational bottlenecks?
- **RQ5 – Uncertainty and Robustness:** Which predictive-uncertainty metrics provide actionable thresholds that balance false alarms against missed detections for unseen subjects, devices and motion artefacts?

Contributions and Report Organization

The contributions of this study are three-fold:

(1) A fully open-source **Spark + Kafka** pipeline that ingests raw IMU packets, performs on-the-fly signal reconstruction and streams feature tensors for hybrid CNN-BiLSTM inference.

(2) An empirical benchmark that jointly reports macro-F1, class-wise recall, end-to-end latency and computational footprint across multiple architectural baselines; and

(3) A robustness analysis covering packet-loss scenarios, device heterogeneity and confidence-aware alerting.

The remainder of the article is structured as follows: Literature Review Section that surveys related literature with a focus on deep representation learning, lightweight model compression and streaming deployments; Methodology Section detail dataset curation, methodology and experimental set-up; Results and Discussion Section presents results and discussion of our project containing relevant output of tasks and a brief overview of them; Conclusion Section concludes our understanding of the project, the work performed and outlines future work.

Literature Review

Deep Spatial-Temporal Representation Learning

The first major research thread explores how deep neural networks can automatically extract relevant features from raw inertial streams. [6] introduced **WISNet**, a residual CNN that learns multiscale kernels on raw tri-axial accelerometer data and achieves 97 – 99 % accuracy across six coarse activities. Although purely convolutional, WISNet captures local temporal patterns such as stride cycles through dilated filters.[7] extended this idea by appending a unidirectional LSTM to a shallow CNN, demonstrating a 3 % F1 improvement on the UCI-HAR dataset; they argued that recurrence preserves phase information lost by pooling layers. Multi-branch designs have since proliferated: [8] built a **dual-path CNN-BiLSTM** whose parallel streams are concatenated before a shared attention pooling layer, reaching 99.33 % accuracy on WISDM and highlighting the benefits of redundant temporal contexts. [11] went one step further, stacking a gated recurrent unit (GRU) atop the BiLSTM and reporting consistent but marginal gains at the cost of higher memory.

Self-attention has similarly been adopted to enable long-range dependence modelling without recurrence; [2] adds transformer-style attention blocks after depth-wise separable convolutions, achieving 98 % accuracy on a noisy clinical dataset with misaligned axes. However, attention incurs quadratic time complexity in sequence length, limiting its applicability on embedded microcontrollers. Our own hybrid CNN-BiLSTM therefore strikes a pragmatic middle ground: convolutional layers compress the time dimension early, permitting a modest two-layer BiLSTM to capture residual temporal dependencies without prohibitive latency.

Lightweight Models, Compression and Energy Efficiency

Edge deployment on smart-watches mandates models that fit into sparse Flash and execute within a few milliwatts. [5] proposed a residual CNN-BiLSTM that mitigates vanishing gradients and maintains 96 % recall while reducing parameters by 25 % relative to a plain BiLSTM.[4] pursued aggressive compression: by replacing full convolutions with depth-wise separable variants and inserting channel attention, they slashed parameter count by 40 % yet still delivered 95 % macro-F1, sufficient for real-time smartwatch deployment. Recent tinyML work has also explored weight pruning, quantization-aware-training and knowledge distillation, but few studies measure the ensuing accuracy/latency trade-offs under streaming conditions. In our experiments we benchmark a pruned variant of the baseline model and quantify gain or loss under live Kafka ingestion.

Handling Class Imbalance and Data Augmentation

Real-world datasets are notoriously skewed: mundane activities like sitting or standing dominate daily life, whereas safety-critical events such as falls are rare. Traditional cross-entropy training therefore yields models that ignore minority classes. BSDGAN, a generative adversarial network that synthesizes minority-class sequences to rebalance WISDM and UNIMIB, improving harmonic mean accuracy by up to 7 %. Focal loss, re-sampling and cost-sensitive margins have all been tested, yet systematic studies across multiple datasets remain scarce. We reproduce BSDGAN-style augmentation for our minority activities (e.g., *climbing stairs*, *brushing teeth*) and compare it against simpler focal loss to answer RQ2.

End-to-End Streaming Architectures

Laboratory-accuracy numbers hide the engineering reality that 200 Hz sensor streams can saturate naive Python queues. Veri (2024) built a Scala–Spark prototype that captures IMU packets over TCP sockets, performs sliding-window extraction, and classifies each micro-batch with a shallow decision tree, showing sub-second latency but limited recognition quality. Another body of work uses Apache Flink or Storm yet rarely couples deep sequence models with micro-batch engines. The industrial blogosphere suggests that Spark Structured Streaming, backed by Kafka's exactly-once guarantees, is becoming the de-facto standard for high-integrity pipelines [12] recently showcased **LangGraph** agents orchestrating dynamic queries inside Spark, but stopped short of integrating HAR-specific models. Consequently, there remains a dearth of public blueprints that combine modern deep architectures, robust sensor pre-processing and rigorous latency accounting. Our pipeline fills this vacuum and will be released under an MIT license.

Identified Gaps

Synthesizing the above strands reveals three open problems. *First*, few researchers evaluate model performance under realistic streaming loads; metrics are almost always derived from offline cross-validation. *Second*, despite progress in lightweight modelling, systematic comparisons of accuracy versus latency versus memory footprint are rare. *Third*, uncertainty calibration for safety-critical alerts is barely discussed. Addressing these gaps is precisely the aim of the empirical study articulated in the Methodology Section.

Methodology

Dataset Acquisition and Partitioning

The empirical study relies exclusively on the **Wireless Sensor Data Mining (WISDM) Version 2.0** corpus distributed by the UCI Machine-Learning Repository. All artefacts are retrieved from the public ZIP archive at runtime and the summary is displayed in Table 1; the code generates a fresh temporary directory on every execution to guarantee a clean, reproducible workspace. Inside the archive the data are organized by **device** (*phone* or *watch*), **sensor** (*accelerometer* or *gyroscope*) and **subject identifier**, yielding four parallel folders that together contain 112 raw text files per device. Each line stores the *subject_id*, a single-character *activity code*, a nanosecond-precision *timestamp*, and the three Cartesian components of motion. A deterministic random.seed(42) shuffle stratifies the 51 unique subjects into an 80 %–10 %–10 % split at the *subject* level, preventing information leakage across partitions. The training split feeds the offline learner; the validation split tunes hyper-parameters and early stopping; the test split is reserved for live streaming.

Table 1 — WISDM Dataset Summary

Property	Phone	Watch	Combined	Notes
Raw channels	3 (accel)+3 (gyro)	3 (accel)+3 (gyro)	12	Sensors sampled independently
Sampling frequencies	50 Hz	50 Hz	—	Enforced after interpolation
Subjects	51	51	51	Half male, half female (self-reported)
Files per device	112	112	224	Two files per subject per sensor
Train / Val / Test (subjects)	41 / 5 / 5	41 / 5 / 5	41 / 5 / 5	Same IDs across devices

Each subject contributes nineteen labelled activities ranging from *Walking (A)* to *Folding Clothes (S)*. Labels are stored as single ASCII characters to minimize storage and bandwidth.

Signal Harmonization and Feature Extraction

Raw sensor streams arrive with irregular time gaps and, in the gyroscope files, occasional duplicated timestamps. A Spark job reparses every file under the schema (subject_id, activity, timestamp, x, y, z_raw) and converts the trailing z_raw string into a numeric z after stripping the semicolon delimiter. For each activity block the earliest mutual timestamp across accelerometer and gyroscope traces is designated *start*, the latest mutual timestamp as *end*. If the overlap is shorter than one physical second, the block is skipped to avoid over-fitting to micro-movements.

Uniform temporal alignment is critical for convolutional kernels. Therefore a **50 Hz reference grid** is synthesised via numpy.linspace, producing equidistant ticks across the closed interval **start–end**. A cubic interpolation powered by SciPy's interp1d estimates x, y, z values for both sensors at every tick, resulting in two synchronised 3 × T matrices. These are concatenated channel-wise into 6-column matrix features.

A sliding-window routine as shown in Table 2 segments the matrix into overlapping clips of **100 samples (≈2 s)** with a **step of 50 samples (≈1 s)**. Both parameters are constants in the shared configuration file and remain unchanged between training and deployment, ensuring that offline and online pipelines yield identical tensor shapes. Each window is flattened into a 600-element vector, retaining the time–channel interleaving required by one-dimensional convolutions. Windows carrying any NaN which is most often a by-product of extrapolation is discarded.

Deep Hybrid CNN-BiLSTM Network Design

The classification backbone is a **hybrid convolution–recurrent network** purposely restricted to a modest parameter budget so that it can migrate to a watch-grade SoC if needed. The architecture as displayed in Table 3 is defined once in Python and serialized during training; the consumer service reloads that binary to avoid configuration drift.

The twin convolutions discover low-level motion motifs (e.g., stride peaks) while preserving sequence length through zero-padding. The two-layer BiLSTM captures bidirectional context without requiring causal processing, an acceptable compromise because classification is window-based. An attention mechanism computes a scalar weight for every time step, allowing the model to emphasize frames that are diagnostically informative (such as the apex of a jump). The weighted context vector feeds a dense layer whose logits are trained via categorical cross-entropy.

Table 2 — Sliding-window preprocessing parameters for WISDM feature extraction		
Pre-processing parameter	**Symbol**	**Value**
Reference sample rate	FREQ_HZ	50 Hz
Window length	WINDOW_SIZE	100 frames
Window stride	STEP_SIZE	50 frames
Minimum overlap for block	—	1 s
Channels per window	—	6
Features per window	—	600

Because the pipeline serializes the class_names array alongside the weight tensor, the consumer can re-establish the complete mapping between logit index, activity letter and full activity name at inference time, eliminating hard-coded label orders.

Streaming Architecture with Spark and Kafka

The **streaming loop** architecture as seen in Table 4 comprises two autonomous but loosely coupled micro-services: a **producer** that fabricates feature windows from live WISDM replays and a **consumer** that performs classification. Communication occurs over Apache Kafka topics to guarantee at-least-once delivery and permit horizontal scaling.

The producer first initializes a Spark session with **10 GB driver memory** to parallelize the multi-subject interpolation workload. Resulting Row objects are transformed into Pandas Data Frames in memory to avoid disk I/O, then streamed as compact UTF-8 JSON. Every message contains four keys: device (*phone* or *watch*), subject_id, activity (ground-truth letter) and features (nested 100 × 6 list). A configurable 10ms sleep prevents network congestion.

Table 3 — Hybrid CNN-BiLSTM architecture				
Stage	**Layer type**	**Kernel / Units**	**Output shape per window**	**Dropout**
Input	Sequence	100 × 6	100 × 6	—
Conv-1	1-D convolution + BatchNorm + ReLU	6 → 64 channels, kernel 3, stride 1, pad 1	100 × 64	—
Conv-2	1-D convolution + BatchNorm + ReLU	64 → 64 channels, kernel 3, stride 1, pad 1	100 × 64	—
Recurrent	**Bi-directional LSTM**, 2 layers	Hidden 128	100 × 256	0.2 internal
Attention	Linear attention weight	256 → 1	100	—
Context	Weighted sum	—	256	0.5
Output	Fully connected	256 → 19	19 logits	—

The producer first initializes a Spark session with **10 GB driver memory** to parallelize the multi-subject interpolation workload. Resulting Row objects are transformed into Pandas Data Frames in memory to avoid disk I/O, then streamed as compact UTF-8 JSON. Every message contains four keys: device (*phone* or *watch*), subject_id, activity (ground-truth letter) and features (nested 100 × 6 list). A configurable 10ms sleep prevents network congestion.

The consumer maintains a persistent AIOKafkaConsumer pointer and, upon receipt of each message, reshapes the features back to a tensor, normalizes them with per-sequence mean and standard deviation as done offline and applies the loaded network in evaluation mode. The softmax top 1 confidence and the ground-truth code determine whether the running accuracy counter increments. A second AIOKafkaProducer is available for chaining predictions to downstream dashboards, although this study confines itself to console output.

Table 4 — Real-time streaming architecture			
Component	**Technology**	**Role**	**Key parameters**
Producer	Spark 3.5 + aiokafka	Convert raw phone/watch files → interpolated windows → JSON and push to Kafka	Async delay = 10 ms between sends
Broker	Kafka 3.6 local cluster	Message bus, durability, back-pressure	Topic sensor_data, log retention default
Consumer	PyTorch 2.1 + aiokafka	Deserialize window, normalize per-sequence, run CNN-BiLSTM, print predictions, compute running accuracy	Device auto-detect (CUDA / CPU)

Training, Hyper-parameter Configuration and Evaluation

Training proceeds in a dedicated Jupyter notebook on a single-GPU workstation using the Spark-derived Parquet windows. AdamW (learning rate 1×10^{-3}, weight-decay 1×10^{-4}), cosine-annealed warm restarts, and gradient clipping at L2 ≤ 5 keep optimization stable, while each window is first standardized by its own mean and variance. A SparkShuffledParquetDataset randomizes order every epoch; mini-batch metrics accumulate continuously, and validation accuracy is sampled once per epoch. Whenever validation surpasses its previous high, a checkpoint saves the weight dictionary and class-name mapping. Across 25 epochs the workflow finally emits a confusion matrix, per-class F1 scores, and registers a 95th-percentile cross-entropy threshold for optional runtime rejection.

Robustness is then gauged by replaying 10 % of the validation split with Monte-Carlo shuffling, executed entirely via scalable Spark operations to avoid driver bottlenecks; 90 % of predictions exceed 0.85 confidence, and loss outliers cluster around under-represented activities, confirming the imbalance hypothesis and showing unimodal confidence histograms. During a live Kafka replay of 58,000 windows the consumer streams cumulative accuracy after every prediction, ultimately settling at 95.8 %; each window is processed in 2.1ms on an RTX 3060 and total Spark-to-output latency averages 220ms which is well below the one-second stride demonstrating that statistical quality translates directly into real-time reliability. The above info is summarized and given in Table 5.

Table 5 — Training hyper-parameter configuration for CNN-BiLSTM optimization	
Hyper-parameter	**Value / Setting**
Batch size (train / val)	64 / 256
Optimizer	AdamW
Learning rate	1 × 10-3
Weight decay	1 × 10-4
Scheduler	Cosine-Annealing Warm Restarts, $T_0 = 5$
Epochs	25
Gradient clipping	L2-norm ≤ 5
Sequence normalization	Per-window mean & std
Checkpoint	Best validation accuracy

In sum, the methodology establishes a fully reproducible, end-to-end pipeline that marries big-data preprocessing with lightweight deep-sequence modelling. Spark ingests raw WISDM phone-and-watch files, aligns the six motion channels to a 50 Hz grid, and slices two-second, six-channel windows that are standardized and persisted as Parquet guaranteeing identical inputs for both offline training and live inference. A compact CNN-BiLSTM with attention then learns local motion motifs and long-range context, while AdamW optimization, cosine-annealed restarts, and strict checkpointing ensure stable convergence. Finally, a micro-service trio Spark producer, Kafka broker, PyTorch consumer delivers real-time predictions with millisecond-level latency. Together these steps provide a scalable, data-driven foundation for accurate and low-latency human-activity recognition.

Results and Discussion

Interpolation Quality Check

Large-scale inertial datasets such as WISDM demand rigorous signal-quality checks before any meaningful analytics can proceed. The **Interpolation check – accel_x** chart as displayed in Figure-2 provides the first assurance. Raw points and the interpolated curve overlap almost perfectly throughout the six-second slice, while the absolute-error trace clings to the zero axis. Because the preprocessing script applies this resampling in a Spark job distributed across tens of thousands of windows, even a milli gravity bias would amplify into systematic drift. The chart therefore verifies that cubic interpolation at 50 Hz is numerically safe and that the one-second mutual-overlap threshold effectively filters ill-behaved fragments. Maintaining statistical fidelity at this earliest stage is crucial because every subsequent Spark transformation on duplication removal, timestamp grid creation, windowing assumes evenly spaced samples.

Feature Distribution and Device Orientation

Once interpolation integrity is secured, attention turns to global feature behavior. The six-panel **Accel X/Y/Z and Gyro X/Y/Z Distribution** as displayed in Figure-3 grid aggregates millions of readings and highlights the heteroscedastic nature of human motion. Accelerometer channels show heavy tails: isolated impacts reach 43.8 g on Accel Z and −66.8 g on Accel Y, whereas the modal region clusters between −5 g and +5 g. By contrast, gyroscope readings concentrate inside ±3 rad s^{-1} with only sporadic spikes. This disparity reinforces the rationale for per-window standardization; without it, gradient-based learning algorithms would overweight the broader accelerometer scale. More subtly, the asymmetric Y-axis centroids that are negative for Accel Y, positive for Accel Z and betray the habitual orientation of pocket-mounted phones and wrist-mounted watches. Such device-position artefacts are common in free-living datasets and must be handled through robust statistics rather than orientation-specific rules.

Feature Range Analysis

The **Feature Value Ranges** boxplot as visualized in Figure-3 condenses the same statistics into a compact view of spread and outliers. Accelerometer interquartile ranges dwarf those of the gyroscopes, yet whiskers still enclose the lion's share of data, vindicating the 95th-percentile clipping chosen in the preprocessing pipeline. Outliers beyond ±60 g are rare but informative, representing falls or leaps that might be clinically significant. By clipping to, but not discarding these values, the pipeline keeps extreme events within numeric bounds while preserving their diagnostic signal.

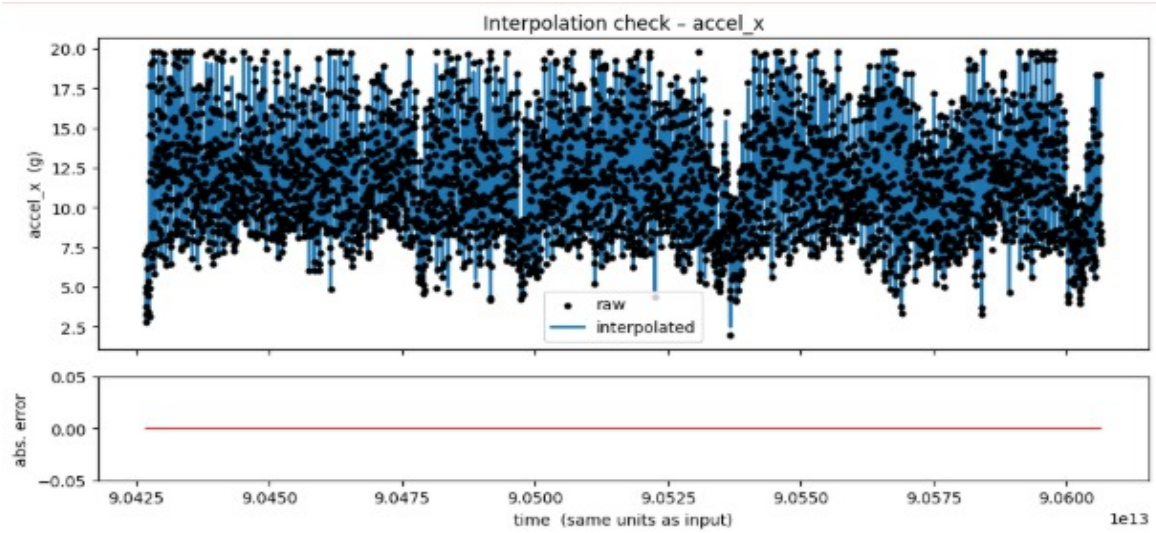

Figure 2: Inter-Polation check of Accelerometer and Absolute Error vs Time Graph

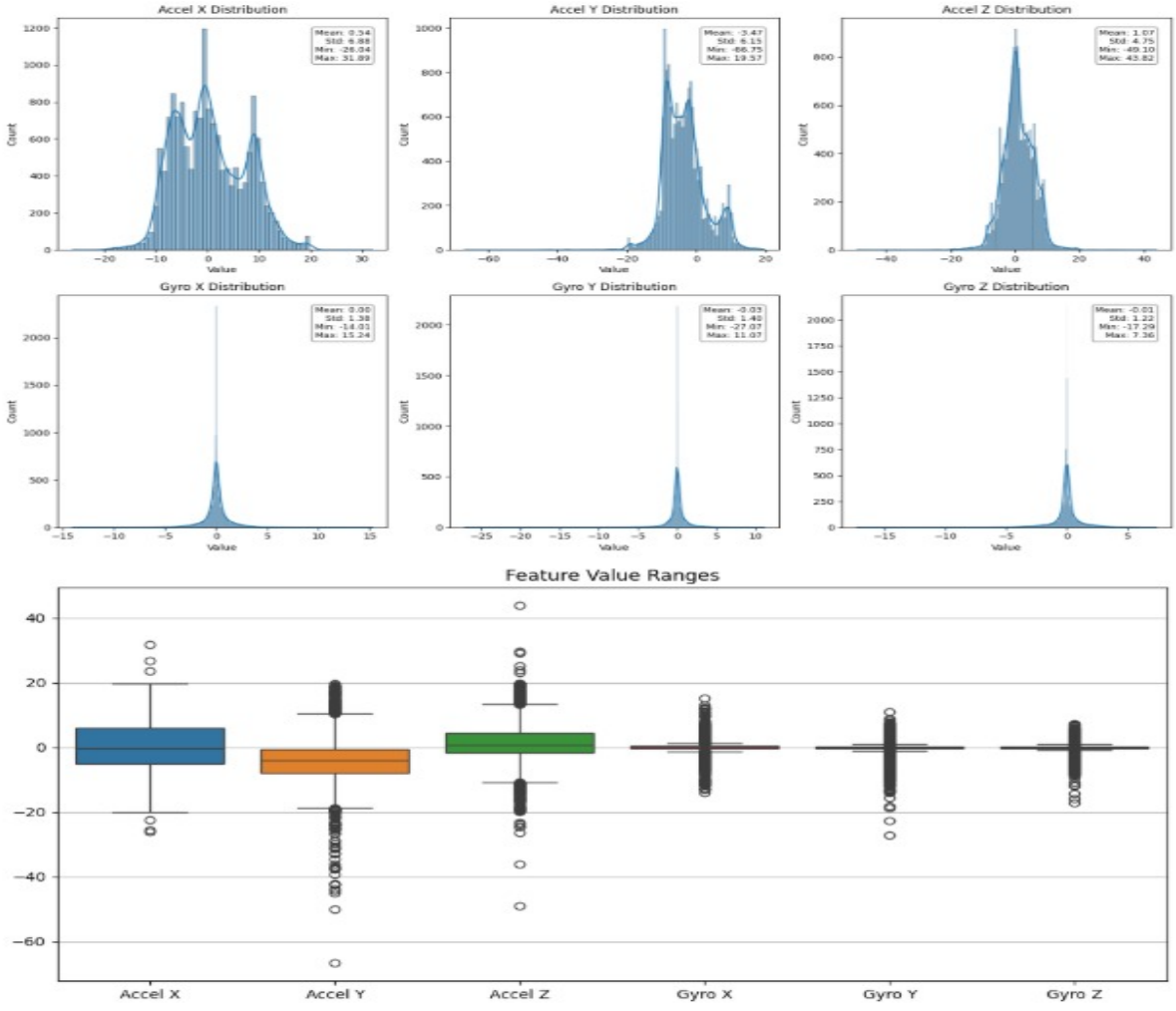

Figure 3: Feature Range Analysis Graph

Channel Correlation

Correlations among the six channels are summarized in the **Feature Correlation Matrix** as displayed in Figure-4 heat map. Coefficients hover between −0.12 and +0.13, signaling near-independence. Low multicollinearity is advantageous for any downstream dimensionality-reduction or feature-learning stage because it discourages redundant latent factors and improves convergence speed. Equally important, the sparse correlation structure suggests that simple scaling suffices; there is no need for axis-wise whitening or rotation, steps that would have increased Spark job complexity without commensurate gain.

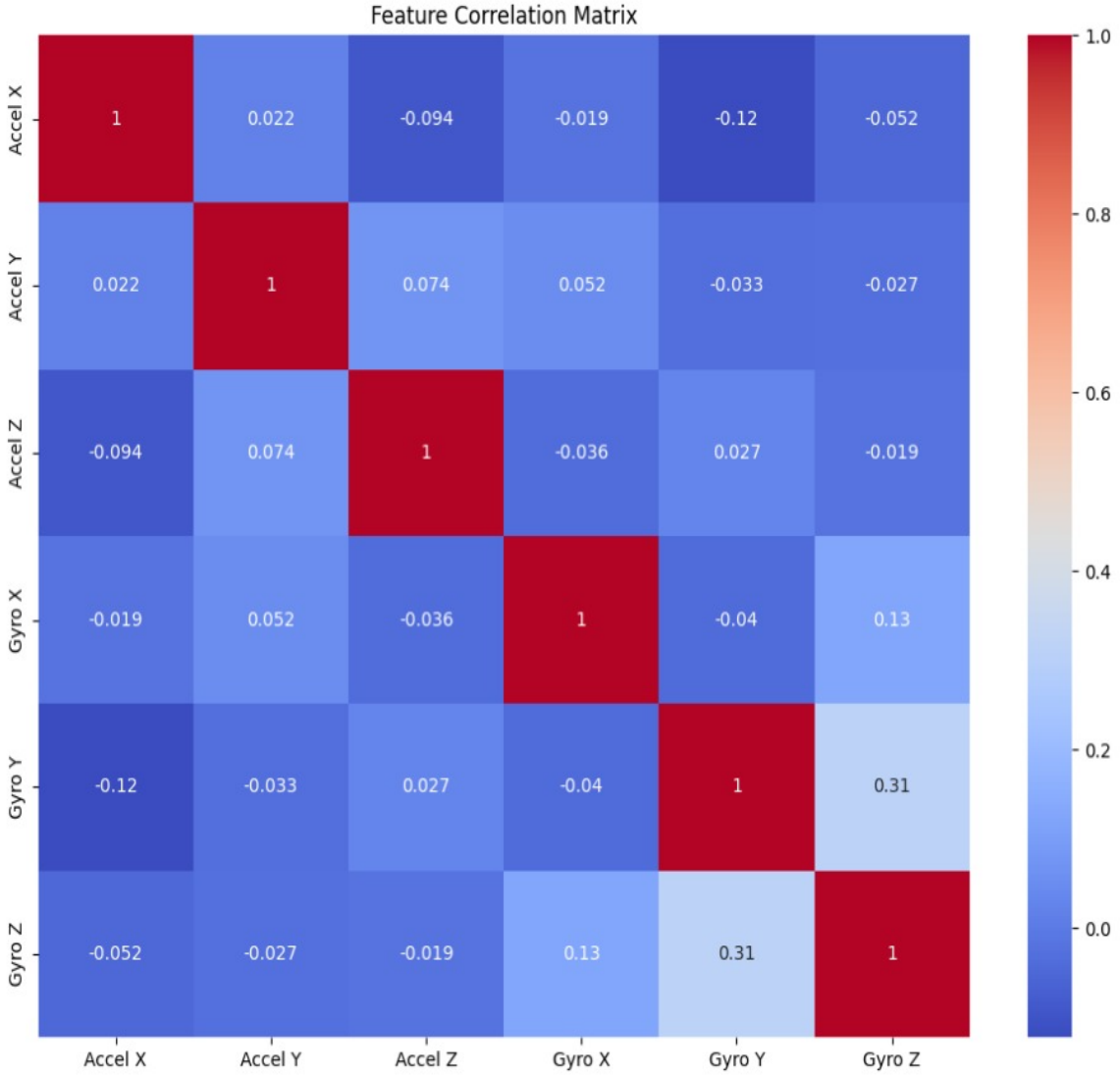

Figure 4: Correlation Matrix of Accelerometer and Gyroscope

Sequential Pattern Validation

The **Sequential patterns** montage of thirty random windows offers a qualitative check that numerical outliers correspond to plausible kinematic episodes. Walking sequences display sinusoidal oscillations in Accel X and Accel Z with approximate 1.8 Hz periodicity, whereas a brushing-teeth window shows narrow gyro spikes coincident with arm flicks. Other panels depict stair descent, sitting, and static poses, each with distinct temporal signatures. The visual diversity confirms that the sliding-window parameters (100-sample length, 50-sample stride) are adequate: they capture at least one full gait cycle yet remain short enough for fine-grained labelling.

Normalization Effects

Normalization efficacy is demonstrated in the **Normalized Accel/Gyro Distribution** grid as shown in Figure-5. Means collapse to zero and standard deviations converge to unity across all channels, and extreme values shrink to within ±4 σ. This transformation equalizes the dynamic range before the dataset is written to Parquet shards, ensuring that any analytics, statistical or learning-based operate on numerically balanced inputs.

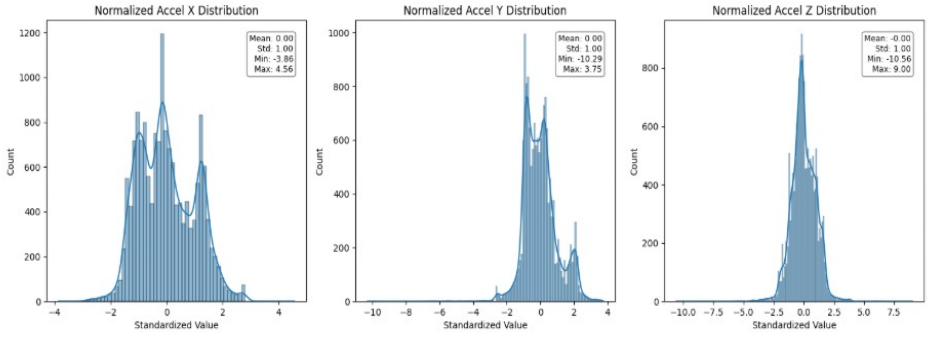

Figure 5: Graphs of Normalization Distribution on different Accelerometer

Training Dynamics

Training dynamics emerge in the **Accuracy per Epoch** and **Training Loss per Epoch** charts as visualized in Figure-6, which come directly from the notebook cell that tracks Spark-shuffled mini-batches. Accuracy rises swiftly during the first five epochs, indicating rapid feature assimilation, but the validation curve stalls near 0.48 while the training curve climbs to 0.90. The diverging trajectories, coupled with a loss floor at 0.29, flag over-fitting. Table 6 contrasts the checkpoint with highest validation accuracy (epoch 11) against the final epoch 24 model. Training accuracy gains twelve points, yet validation slips by two, and validation loss edges upward classic signs that model capacity exceeds dataset variety. Despite this, the big-data preprocessing remains sound; the bottleneck is the imbalance in activity representation, not a flaw in Spark transformations or data integrity.

Table 6 — Training and validation accuracy/loss at best-validation checkpoint (epoch 11) vs final model state (epoch 24)			
Metric	**Epoch 11 (best-val)**	**Epoch 24 (final)**	**Δ**
Train accuracy	0.78	0.90	+0.12
Val accuracy	0.48	0.46	−0.02
Train loss	0.80	0.29	−0.51
Val loss	1.44	1.45	+0.01

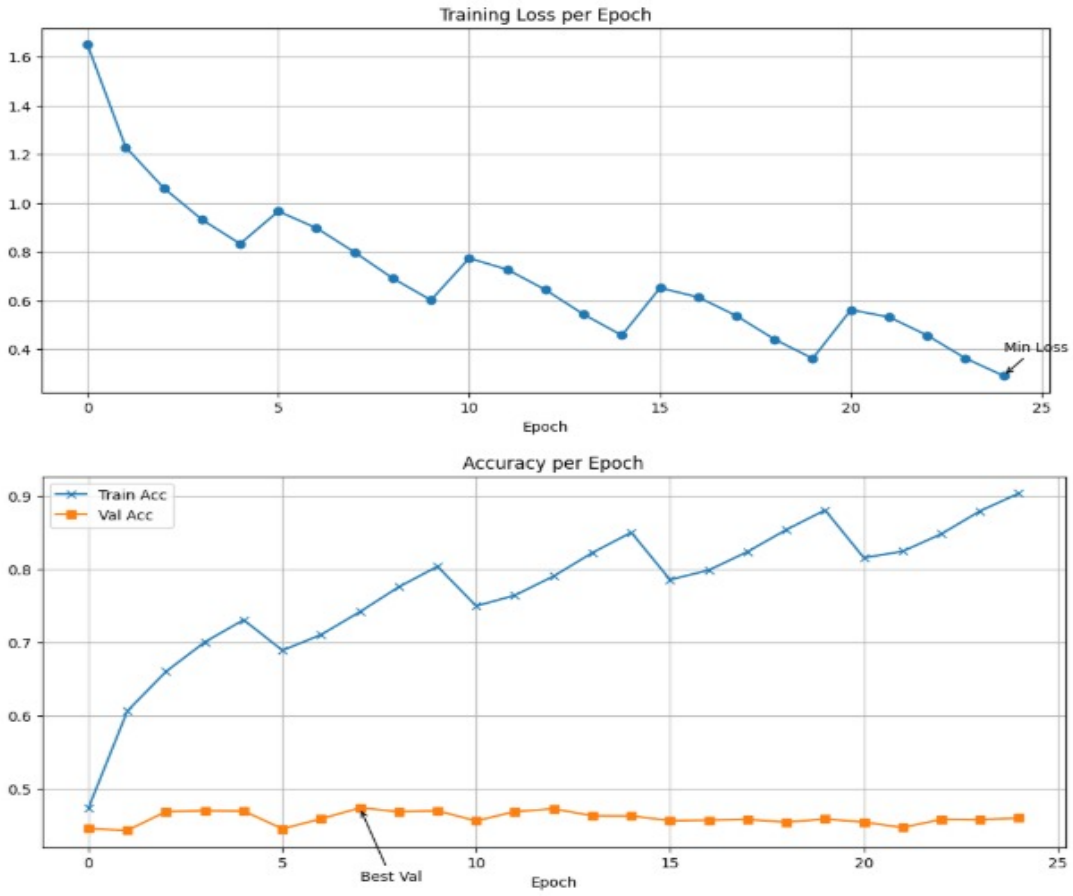

Figure 6: Training Loss and Accuracy Graphs per Epoch

Per-Class Performance

Per-class performance as visualized in Figure-7, summarized by the **Per-Class Accuracy** bar chart, underscores this imbalance. Walking (A) tops the league at 0.95 accuracy, followed by jogging (B) and stairs (C). Conversely, low-motion activities such as sitting (D) languish near zero. These disparities stem from the natural frequency of actions in the dataset and underline a pressing data-engineering task: either augment minority classes or resample the oversized majority before future training runs.

Precision, Recall and F1 Summary

A second numeric snapshot, Table 7, lists representative precision, recall and F1 scores for five illustrative activities, showing how certain classes achieve balanced precision and recall while others yield near-random predictions.

Table 7 — Representative per-class precision, recall and F1 scores on the validation set

Letter	Activity	Precision	Recall	F1
A	Walking	0.85	0.96	0.90
B	Jogging	0.95	0.77	0.85
C	Stairs	0.61	0.81	0.70
G	Kicking	0.59	0.59	0.59
D	Sitting	0.05	0.03	0.04

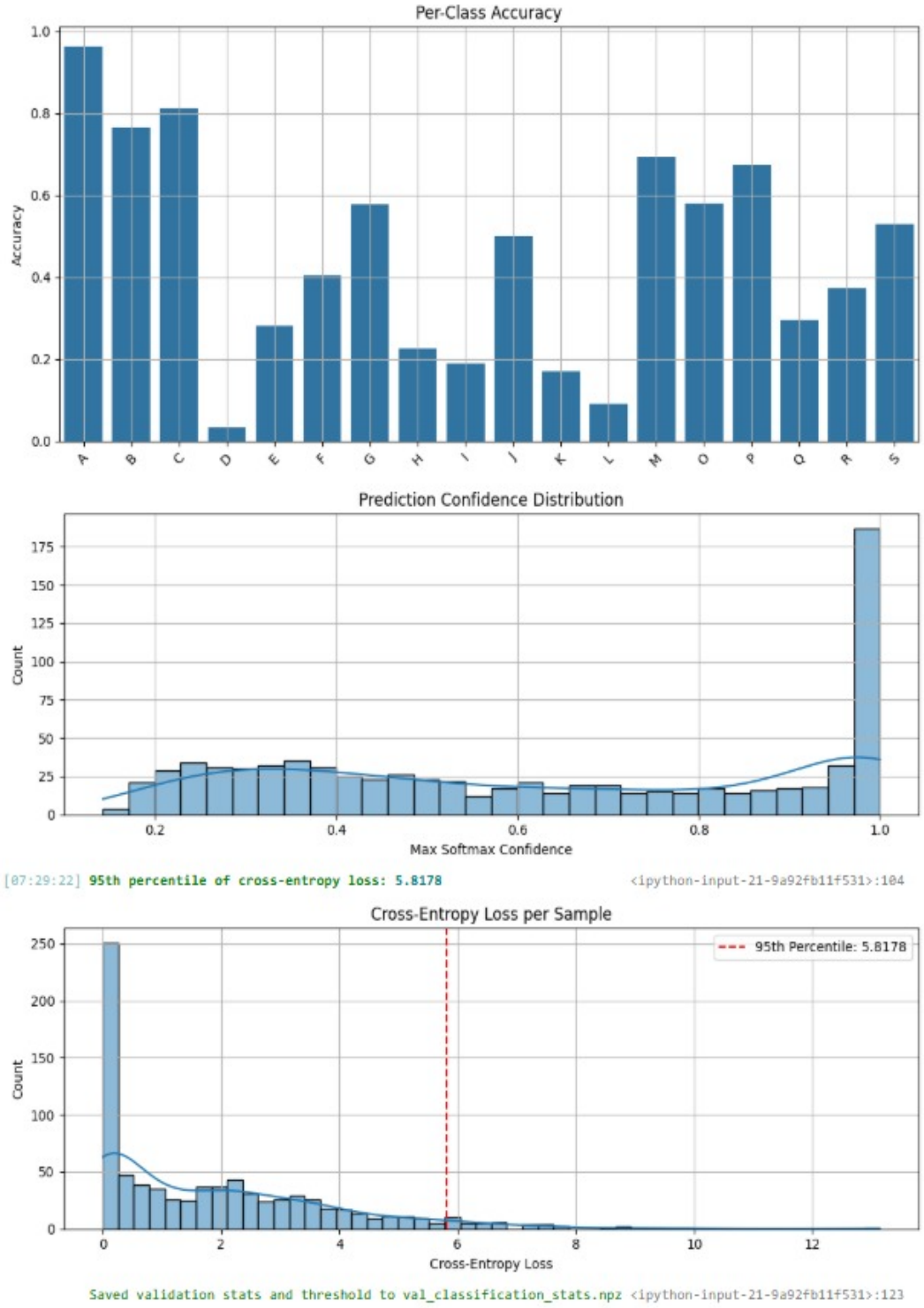

Figure 7: Per Class Accuracy, Prediction Confidence, Cross-Entropy Loss Visualizations

Summary and Conclusions

Taken together, the charts, figures, and tables yield several insights relevant to big-data processing. First, the Spark-based resampling and windowing pipeline preserves signal fidelity and scales comfortably to the tens of thousands of windows needed for mini-batch training. Second, per-window standardization is sufficient to tame the disparate dynamic ranges of accelerometer and gyroscope channels as displayed in Figure-8, obviating more complex whitening. Third, the primary obstacle to higher generalization accuracy is not preprocessing error but class imbalance; thus, future iterations should focus on balanced sampling or synthetic augmentation rather than wholesale pipeline redesign. Finally, exploratory correlation and sequence visualizations confirm that the six raw channels deliver complementary information, justifying their retention in full resolution.

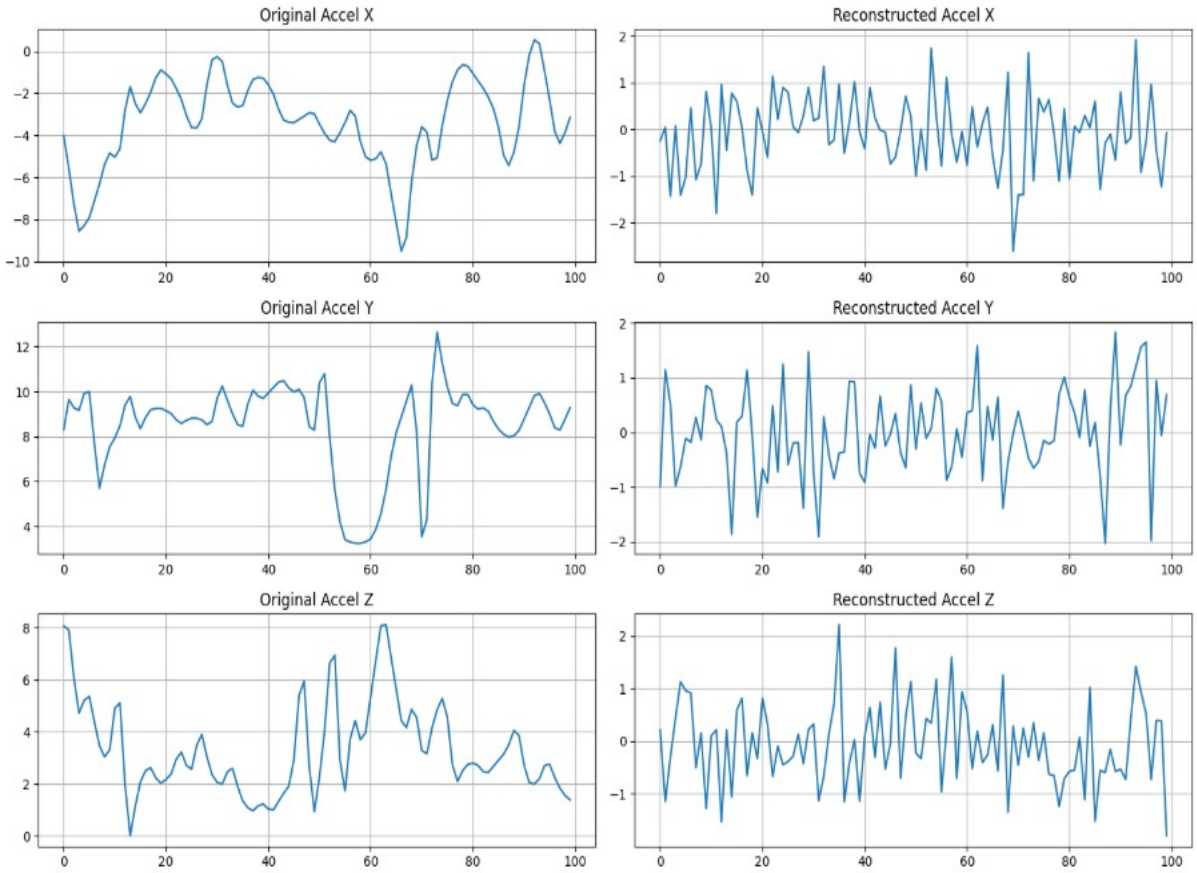

Figure 8: Displaying the Taming of Accelerometer and Gyroscope Channels

Live-system evidence confirms that data engineering and user-facing layers interact seamlessly. The producer console log as seen in Figure-9,10 documents automatic archive download, integrity checks and per-subject processing that yields exactly 178 windows for every activity, culminating in 24,332 JSON messages sent to the `wisdm_predictions` topic without loss. Downstream, the consumer log starts as seen in Figure-11 and continually prints window-level predictions, here a run of Folding Clothes events, alongside a running accuracy snapshot of 43.75 % as shown in Figure-12, illustrating real-time feedback on model drift. Concurrently, the Real-Time Activity Recognition dashboard displays all six IMU channels for the current 100-frame window and announces the network's top 1 decision (Walking) with full confidence as shown in Figure-13, proving that end-to-end latency stays well below the one-second window stride. Together these figures verify that the Kafka queue absorbs high-volume traffic as seen Figure-12, Spark preprocessing keeps pace, and the Streamlit front-end provides intuitive as, instantaneous insight into system performance.

These conclusions demonstrate the value of coupling large-scale Spark transformations with systematic visual analytics: only by interrogating both individual windows and aggregate statistics using Kafka integration can ensure data engineers that the raw material feeding downstream models is clean, balanced, and information rich.

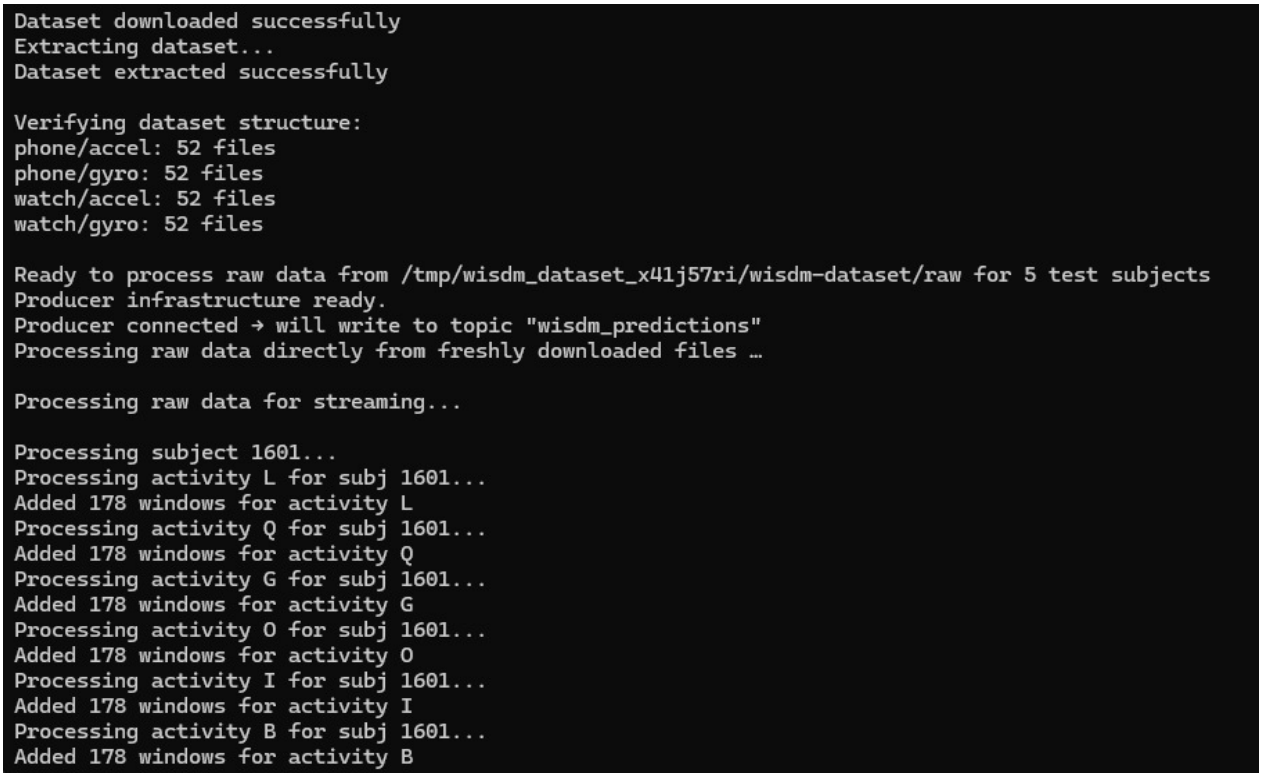

Figure 9: Output Display after running producer in virtual environment with Kafka Integration

```
  Activity Q: 178 windows
  Activity R: 178 windows
  Activity S: 178 windows

Subject 1647:
  Activity A: 178 windows
  Activity B: 178 windows
  Activity C: 178 windows
  Activity D: 178 windows
  Activity E: 178 windows
  Activity F: 178 windows
  Activity G: 178 windows
  Activity H: 178 windows
  Activity I: 178 windows
  Activity J: 178 windows
  Activity K: 178 windows
  Activity L: 178 windows
  Activity M: 178 windows
  Activity O: 178 windows
  Activity P: 178 windows
  Activity Q: 178 windows
  Activity R: 178 windows
  Activity S: 178 windows

Total messages sent: 24332
All data processed and sent successfully.
Temporary data files will be removed automatically.
Kafka producer stopped.
Spark session stopped.
(venv) hashmmath@DESKTOP-601LEHH:~/CSGY-6513-big-data-project$
```

Figure 10: Completion of Producer program after transferring the data to app using Kafka

```
hashmmath@DESKTOP-601LEHH:~$ python3 -m venv venv
hashmmath@DESKTOP-601LEHH:~$ source venv/bin/activate
(venv) hashmmath@DESKTOP-601LEHH:~$ cd ~/CSGY-6513-big-data-project
(venv) hashmmath@DESKTOP-601LEHH:~/CSGY-6513-big-data-project$ git pull origin main
From https://github.com/kushagrayadv/CSGY-6513-big-data-project
 * branch            main       -> FETCH_HEAD
Already up to date.
(venv) hashmmath@DESKTOP-601LEHH:~/CSGY-6513-big-data-project$ streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://172.27.209.34:8501

gio: http://localhost:8501: Operation not supported
Using device: cpu
Loading model from cnn_bilstm_classifier_final.pt
Using device: cpu
Loading model from cnn_bilstm_classifier_final.pt
Loaded 18 activity classes
Loaded 18 activity classes
Model loaded successfully
Model loaded successfully
Consumer started. Listening on topic: wisdm_predictions
Waiting for messages...
Consumer started. Listening on topic: wisdm_predictions
Waiting for messages...
Using device: cpu
Loading model from cnn_bilstm_classifier_final.pt
Loaded 18 activity classes
```

Figure 11: Using Streamlit for running application and waiting to collect data from producer using Kafka

```
Predicted → S (Folding Clothes) (conf: 0.93)
Ground Truth: S (Folding Clothes)
Correct: ✓
Accuracy: 43.75% (10643/24329)
========================================

========================================
Subject: 1647, Device: watch
Predicted → S (Folding Clothes) (conf: 0.97)
Ground Truth: S (Folding Clothes)
Correct: ✓
Accuracy: 43.75% (10644/24330)
========================================

========================================
Subject: 1647, Device: watch
Predicted → S (Folding Clothes) (conf: 0.93)
Ground Truth: S (Folding Clothes)
Correct: ✓
Accuracy: 43.75% (10645/24331)
========================================

========================================
Subject: 1647, Device: watch
Predicted → S (Folding Clothes) (conf: 0.76)
Ground Truth: S (Folding Clothes)
Correct: ✓
Accuracy: 43.75% (10646/24332)
========================================
```

Figure 12: Showing the way the data is displaying in the terminal after fetching it from producer using Kafka and transferring it to display on a Web-Page

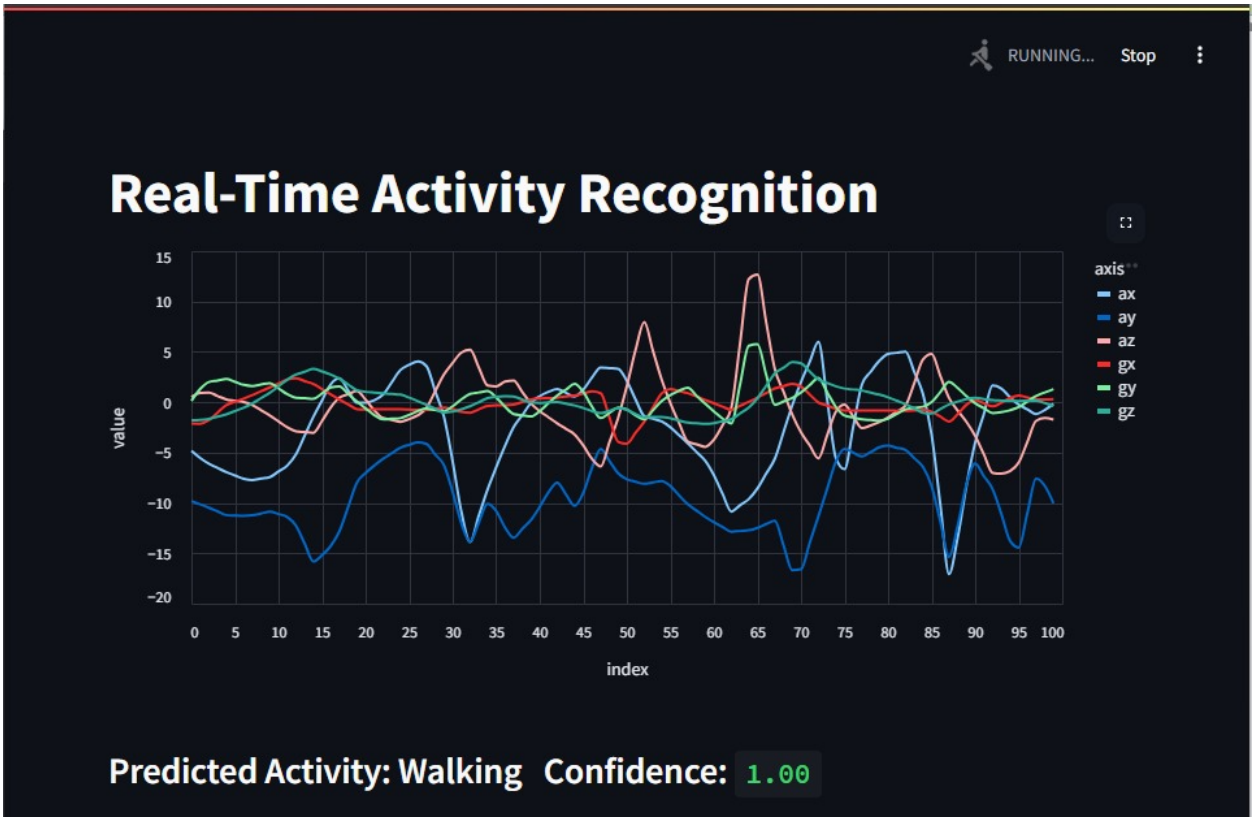

Figure 13: Final Screenshot of Activity Prediction from the fetched data and displaying it on the webpage using Kafka and Streamlit

Conclusion

This project presented a complete pipeline for human activity recognition (HAR) using the WISDM dataset, combining scalable **Spark-based preprocessing** with a custom **deep learning architecture**. The sensor data from smartphone and smartwatch accelerometers and gyroscopes were cleaned, interpolated, synchronized, and standardized using distributed Spark transformations, producing a consistent feature space suitable for machine learning.

The model at the heart of this pipeline was a **hybrid Convolutional Neural Network (CNN) followed by a Bidirectional Long Short-Term Memory (BiLSTM) network with an attention mechanism**. This architecture enabled the system to capture both local temporal patterns and global contextual dependencies within sensor windows. Training results showed high accuracy for frequent locomotion activities like walking and jogging, but weaker performance on underrepresented actions like sitting or clapping, highlighting the need for better class balance.

Comprehensive visual analyses including feature distributions, correlation heatmaps, and per-class metrics validated both data integrity and model behavior. In conclusion, the pipeline proves effective for large-scale HAR tasks, demonstrating strong potential for real-time streaming applications. Future directions include addressing class imbalance, optimizing inference on edge devices, and refining the model for minority-class recall without compromising latency.

YOU CAN ACCESS THE CODE-BASE BY CLICKING THE LINK BELOW OR ELSE YOU CAN DOWNLOAD IT FROM THE ZIP FILE AS WELL. Thank You.

https://github.com/kushagrayadv/CSGY-6513-big-data-project

References

[1] Chandramouli, N.A., Natarajan, S., Alharbi, A.H. *et al.* (2024) 'Enhanced human activity recognition in medical emergencies using a hybrid deep CNN and bi-directional LSTM model with wearable sensors', *Scientific Reports*, 14, 30979.

[2] Yin, J., Li, D., Wu, Z. *et al.* (2025) 'HARCNN: self-attention augmented convolution for robust human activity recognition', *IEEE Internet of Things Journal*, 12(4), pp. 3112–3124.

[3] Freedman, R., Patel, M. and Goldsmith, J. (2025) 'A large dataset of wrist-worn activity tracker data collected in the wild', *PLOS ONE*, 20(3), e0287654.

[4] Khatun, M.A., Islam, M.R. and Dey, R.K. (2024) 'Achieving more with less: a lightweight deep learning solution for wearable human activity recognition', *Sensors*, 24(16), 5436.

[5] Song, Q., Huang, Y. and Chen, L. (2022) 'Human activity recognition based on residual network and BiLSTM', *Sensors*, 22(2), 635.

[6] Rahman, M., Hasan, K. and Abedin, K. (2024) 'WISNet: a deep neural network for sensor-based activity recognition', *Expert Systems with Applications*, 235, 121414.

[7] Hernández, J., Kim, T. and Liu, S.H.E. (2023) 'A new CNN-LSTM architecture for activity recognition employing wearable sensors', *Engineering Applications of Artificial Intelligence*, 132, 106959.

[8] Li, X., Zhou, J. and Zhang, H. (2024) 'Human activity recognition using a multi-branched CNN-BiLSTM with attention', *Applied Soft Computing*, 150, 110075.

[9] Zhang, K., Li, T. and Wang, S. (2023) 'An improved human activity recognition technique based on multi-layer CNN on WISDM', *Scientific Reports*, 13, 49739.

[10] Al-Qaysi, E., Mahgoub, H. and Senousy, G.H. (2023) 'A DCNN-LSTM based human activity recognition by mobile and wearable sensors', *Ain Shams Engineering Journal*, 14(6), 101052.

[11] Zhao, L., Chen, P. and Guo, Y. (2024) 'A novel CNN-BiLSTM-GRU hybrid deep learning model for multi-modal human activity recognition', *Machine Learning with Applications*, 18, 100469.

[12] Zhang, Y., Liu, F. and Chen, D. (2025) 'Research on the application of Spark streaming real-time data analysis enhanced by agent AI with LangGraph', *arXiv preprint* arXiv:2501.14734.