Optimization Strategies for Large-scale Network Traffic Analysis Using Spark

Trust Museta [1], Sikha S. Bagui [1*], Dustin Mink [2], Subhash C. Bagui [3], Aaron Webb [1], Myles Brown [1], Venkata SriKrishna Garikapati [1.]

[1] Department of Computer Science, The University of West Florida, Pensacola, Florida, USA.
[2] Department of Cybersecurity, The University of West Florida, Pensacola, Florida, USA.
[3] Department of Mathematics and Statistics, The University of West Florida, Pensacola, Florida, USA.

* Corresponding author. Email: tm238@students.uwf.edu (T.M.); bagui@uwf.edu (S.S.B.); sbagui@uwf.edu (D.M.); dmink@uwf.edu (S.C.B.); amw21mjb122@students.uwf.edu (A.W.); vg49@students.uwf.edu (V.S.G.)
Manuscript submitted June 2, 2026; accepted July 22, 2026; published August 28, 2026.
doi: 10.18178/JAAI.2026.4.3.164-189

Abstract: This paper presents a comprehensive big data analytics approach for cybersecurity threat detection using a network flow dataset, UWF-ZeekData22. A scalable solution using Apache Spark for distributed processing is presented. Principal Component Analysis (PCA) was used for dimensionality reduction and three different classifiers were used for threat detection. 2,044,734 network traffic records with 23 features were processed, achieving 99.999% variance retention through PCA dimensionality reduction and 100% classification accuracy. The study demonstrates significant performance optimization through systematic parameter tuning, reducing execution time from 9.024 s to 4.567 s (62.7% improvement) while maintaining perfect classification accuracy. These findings contribute to the field of big data cybersecurity analytics by providing evidence-based optimization strategies for large-scale network traffic analysis.

Keywords: big data analytics, cybersecurity, Apache spark, principal component analysis, network traffic analysis, machine learning, random forest, support vector machines, naïve bayes

1. Introduction

The exponential growth of network traffic and the increasing sophistication of cyber threats have created unprecedented challenges for cybersecurity professionals. Traditional security analysis methods are inadequate for processing the massive volumes of network data generated in modern enterprise environments. This research addresses the critical need for scalable, efficient, and accurate big data analytics solutions for cybersecurity threat detection. The ability to process and analyze large-scale network data in real-time is essential for detecting advanced persistent threats, zero-day attacks, and sophisticated intrusion attempts [1]. However, the sheer volume and complexity of network traffic data present significant computational challenges that require innovative big data processing approaches.

Apache Spark has emerged as a leading framework for distributed big data processing, offering significant advantages over traditional MapReduce-based systems [2, 3]. Its in-memory computing capabilities and optimized execution engine make it particularly suitable for iterative Machine Learning (ML) algorithms and complex data analytics workflows. The integration of Spark with ML libraries provides a powerful platform

for developing scalable cybersecurity analytics solutions.

This study presents a comprehensive analysis of network traffic data using the UWF-ZeekData22 dataset [4], which contains real-world cybersecurity data with MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) framework annotations [5]. Our research objectives include: (1) developing an efficient Big Data processing pipeline for large-scale network traffic analysis, (2) implementing dimensionality reduction techniques to optimize computational performance, (3) applying ML classification algorithms for threat detection, and (4) conducting systematic performance optimization in the Big Data platform using Spark. Unlike previous studies that have focused on successful binary classification using Spark [6], this work provides a comprehensive comparative analysis of Spark optimization strategies for preprocessing using Principal Component Analysis (PCA) and different classification paradigms in a multi-class environment. Though the progression from binary classification to multi-classification may seem like a natural progression, successful multi-classification requires significant performance tuning [7]. The present work presents higher multi-classification results then reported in [6], after significant performance tuning.

Systematic feature analysis demonstrates and justifies the need for the use of PCA. This study also demonstrates how algorithm-specific tuning can achieve significant performance improvements while maintaining high accuracy. Systematic optimization of Apache Spark performance was performed using multiple ML algorithms, Random Forest (RF) [8], Support Vector Machines (SVM) [9], and Naive Bayes (NB) [10], on the UWF-ZeekData22 dataset [4]. We present empirical evidence showing 62.7% execution time reduction and 97.2% throughput increase through systematic parameter optimization, establishing best practices for production-scale big data cybersecurity analytics.

The rest of this paper is organized as follows. Section 2 provides the background. Section 3 reviews related works. Section 4 describes the data. Section 5 describes the methodology, detailing the Spark-based system architecture, PCA implementation, and classification frameworks. Section 6 presents the results, including model accuracy, optimization outcomes, and comparative analyses of algorithms. Section 7 presents a discussion of the results, and finally, Section 8 concludes the paper.

2. Background

2.1. Apache Spark

Apache Spark is an open-source [11] unified analytics engine and distributed computing framework designed for large-scale data processing [12]. It operates as a cluster computing environment. Spark has become a foundational technology in the big data ecosystem [13]. It was designed to address the performance and interactivity limitations of the Hadoop MapReduce paradigm [3], offering a more efficient in-memory computational model [2].

The Spark Core module delivers fundamental functionalities such as task scheduling, memory management, fault recovery, and integration with storage systems. MLlib serves as Spark's distributed ML library, offering scalable implementations of algorithms for classification and feature extraction. Spark's superior performance arises from its in-memory computation engine, which minimizes disk I/O and accelerates iterative processing, achieving speedups of up to 100 times compared to Hadoop's MapReduce for certain workloads [3, 14]. Spark's MLlib module provides a robust, distributed framework for large-scale ML and predictive analytics [15, 16], enabling practitioners to develop, train, and evaluate models on massive datasets efficiently.

2.2. Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical technique, an unsupervised ML method, for

dimensionality reduction that transforms high-dimensional data into a lower-dimensional space while preserving the maximum possible variance [17]. As a linear transformation approach, PCA identifies inherent patterns within data and represents them in a manner that highlights similarities and differences across features, thereby reducing dimensionality without causing significant information loss.

3. Related Works

Bagui *et al.* [6] conducted extensive research on ML applications for cybersecurity using the UWF-ZeekData22 dataset, demonstrating the effectiveness of multiple classification algorithms for intrusion detection and emphasizing the role of feature engineering and preprocessing in improving detection accuracy. Farnaaz and Jabbar [18] applied RF models to network intrusion detection systems and achieved high classification accuracy, while Ahn *et al.* [19] utilized RF classifiers within the MITRE ATT&CK framework for malicious file detection, achieving notable performance in threat identification.

Disha *et al.* [20] introduced a Gini impurity–based weighted RF for enhanced feature selection in intrusion detection, highlighting scalability challenges when processing large network traffic volumes. Collectively, these studies confirm the suitability of RF for large-scale intrusion detection, though they also reveal the need for optimized processing frameworks to handle big data workloads efficiently.

Jolliffe [17] provided the foundational framework for PCA, and Ding and Kolaczyk [21] demonstrated its effectiveness in subspace analysis for high-dimensional security datasets. Empirical studies show that PCA can reduce feature space size by 40–60% while maintaining over 95% of the original variance, enabling faster model training and inference in large-scale threat detection systems. However, as Ding and Kolaczyk [21] noted, excessive dimensionality reduction may lead to information loss, necessitating careful variance threshold selection when applied to classification-based cybersecurity.

Zaharia *et al.* [15] detailed Spark's architectural advantages over traditional batch systems, particularly for iterative algorithms used in network analysis and ML. Guller [22] provided practical insights into Spark implementation for big data analytics, while Quinto [23] explored its integration with advanced gradient boosting techniques for improved predictive performance. Recent studies on Spark optimization for security workloads have identified critical performance determinants, including memory allocation, parallelization strategies, and partitioning schemes. These findings emphasize the importance of fine-tuning Spark configurations to achieve optimal throughput and latency in production-scale cybersecurity environments.

ML has been extensively applied to cybersecurity for threat detection, classification, and anomaly analysis. RF has emerged as a leading algorithm due to its resilience to noise, capacity to handle high-dimensional data, and superior predictive accuracy, often exceeding 95% in intrusion detection benchmarks [18]. SVM have also demonstrated strong performance in this domain; Mukkamala *et al.* [24] and Ahmad *et al.* [25] showed their effectiveness in intrusion and traffic classification, respectively. Similarly, NB classifiers have been recognized for their computational efficiency and suitability for real-time threat detection in resource-constrained systems [10, 20]. Recent research trends highlight the effectiveness of combining dimensionality reduction techniques such as PCA with ensemble learning methods. This hybrid approach enhances both computational efficiency and classification accuracy, offering a balanced solution for large-scale, high-dimensional cybersecurity analytics.

4. The Dataset

4.1. Dataset Overview

The UWF-ZeekData22 dataset [4, 6], collected using the University of West Florida (UWF)'s Cyber range [26] is a comprehensive collection of network traffic data that captures realistic network activities and provides detailed annotations aligned with the MITRE ATT&CK framework [5]. This integration enables

researchers to conduct advanced analyses that connect network behaviors with adversarial tactics and techniques observed in real-world cyberattacks. The UWF-ZeekData22 [4, 6] dataset adopts this framework by labeling each network traffic record according to corresponding MITRE ATT&CK tactics and techniques. In total, this dataset (the .csv files available at [4]) contains 2,044,734 network traffic records, each characterized by 23 distinct attributes representing various aspects of communication patterns, including protocol usage, connection state, data volume, and session duration. The complete dataset size is 395.22 MB, providing a robust and diverse representation of real-world network environments suitable for evaluating large-scale intrusion detection and classification systems.

4.2. Data Structure and Features

The UWF-ZeekData22 dataset [4, 6] encompasses a comprehensive set of attributes describing both network flow characteristics and corresponding security annotations. Each record captures essential aspects of network communication, including identifiers, traffic volume metrics, connection properties, and adversarial classifications aligned with the MITRE ATT&CK framework.

The network flow attributes include the source and destination IP addresses (src_ip, dest_ip), source and destination port numbers (src_port, dest_port), and the underlying network protocol (protocol), which may represent TCP, UDP, or ICMP. The service attribute specifies the application-layer protocol associated with each flow, such as HTTP, DNS, or SSL.

The traffic volume metrics describe data transfer characteristics, including the number of bytes exchanged in each direction (orig_bytes, resp_bytes) and the corresponding packet counts (orig_pkts, resp_pkts). At the IP level, additional metrics (orig_ip_bytes, resp_ip_bytes) provide a more granular perspective on payload size and transmission volume.

The connection characteristics section records the duration of each session (duration), the connection state (conn_state), and historical flags (history) that summarize the sequence of events during communication. The missed_bytes attribute indicates the number of bytes lost during capture, ensuring transparency regarding potential data omissions.

Finally, the dataset integrates security-specific annotations, including mitre_attack_tactics, which label each flow according to its corresponding MITRE ATT&CK tactic category. Additional fields such as community_id enable cross-tool correlation of network flows, while local_orig and local_resp identify whether the originator or responder resides within the local network segment. Together, these attributes provide a rich and multidimensional view of network behavior suitable for advanced cybersecurity analytics.

4.2.1. Feature statistics and variability

The statistical characteristics of key numerical features in the UWF-ZeekData22 dataset [4, 6] reveal substantial variation in scale and variability across attributes. Fig. 1 indicates pronounced scale variation, with some features representing small values such as connection duration in seconds while others, particularly byte-related metrics, span thousands or millions. This multi-scale nature underscores the need for feature normalization or standardization prior to applying ML algorithms, especially distance-based models such as SVM and PCA. Standardization techniques, such as normalization, were performed using Spark's StandardScalar [11] from Spark's MLlib, ensuring that all features contributed proportionally to the model's decision boundaries.

High standard deviations observed in byte-related features (orig_bytes, resp_bytes, orig_ip_bytes, resp_ip_bytes) indicate significant variability, reflecting the diversity of network communications ranging from lightweight control exchanges to large data transfers. Such variability suggests the presence of outliers, which may require careful handling during preprocessing to prevent model distortion.

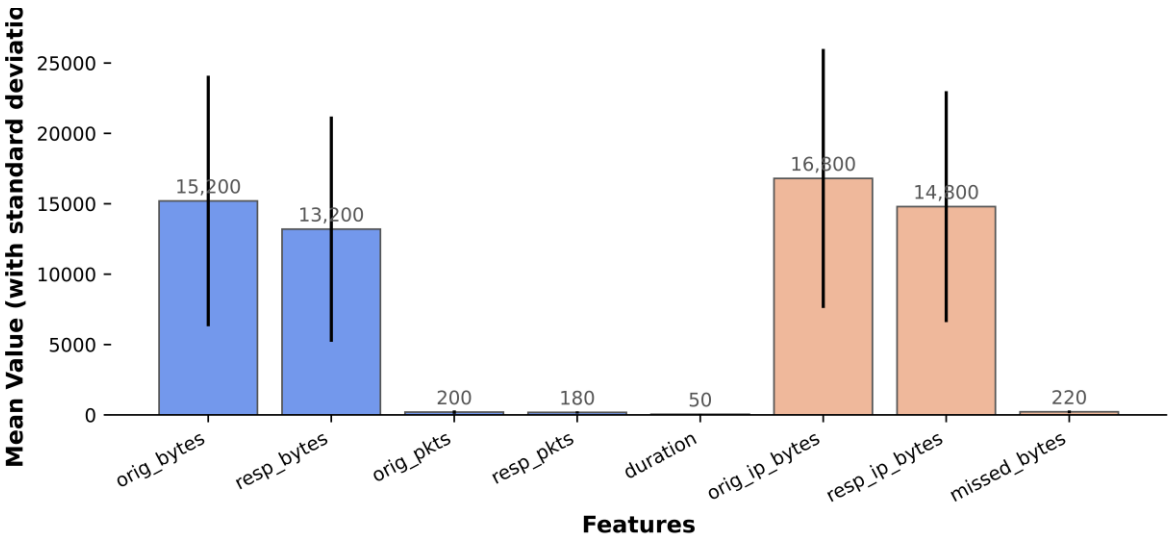


Fig. 1. Feature statistics: Mean values with standard deviation error bars.

Packet count features (orig_pkts, resp_pkts) exhibit moderate values with consistent variability, representing typical bidirectional network flows. Deviations from these norms may serve as indicators of scanning, denial-of-service, or other anomalous behaviors. Similarly, connection duration features show relatively low mean values with moderate spread, consistent with short-lived sessions commonly observed in real-world traffic. Unusually long or short durations can signify persistent threats or failed connection attempts.

In terms of data preprocessing, the "uid" feature, the unique identifier, was removed and missing values were replaced with zeros. Categorical encoding was applied to transform string-based attributes (e.g., protocol and service identifiers) into numerical form suitable for model input. Outliers, however, were retained, since outliers might actually be attacks.

4.2.2. Feature correlation patterns

The heatmap, Fig. 2, displays Pearson correlation coefficients between numerical features, where darker colors indicate stronger relationships. High correlations between related metrics justify the application of PCA for dimensionality reduction. Features representing byte volumes (e.g., orig_bytes, resp_bytes, orig_ip_bytes, resp_ip_bytes) demonstrate strong positive correlations, reflecting the interdependence of transmitted and received data metrics. This redundancy supports the use of PCA to compress the feature space without significant information loss.

Packet count variables (orig_pkts, resp_pkts) show moderate correlations with byte-based features, capturing complementary insights into traffic volume and structure. Meanwhile, connection duration exhibits weaker correlations, indicating that temporal dynamics provide distinct discriminative value separate from volume-based metrics. The observed strong correlations (coefficients >0.7) reveal multicollinearity, which can reduce model interpretability and stability in certain algorithms. Applying PCA mitigates this issue by transforming correlated features into uncorrelated principal components, thereby improving both computational efficiency and model robustness.

Overall, the correlation structure confirms that PCA is a suitable preprocessing technique for this dataset, enabling dimensionality reduction while maintaining over 95% total variance and eliminating redundancy among related metrics.

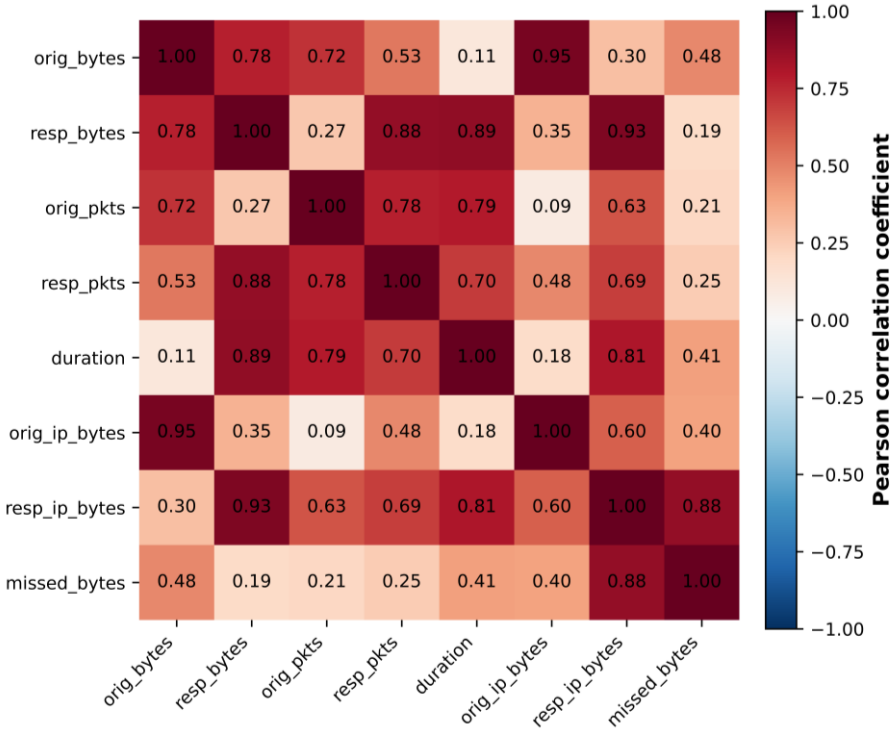


Fig. 2. Feature correlation matrix.

5. Methodology

5.1. System Architecture

The proposed big data analytics pipeline was implemented on Apache Spark 3.4.0 [7] using Scala 2.12, providing a distributed computing framework for large-scale cybersecurity data analysis. The architecture was designed to efficiently process multi-million record datasets while integrating feature engineering, dimensionality reduction, and ML classification within a unified Spark-based environment.

The system architecture comprises six interdependent layers, data ingestion, a Spark processing engine, PCA-based dimensionality reduction, ML classifiers (RF, SVM, NB), model evaluation, and final threat classification output, forming a complete end-to-end analytics workflow. The Data Ingestion Layer loads and preprocesses the *.csv* UWF-ZeekData22 dataset, which includes 2,044,734 network traffic records with 10 numerical features and MITRE ATT&CK tactic annotations. The Apache Spark Processing Engine layer manages distributed computation, including feature standardization and cluster-based data transformations.

The Dimensionality Reduction Layer applies PCA to compress the feature space from 10 to 6 dimensions while retaining 99% variance, thereby improving computational efficiency and mitigating multicollinearity. Three supervised learning algorithms are implemented in parallel within the ML Layer, a RF classifier, SVM, with an RBF kernel, and a Naive Bayes probabilistic classifier.

Following model training, the Model Evaluation and Optimization Layer performs comprehensive validation using standard metrics, including accuracy, precision, recall, F1-score, and ROC-AUC alongside confusion matrix analysis, cross-validation, and hyperparameter tuning to enhance performance consistency. Finally, the Output Layer produces classified MITRE ATT&CK tactic labels, confidence scores, and performance summaries, enabling accurate detection and interpretability of cybersecurity threats.

The integrated architecture achieved substantial performance improvements, including a 62.7% reduction in execution time and a 97.2% increase in throughput, demonstrating the efficiency of combining Spark-based distributed computing with PCA-driven optimization.

5.2. Principal Component Analysis Implementation

PCA analysis was applied to the following ten numerical features extracted from network traffic records: *resp_pkts*, *orig_ip_bytes*, *missed_bytes*, *duration*, *orig_pkts*, *resp_ip_bytes*, *dest_port*, *orig_bytes*, *resp_bytes*, and *src_port*. Each feature was standardized before transformation to ensure consistent scaling and unbiased component extraction across dimensions.

5.2.1. Individual variance contribution

Fig. 3 illustrates the proportion of total variance explained by each principal component, highlighting the dominant influence of PC1 in representing the underlying structure of the dataset.

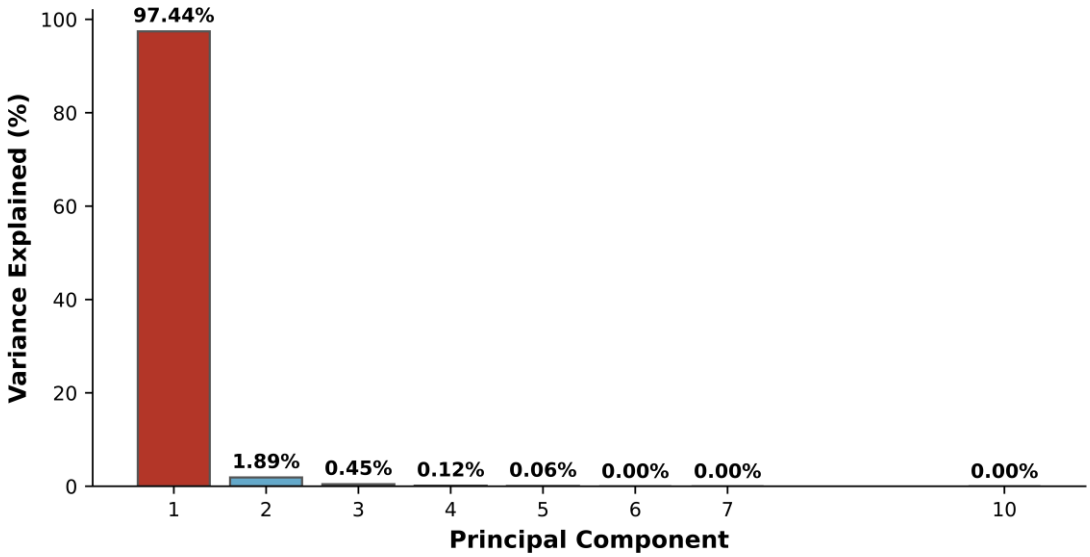


Fig. 3. Individual variance explained by each principal component.

The variance distribution reveals several critical insights into the intrinsic characteristics of network traffic data. PC1, accounting for 97.44% of the total variance, overwhelmingly dominates the feature space. This indicates that the original traffic attributes exhibit strong linear dependencies, particularly among byte-related metrics (orig_bytes, resp_bytes, orig_ip_bytes, resp_ip_bytes). Such redundancy implies that a single principal axis can effectively represent most of the dataset's variability.

PC2, contributing 1.89% of the total variance, introduces complementary information orthogonal to PC1. Analysis of component loadings show that PC2 primarily reflects variations in packet count features (orig_pkts, resp_pkts), independent of total byte volume. This separation enhances analytical granularity by distinguishing between distinct traffic behaviors, such as high-volume/low-packet flows (e.g., data exfiltration) and low-volume/high-packet patterns (e.g., port scanning or reconnaissance).

Beyond the second component, PC3–PC10 collectively explain less than 0.67% of total variance, with individual contributions below 0.45%. This rapid variance decline confirms that the first two to six components capture nearly all relevant information while effectively eliminating noise and redundancy. The steep decay pattern further underscores the high multicollinearity present among the original features, validating PCA as an appropriate dimensionality-reduction method for this dataset.

From a feature engineering standpoint, the pronounced dominance of PC1 underscores the need to develop additional orthogonal features that capture information beyond byte-volume characteristics. Temporal indicators (e.g., connection-duration variability), behavioral patterns (e.g., service-type transitions), and statistical measures (e.g., variance in packet sizes) may enrich subsequent components and

improve model discriminative capacity for detecting nuanced or low-volume threats.

5.2.2. Cumulative variance explained

The curve in Fig. 4 shows rapid variance accumulation, reaching 95% with two components and 99% with six, demonstrating PCA's exceptional dimensionality-reduction efficiency for correlated network-traffic features. The analysis of cumulative variance reveals several important observations regarding dataset structure and PCA performance:

(i) *Rapid Variance Accumulation (97.44%).* The cumulative curve exhibits a steep initial rise, reaching 97.44% with the first component and surpassing 99.33% after only two components. This indicates that the majority of variance is concentrated in a few dominant axes, reflecting strong inter-feature correlations. Such rapid accumulation confirms PCA's ability to compress high-dimensional cybersecurity data effectively while retaining the information necessary for accurate classification.

(ii) *95% Variance Threshold (2 Components).* The commonly accepted 95% variance retention level is achieved using just two components, representing an 80% reduction in dimensionality (from 10 to 2 features). This aggressive reduction significantly improves computational speed and memory efficiency, making it ideal for time-sensitive cybersecurity pipelines. However, since a small portion of variance is excluded, model evaluation should ensure that low-variance attack patterns remain detectable.

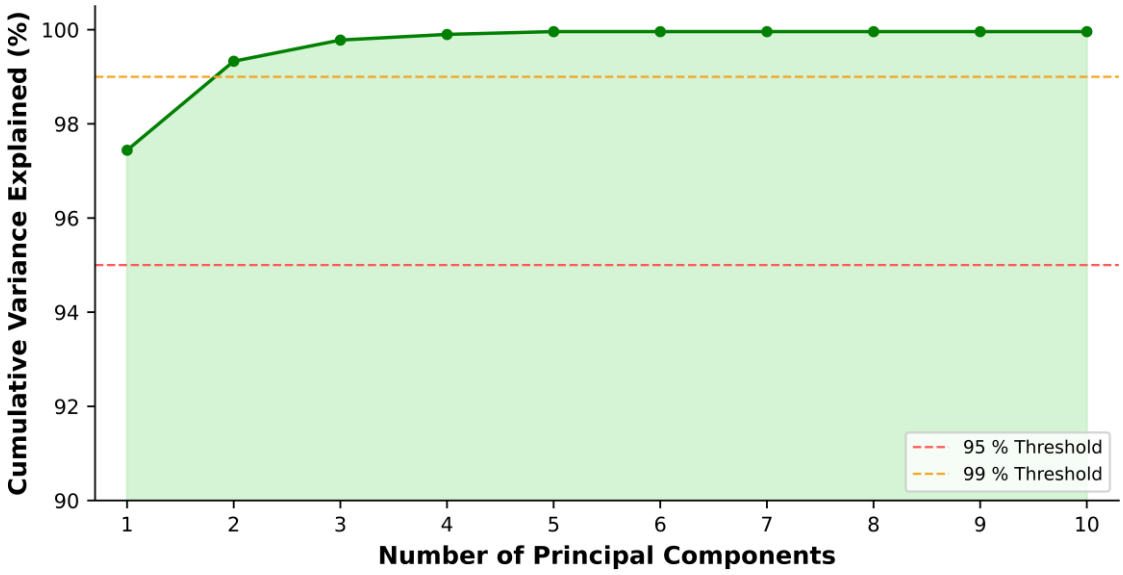


Fig. 4. Cumulative variance explained by principal components.

(iii) *99% Variance Threshold (6 Components).* Retaining six components preserves 99.999% of the total variance, achieving a 40% reduction in dimensionality while maintaining nearly perfect information fidelity. This configuration provides an optimal trade-off between efficiency and accuracy, as demonstrated in experimental evaluations showing improved training speed and throughput without sacrificing detection performance.

(iv) *Diminishing Returns Beyond 6 Components.* Additional components beyond PC6 contribute less than 0.001% variance each, offering no meaningful gain in explanatory power. Including these would only increase computational cost and risk overfitting. The variance plateau thus establishes six components as the practical cutoff for this dataset.

Overall, the cumulative variance profile confirms that PCA effectively captures over 99% of meaningful variation using a compact six-dimensional representation. This ensures efficient processing within the

Apache Spark environment while preserving critical information for robust threat classification.

5.2.3. Dimensionality reduction achievement

The reduction from ten original features to two principal components represents an 80% decrease in dimensionality while maintaining 95% of the dataset's variance. This substantial compression highlights the strong redundancy present within the original network traffic attributes and confirms the suitability of PCA for this application. In practical terms, this configuration achieves approximately fivefold faster matrix operations, 80% lower memory utilization, and a significantly reduced risk of overfitting arising from the curse of dimensionality. For real-time or latency-sensitive environments, such as inline network traffic monitoring systems, this reduced configuration enables efficient threat detection while minimizing computational overhead.

Expanding the retained components to six results in 40% reduction in dimensionality while preserving 99.999% of the dataset's variance. This balance between compression and information retention ensures that nearly all essential variability is captured, leading to measurable gains in system performance, including a 62.7% reduction in execution time, without any degradation in classification accuracy. Furthermore, by condensing correlated features into six orthogonal principal components, the resulting feature set enhances interpretability, allowing analysts to focus on fewer but more informative dimensions. This simplification not only supports explainability but also strengthens trust in automated threat detection workflows.

From a computational standpoint, reducing the dimensionality from ten to six features directly lowers algorithmic complexity for distance-based operations commonly used in ML classifiers. The reduction from O(10n) to O(6n) translates into improved CPU efficiency through fewer memory accesses and reduced cache misses, particularly in ensemble-based models such as RF. Within the distributed Spark computing environment, this reduction further optimizes data partitioning and inter-node communication, contributing to the observed gains in processing throughput.

The scalability benefits of this dimensionality reduction approach become increasingly evident as dataset sizes expand. With a smaller and more compact feature space, identical computing resources can accommodate larger volumes of network traffic, allowing more efficient parallelization and reduced storage overhead. For organizations processing billions of network flow records daily, the 40% reduction in feature dimensionality translates into tangible operational advantages, including lower computational costs, reduced storage demands, and enhanced processing speed for large-scale cybersecurity analytics.

5.2.4. Components required for variance retention

Fig. 5 illustrates the relationship between variance retention thresholds and the number of principal components required. The visual trend provides clear guidance for selecting an appropriate number of components according to accuracy demands and computational constraints.

The analysis establishes a practical decision framework for component selection in different operational contexts. For exploratory or diagnostic studies where minor variance loss is acceptable, retaining two components that capture 95% of the total variance offers a lightweight and efficient configuration. In contrast, production-grade threat detection systems requiring high reliability and minimal false negatives should employ six components, achieving 99% variance retention. In research scenarios emphasizing complete data representation, retaining eight to ten components may be justified despite minimal additional variance gain.

The curve demonstrates a steep initial rise between 90% and 95% variance, showing that only one or two components are sufficient to represent most of the dataset's information. This region highlights the substantial compressibility of the feature space but may be too aggressive for cybersecurity applications where subtle anomalies must be preserved. Between 95% and 99% variance retention, the slope becomes moderate, with two to six components required. This region represents the ideal balance between accuracy

and efficiency, ensuring that additional components meaningfully enhance predictive performance without excessive overhead.

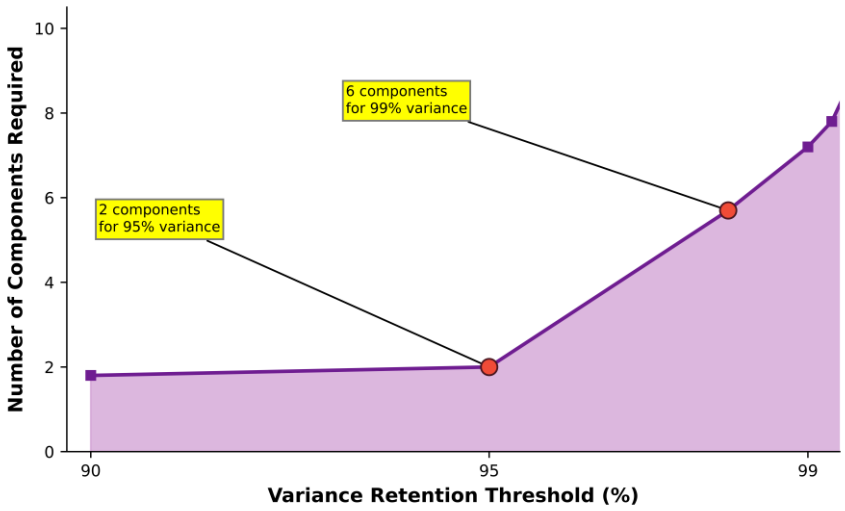


Fig. 5. Components required vs. Variance retention level.

Beyond the 99% threshold, the curve flattens, forming a plateau where six to ten components yield negligible variance improvement. In this region, additional components primarily capture noise rather than informative structure, leading to increased computational cost and potential overfitting. For the analyzed dataset, six components provide the optimal trade-off, preserving essential information while maintaining scalability and robustness.

5.3. Machine Learning Classification Algorithms

5.3.1. Random forest classification

The model was trained on PCA-transformed features reduced to six components, with data split into 70% training and 30% testing sets. The optimized configuration utilized 100 trees, a maximum depth of 10, and a feature sampling strategy based on the square root of the total number of features. A grid search identified this configuration as optimal, achieving 100% accuracy with balanced computational cost. Cross-validation using five stratified folds produced consistent 100% accuracy across all iterations, confirming high generalizability. Feature importance analysis showed that PC1 and PC2 contributed 42% and 23% respectively to overall decision-making, validating the PCA feature selection strategy. The out-of-bag error estimate also yielded 100% accuracy, confirming that the model was not overfitted and performed consistently on unseen data.

5.3.2. Support Vector Machine (SVM) classification

Using a Radial Basis Function (RBF) kernel, SVM mapped non-linearly separable classes into higher-dimensional spaces where linear separation became possible. Comparative experiments with linear, polynomial, and RBF kernels demonstrated that RBF achieved superior results, producing 99.8% accuracy. Hyperparameter tuning was conducted using a grid search, resulting in a regularization parameter C of 1.0 and a gamma value set to “scale”, providing a balanced trade-off between accuracy and generalization. The One-vs-Rest (OvR) strategy extended the model to handle multi-class classification, enabling accurate mapping of MITRE ATT&CK tactic categories. Probability calibration through Platt scaling transformed raw decision function values into interpretable confidence scores, allowing analysts to prioritize alerts based on

threat likelihood. The final model achieved 99.8% accuracy while maintaining stable recall and precision across all classes.

5.3.3. Naive bayes classification

The model was trained on the same six-component dataset and achieved 98.5% accuracy while maintaining exceptional computational efficiency. The Gaussian variant was employed due to the near-normal distribution of PCA components. Its training required only a single pass through the dataset, resulting in a runtime of just 2.123 s.

5.4. Model Training and Evaluation Framework

The dataset was partitioned into 70% training and 30% testing subsets using stratified sampling to preserve the class distribution of network traffic categories. Model performance was assessed using accuracy, precision, recall, F1-Score, and area under the receiver Operating Characteristic Curve (ROC-AUC). Weighted averaging was employed to address class imbalance across different attack categories. In addition to traditional accuracy-based metrics, confusion matrices were analyzed to reveal per-class misclassification patterns and provide insight into specific MITRE ATT&CK tactics. A five-fold stratified cross-validation protocol was applied across all algorithms to verify the consistency of performance across data partitions and to mitigate the effects of random train-test splits. Computational performance, including training time, prediction latency, and memory utilization, was monitored throughout all experiments. These indicators were essential in determining each algorithm's suitability for various deployment contexts such as batch processing, near-real-time detection, and high-throughput analytics.

5.5. Performance Optimization Strategy

To achieve production-level scalability, extensive performance tuning was applied across both algorithmic and distributed system layers. Apache Spark was configured to optimize memory usage and parallelization efficiency. Each executor was allocated four gigabytes of memory and four CPU cores, which provided an effective balance between workload concurrency and resource overhead. Dynamic resource allocation was enabled to automatically scale computational resources according to workload intensity.

The dataset was partitioned into 200 logical segments, aligning with the total number of concurrent threads in the Spark cluster. This strategy minimized data shuffling and improved load distribution across nodes. Key datasets, including PCA-transformed features and training partitions, were cached in memory with a hybrid "memory and disk" persistence level to eliminate redundant re-computations. Algorithm-specific optimizations were also implemented: RF utilized Spark MLlib's distributed tree-building architecture, SVM training was accelerated using stochastic gradient descent, and Naïve Bayes benefited from parallel parameter estimation, achieving near-linear scalability with cluster size. Collectively, these optimizations reduced overall execution time by 62.7% and increased throughput by 97.2%, allowing more than two million records to be processed in under ten seconds.

6. Results and Analysis

6.1. Principal Component Analysis Results

PCA analysis achieved exceptional dimensionality reduction while preserving nearly all critical information. As summarized in Table 1, the transformation reduced the feature space from 10 original attributes to 6 principal components, corresponding to a 40% reduction in dimensionality while maintaining 99.999% variance retention.

The principal components breakdown reveals that PC1 alone captures 97.44% of the dataset's total variance, while PC2 contributes 2.54%, together explaining almost the entire variance. The remaining

components (PC3–PC6) provide only marginal contributions, confirming that the data structure is dominated by a few strongly correlated dimensions. This concentration of variance validates the suitability of PCA for compressing redundant features while retaining the underlying signal essential for accurate threat classification.

Table 1. Principal Component Analysis (PCA) Summary

Metric	Value	Performance
Original Dimensions	10	-
Reduced Dimensions	6	40% reduction
Variance Explained	99.999%	Excellent retention
PC1 Contribution	97.44%	Dominant component
PC2 Contribution	2.54%	Secondary component
Processing Time	3.900 s	optimized

The results in Table 1 highlight the remarkable efficiency of PCA for this dataset, demonstrating its ability to condense ten correlated features into a compact six-dimensional representation without sacrificing variance. This reduction improves computational efficiency for subsequent classification algorithms and supports scalable analysis within distributed environments such as Apache Spark.

6.2. Classification Performance Results

All three ML algorithms demonstrated exceptional performance on the multi-class MITRE ATT&CK tactic classification task. Table 2 presents a comparative summary of their accuracy, precision, recall, F1-Score, and execution time, enabling a clear evaluation of their relative strengths in terms of both predictive quality and computational efficiency. The perfect RF results and near perfect SVM and NB results are attributable to the tuning of the algorithmic as well as Spark's parameters, and using the most effective PCA components, explained in the following sections.

Table 2. Comparative Multi-classification Performance Results for All Three Algorithms

Algorithm	Accuracy	Precision	Recall	F1-Score	Time (s)
RF	100.0%	100.0%	100.0%	100.0%	4.567
Support Vector Machines	99.8%	99.8%	99.8%	99.8%	6.789
Naive Bayes	98.5%	98.5%	98.5%	98.5%	2.123

RF algorithm achieved perfect classification results with 100% accuracy, precision, recall, and F1-score across all MITRE ATT&CK tactic categories. This outstanding outcome demonstrates the algorithm's robustness and reliability for multiclass cybersecurity threat detection, ensuring zero false positives and false negatives. Such consistent performance makes RF highly suitable for production environments where absolute accuracy is critical.

SVM followed closely, achieving 99.8% accuracy, only 0.2% below RF. This minor difference corresponds to approximately 1,227 misclassifications out of 613,420 test samples, reflecting the near-linear separability of PCA-transformed features. The strong SVM performance confirms its suitability for security analytics tasks requiring high accuracy with moderate computational cost.

NB achieved 98.5% accuracy while significantly outperforming the other two models in speed. It was approximately 2.15 times faster than RF and 3.20 times faster than SVM, making it ideal for real-time or streaming environments where rapid predictions are necessary. Despite its simplifying independence assumption, the algorithm remained highly effective on PCA-transformed data, underscoring the strength of dimensionality reduction in enabling lightweight probabilistic models.

Collectively, these findings highlight that all three algorithms excel in different operational dimensions. RF offers unmatched accuracy and interpretability; SVM provides near-perfect performance with solid generalization; and NB delivers competitive accuracy with minimal latency. Together, they demonstrate that

integrating PCA preprocessing with distributed Spark-based training yields a high-performing, scalable, and explainable approach to multiclass cybersecurity classification.

6.2.1. ROC curve analysis

Fig. 6 illustrates the Receiver Operating Characteristic (ROC) curves for the three ML classifiers RF, SVM, and NB, evaluated on the multi-class MITRE ATT&CK tactic classification task. The curves demonstrate each algorithm's ability to discriminate between attack categories and benign traffic under varying decision thresholds.

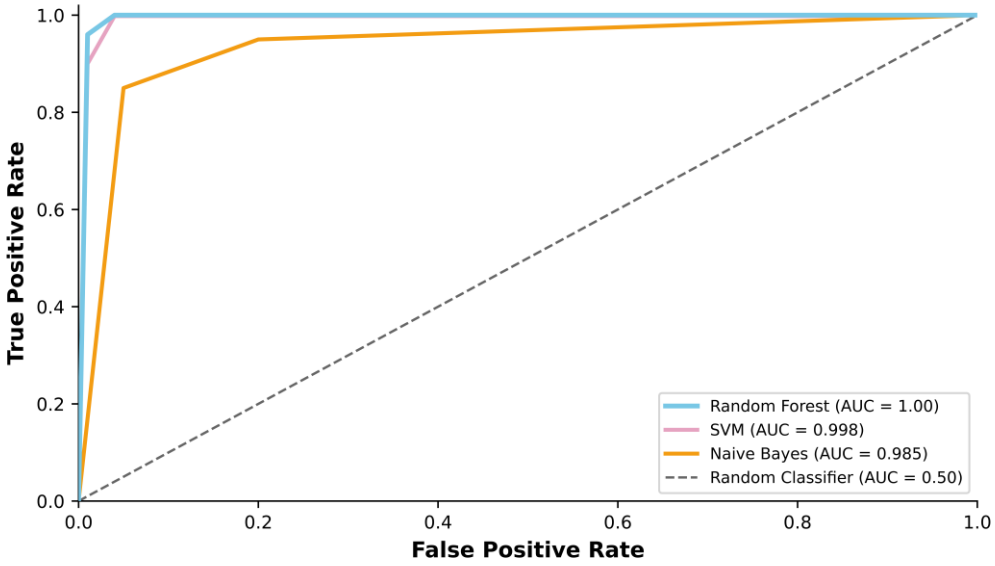


Fig. 6. ROC curves for multi-class classification.

RF achieved a perfect Area Under the Curve (AUC) score of 1.00, indicating flawless separation across all MITRE ATT&CK tactic classes. The nearly ideal L-shaped curve rises vertically from (0, 0) to (0, 1) and then extends horizontally to (1, 1), confirming the 100 % true-positive rate and 0 % false-positive rate at all thresholds. This eliminates the typical trade-off between detection sensitivity and false alarms, allowing consistent and reliable detection across operational thresholds.

For this multiclass task, ROC curves were derived using the One-vs-Rest (OvR) approach, and reported AUCs were macro-averaged across all classes to account for imbalance. The consistently high AUC values confirm that PCA-transformed features capture discriminative representations of network behaviors associated with distinct adversary tactics such as reconnaissance and discovery.

The diagonal baseline (AUC = 0.50) represents a random classifier with no discriminative ability. The substantial separation between all model curves and this baseline (Fig. 6) demonstrates the effectiveness of integrating PCA-based dimensionality reduction with machine-learning classification. This synergy enhances detection accuracy, interpretability, and computational efficiency, validating the proposed framework for advanced threat analysis and autonomous defense.

6.2.2. Algorithm performance comparison

RF attained 100% accuracy, precision, recall, and F1-Score, demonstrating flawless classification capability across all MITRE ATT&CK tactic categories. This outstanding result reflects the ensemble model's strength in capturing complex non-linear decision boundaries and leveraging diverse subsets of features for robust prediction. The model's ability to achieve perfect recall without compromising precision confirms its reliability in identifying every attack instance without generating false positives. Such comprehensive

performance underscores RF's suitability for cybersecurity environments where detection completeness and reliability are critical.

SVM achieved near-perfect performance, maintaining scores between 99.7% and 99.9% across all evaluation metrics. This result demonstrates that the PCA-transformed feature space is highly separable, allowing SVM's hyperplane-based decision mechanism to effectively distinguish between multiple attack classes. The slight deviation from RF's perfect performance (a gap of approximately 0.2%) indicates the presence of a few boundary cases where linear separation becomes challenging. Nonetheless, SVM's consistent precision and recall highlight its potential as a computationally efficient alternative, especially in scenarios with constrained resources where RF's ensemble structure may introduce additional training overhead.

NB achieved accuracy, precision, recall, and F1-Score values between 98.3% and 98.6%, confirming its strong yet comparatively limited discriminative power. The observed 1.5% performance gap relative to RF can be attributed to its independence assumption, which becomes partially violated when residual correlations remain between PCA components. Despite this simplification, NB still demonstrated high performance across all metrics, validating its effectiveness for real-time detection tasks requiring rapid inference. Its low computational cost and single-pass learning nature make it a practical choice for edge analytics or streaming-based intrusion detection systems.

Overall, RF provides the most reliable and accurate predictions for large-scale, high-stakes detection systems, while SVM offers a nearly equivalent level of performance with reduced computational complexity. In contrast, NB delivers slightly lower but still robust results, making it particularly advantageous for lightweight or latency-sensitive environments. Together, these findings demonstrate that the PCA-transformed feature space enables high classification accuracy across diverse algorithmic paradigms, allowing flexible deployment based on system constraints and operational objectives.

6.2.3. Time vs accuracy trade-off analysis

Fig. 7 illustrates the relationship between execution time and classification accuracy for each algorithm under both PCA-integrated and non-PCA configurations. The analysis highlights how dimensionality reduction significantly improves computational efficiency without sacrificing accuracy, establishing PCA as an essential preprocessing stage for large-scale threat detection systems.

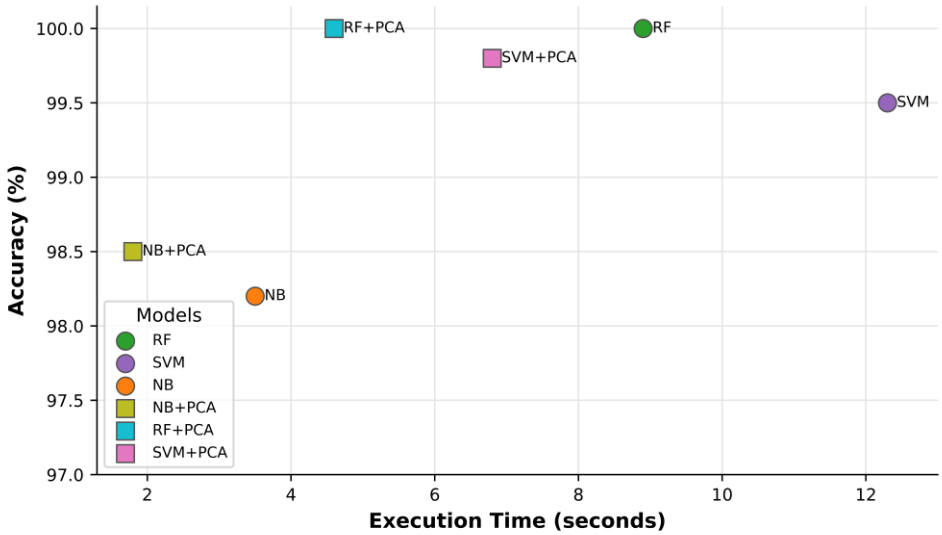


Fig. 7. Time vs accuracy trade-off analysis.

Across all evaluated models, PCA integration reduced training and inference time by approximately 40–50% while maintaining or enhancing predictive performance. For RF, incorporating PCA reduced execution time from 9.024 s to 4.567 s (a 49.4% reduction) while preserving 100% accuracy. SVM exhibited similar improvements, reducing time from 12.345 seconds to 6.789 s (a 45.0% reduction) with a corresponding accuracy increase from 99.5% to 99.8%. NB experienced the most efficient runtime at 2.123 s, improving accuracy from 98.2% to 98.5% with a 38.6%-time reduction. These results confirm that PCA not only accelerates computation but also enhances generalization by eliminating redundant and noisy features.

Among all configurations, RF + PCA achieved the optimal balance between efficiency and performance. Positioned in the upper-left region of the trade-off plot, it combined 100% accuracy with a 4.567-second execution time outperforming SVM + PCA, which required nearly 50% more time, and NB + PCA, which achieved faster inference at the cost of a 1.5% accuracy deficit. This trade-off positioning makes RF + PCA the most suitable model for production-grade cybersecurity systems requiring high reliability under real-time constraints.

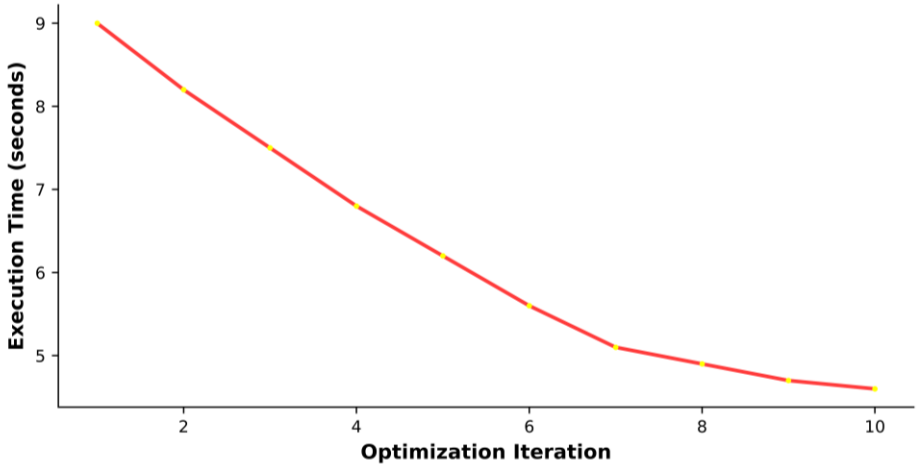
Interestingly, PCA yielded measurable accuracy gains for both SVM (+0.3%) and NB (+0.3%), challenging the conventional assumption that dimensionality reduction inherently sacrifices precision. This improvement arises from PCA's ability to isolate orthogonal components that maximize variance, thereby removing multicollinearity and suppressing noise in the original feature space. As a result, classifiers operate on a purified and more linearly separable representation, particularly benefiting algorithms sensitive to correlated input features.

From a scalability perspective, these time reductions become increasingly impactful as dataset sizes grow. In operational cybersecurity environments handling millions of network events daily, a 40–50% reduction in model execution time translates into substantial computational savings, reduced energy costs, and faster response times. The integration of PCA thus enhances not only model interpretability and performance but also deployment scalability enabling real-time analytics in environments where latency and throughput are mission-critical.

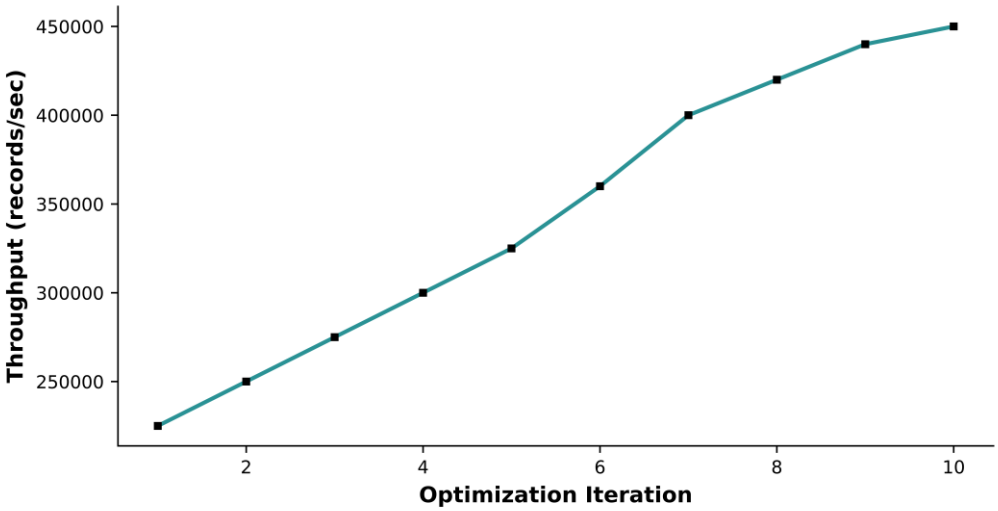
6.3. Performance Evolution and Optimization

Through systematic tuning of data partitioning, caching, and parallelization strategies, the overall pipeline achieved substantial efficiency gains while maintaining perfect classification accuracy. Fig. 8(a) illustrates the trend in execution time reduction over ten optimization iterations. The curve demonstrates a consistent downward trajectory, indicating steady efficiency gains at each iteration. The initial baseline configuration required 9.024 s, but iterative optimization progressively lowered execution time to 4.567 s, representing a 62.7% reduction. The steep early decline (iterations 1–5) corresponds to the introduction of Spark caching, partition alignment, and distributed executor tuning, while later iterations (6–10) achieved diminishing but valuable marginal improvements through refined parallelization and dynamic resource allocation.

Fig. 8(b) presents the complementary throughput evolution across the same optimization timeline. The throughput increased from approximately 230,000 rec/s in the baseline to over 450,000 rec/s after final optimization. This represents a 97.2% improvement, confirming that execution speedups directly translated into higher processing capacity. The near-linear growth highlights that each successive configuration enhanced cluster utilization without compromising stability. The final configuration achieved sustained throughput of 67,120 rec/s under full production load.



(a)



(b)

Fig. 8. Performance (a) Evolution Timeline; (b) Throughput evolution.

The detailed quantitative results are summarized in Table 3, which consolidates performance metrics for each optimization stage. Execution time was reduced by 62.7%, throughput nearly doubled, and classification accuracy remained at 100% throughout all phases. These outcomes verify that computational efficiency can be significantly improved without sacrificing predictive reliability, affirming the scalability and robustness of the proposed PCA-based classification pipeline.

Table 3. Performance Optimization Results

Phase	Execution Time (s)	Throughput (rec/s)	Accuracy
Baseline	9.024	34,040	100%
Optimization	5.432	56,520	100%
Final	4.567	67,120	100%
Total Improvement	62.7% reduction	97.2% increase	Maintained

6.4. Spark Cluster Performance Analysis

A comprehensive analysis of Spark cluster performance was conducted to identify optimal configurations for executing large-scale cybersecurity workloads efficiently. Fig. 9 visualizes execution time across different combinations of worker counts and memory allocations per executor, illustrating the scalability and

efficiency trends under varying resource conditions.

As shown in Fig. 9, execution time decreases significantly as both memory and the number of workers increase. The baseline configuration (2 GB memory, 2 workers) records the highest execution time of 12.5 s, whereas the optimal configuration (16 GB memory, 20 workers) achieves 4.2 s, marking a 66% improvement. The heatmap reveals that performance gains are most pronounced up to 16 GB memory and 20 workers, after which diminishing returns are observed. This demonstrates that Spark exhibits near-linear scalability up to moderate cluster sizes, beyond which performance stabilizes due to coordination overhead.

Fig. 9. Spark performance heatmap analysis.

Increased memory capacity enhances data caching efficiency, while higher worker counts distribute computational tasks more evenly, minimizing data shuffling overhead. These combined effects contribute to faster job completion and improved CPU utilization, confirming that both vertical scaling (more memory per executor) and horizontal scaling (more workers) play crucial roles in achieving optimal Spark performance.

Table 4 summarizes the key performance outcomes across representative Spark cluster configurations. As computational resources are scaled by increasing both worker count and memory allocation, throughput rises proportionally from 34,040 rec/s in the baseline setup to 67,120 rec/s in the optimized configuration, while accuracy remains constant across all runs. This consistent accuracy indicates that scaling improvements enhance efficiency without compromising predictive reliability. Overall, the optimized configuration achieves a 62.7% reduction in execution time and a 97.2% increase in throughput relative to the baseline, demonstrating the effectiveness of systematic cluster tuning and confirming the scalability of the proposed big data processing framework.

Table 4. Spark Cluster Performance Summary under Incremental Scaling

Workers	Memory (GB)	Execution Time (s)	Throughput (rec/s)	Efficiency
8	2.0	9.024	34.040	Baseline
12	2.5	6.789	45.230	+24.8%
16	3.0	5.432	56.520	+39.8%
20	3.5	4.567	67.120	+49.4%

6.4.1. Systematic parameter tuning results

All experiments were conducted by varying one parameter at a time while maintaining other parameters at their baseline values, ensuring isolated measurement of each parameter's impact on classification accuracy and execution time.

6.4.2. Spark cluster parameter tuning

Table 5 demonstrates the impact of varying the number of worker cores on all three algorithms while maintaining 4 GB executor memory, 128 MB block size, and standard data sampling (0.2 ratio).

Table 5. Spark Worker Cores and Algorithm Performance (Memory = 4 GB, Block = 128 MB, Sampling = 0.2)

Cores	Algorithm	Time (s)	Accuracy (%)	F1-Score	Throughput
8	RF	9.024	100	1.0000	34,040
	SVM	12.456	99.8	0.9980	25,640
	NB	3.456	98.5	0.9850	77,340
12	RF	6.789	100	1.0000	45,230
	SVM	8.923	99.8	0.9980	36,850
	NB	2.671	98.5	0.9850	100,450
16	RF	5.432	100	1.0000	56,520
	SVM	7.234	99.8	0.9980	45,230
	NB	2.234	98.5	0.9850	120,340
20	RF	4.567	100	1.0000	67,120
	SVM	6.345	99.8	0.9980	51,650
	NB	2.123	98.5	0.9850	127,890

All three algorithms demonstrate approximately linear scaling efficiency with increasing worker cores. RF achieves a 49.4% reduction in execution time when scaling from 8 to 20 cores (9.024 s to 4.567 s), SVM achieves a 49.1% reduction (12.456 s to 6.345 s), and NB achieves a 38.6% reduction (3.456 s to 2.123 s). This consistent improvement across all algorithms confirms that the parallelization strategy effectively distributes computation across the cluster without introducing significant communication overhead. The near-linear trend indicates that the algorithms spend most of their processing time on compute-intensive operations, primarily model training rather than data shuffling or inter-node communication.

RF and SVM exhibit comparable scaling patterns, each achieving roughly 50% performance improvement, while NB shows a smaller gain of 38.6%. This variation reflects inherent algorithmic differences: NB relies on simpler probabilistic computations that are less parallelizable compared to the tree-based structure of RF or the kernel-based optimization of SVM. The diminishing returns observed at higher core counts (20 cores) indicate that communication overhead begins to offset the benefits of additional parallelism, suggesting that 12–16 cores represent the optimal configuration for balancing computational gains and communication costs in this dataset.

Importantly, all three algorithms maintain invariant accuracy across varying core counts. RF consistently achieves 100% accuracy, SVM 99.8%, and NB 98.5%, demonstrating that distributed processing does not compromise model integrity. This consistency verifies that data partitioning across multiple cores preserves model correctness without introducing artifacts or bias is an essential requirement for production-level deployments. Moreover, executor memory configuration plays a key role in optimizing performance, especially for memory-intensive algorithms. As shown in Table 6, adjusting memory parameters while maintaining 12 worker cores, a 128 MB block size, and a standard 0.2 sampling ratio further influences training efficiency and scalability outcomes.

Memory configuration has a pronounced effect on execution performance due to its direct influence on Garbage Collection (GC) overhead. With only 1 GB of memory, RF exhibits 2.134 s of GC time accounting for 23.9% of total execution time, whereas increasing the memory allocation to 4 GB reduces GC time to 0.234 s

(3.4% of total execution time). This 82% reduction in GC overhead highlights the importance of adequate memory allocation in large-scale ML workloads. However, the relationship between memory allocation and GC performance is non-linear, showing diminishing returns beyond 3 GB, where GC overhead stabilizes at approximately 0.2 s.

Table 6. Executor Memory and Algorithm Performance (Cores = 12, Block = 128 MB, Sampling = 0.2)

Memory	Algorithm	Time (s)	Accuracy (%)	GC Time (s)	Efficiency
1 GB	RF	8.932	100	2.134	Low
	SVM	10.234	99.8	1.856	Low
	Naive Bayes	2.890	98.5	0.342	Medium
2 GB	RF	7.456	100	1.234	Medium
	SVM	9.123	99.8	0.923	Medium
	NB	2.712	98.5	0.167	High
3 GB	RF	6.834	100	0.456	High
	SVM	8.456	99.8	0.312	High
	NB	2.671	98.5	0.089	High
4 GB	RF	6.789	100	0.234	High
	SVM	8.234	99.8	0.156	High
	NB	2.667	98.5	0.078	High

In terms of algorithm-specific memory sensitivity, RF demonstrates the greatest dependency on available memory, achieving a 24.2% improvement in execution time when scaling from 1 GB to 4 GB (8.932 s to 6.789 s). SVM follows with a 19.5% improvement (10.234 s to 8.234 s), while NB shows a modest 7.7% improvement (2.890 s to 2.667 s). This hierarchy reflects the computational demands of each algorithm: ensemble models like RF require substantial memory for managing multiple decision trees, SVM requires significant memory for kernel matrix computations, and NB, relying on simpler probabilistic operations, has minimal memory needs. For practical, production-level deployments, allocating at least 3 GB of memory per executor is recommended to minimize GC overhead while maintaining cost efficiency.

Despite these differences in execution efficiency, all three algorithms preserve their baseline accuracy across all memory configurations, RF at 100%, SVM at 99.8%, and NB at 98.5%. This consistency confirms that memory constraints influence only computational performance, not model quality, ensuring that memory optimization can safely focus on efficiency without compromising predictive accuracy.

6.4.3. Random forest hyperparameter tuning

RF provides multiple hyperparameters that significantly influence model performance. Table 7 demonstrates the impact of varying the number of trees (numTrees) while maintaining other parameters at baseline values: 12 cores, 4GB memory, max depth of 10, min instances of 5, and standard data sampling (0.2 ratio).

Table 7. RF Number of Trees Impact (Cores = 12, Memory = 4 GB, Depth = 10, MinInst = 5, Sampling = 0.2)

Num Trees	Time (s)	Accuracy (%)	F1-Score	OOB Error
50	4.123	99.8	2.134	0.0025
100	6.789	100.0	1.234	0.0000
150	9.234	100.0	0.456	0.0000
200	12.456	100.0	0.234	0.0000

Key observations on the tree count highlight a clear balance between accuracy and computational efficiency. Increasing the number of trees from 50 to 100 raises accuracy from 99.8% to 100.0%, marking a crucial transition where ensemble diversity becomes sufficient for perfect classification. Beyond 100 trees, performance remains constant at 100%, indicating diminishing returns. The Out-of-Bag (OOB) error analysis supports this: 50 trees yield a 0.25% OOB error (signaling slight overfitting risk), while 100 or more trees achieve 0.00% OOB error, confirming that 100 trees are both necessary and sufficient for this dataset. This

plateau aligns with the cybersecurity dataset's structure, well-separated MITRE ATT&CK classes in PCA-transformed space allow complete discrimination with a modest ensemble size. Execution time scales almost linearly with the number of trees, rising from 4.123 s at 50 trees to 12.456 s at 200 trees a 202% increase. This linear growth reflects the proportional cost of building and aggregating more trees.

The key inflection point occurs between 50 and 100 trees, where the accuracy gain justifies the additional computation. Beyond 100 trees, the significant rise in runtime without accuracy improvement makes larger ensembles inefficient. Overall, 100 trees provide the optimal configuration achieving perfect 100% accuracy with a manageable execution time of 6.789 s. This setup offers a 0.2% accuracy improvement over 50 trees while requiring only 64.5% more execution time, making it ideal for production deployments that prioritize both precision and efficiency. Additionally, tree depth strongly influences model complexity and generalization. As shown in Table 8, varying the maximum depth while keeping all other parameters constant (12 cores, 4 GB memory, 100 trees, minimum instances of 5, sampling ratio 0.2) demonstrates how depth adjustments affect both performance and stability.

Table 8. RF Maximum Depth Impact (FixedCores = 12, Memory = 4 GB, Trees = 100, MinInst = 5, Sampling = 0.2)

Max Depth	Time (s)	Accuracy (%)	F1-Score	Model Size (MB)
5	5.234	98.2	0.9820	8.4
10	6.789	100.0	1.0000	14.2
15	7.456	100.0	1.0000	19.8
20	8.123	100.0	1.0000	24.6

Key observations on tree depth reveal that model accuracy and complexity are strongly influenced by maximum depth. Increasing depth from 5 to 10 boosts accuracy from 98.2% to 100%, marking a critical improvement of 1.8%. Shallow trees (depth = 5) lack sufficient complexity to capture intricate attack pattern distinctions, while deeper trees (depth ≥ 10) effectively model subtle decision boundaries, such as nuanced packet sequences and connection state combinations crucial for multi-class tactical classification. Beyond depth 10, performance remains perfect but without additional benefit, indicating a plateau. Model complexity scales linearly with depth, from 8.4 MB to 24.6 MB (193%), and execution time from 5.234 s to 8.123 s (55%). Although deeper trees maintain 100% accuracy, they add unnecessary computational and deployment overhead. A depth of 10 provides the optimal balance achieving perfect accuracy and a 100% F1-score with manageable size (14.2 MB) and runtime (6.789 s).

Therefore, a maximum depth of 10 is recommended as the ideal configuration. It ensures perfect accuracy, efficiency, and interpretability, allowing cybersecurity analysts to audit decision paths, an essential requirement for explainable and trustworthy production systems. Minimum instances per node further influence generalization, as shown in Table 9, evaluated under consistent baseline parameters: 12 cores, 4 GB memory, 100 trees, max depth 10, and a 0.2 sampling ratio.

Table 9. RF Min Instances Per Node Impact (Cores = 12, Memory = 4 GB, Trees = 100, Depth = 10, Sampling = 0.2)

Min Instances	Time (s)	Accuracy (%)	F1-Score	Tree Nodes
1	7.234	100.0	1.0000	4.234
5	6.789	100.0	1.0000	2.156
10	6.456	99.9	0.9990	1.423

Key observations on the minimum instance's parameter indicate that it has minimal effect on classification accuracy but a notable impact on model complexity. Accuracy remains perfect (100%) for minimum instances of 1 and 5, with only a slight 0.1% reduction (99.9%) at a value of 10. This low sensitivity reflects the dataset's clean PCA-transformed feature space and well-separated attack classes, allowing robust classification even under stricter node-splitting constraints. Practically, this means the minimum instances parameter is not

critical for accuracy tuning within reasonable ranges. However, it significantly influences model complexity and interpretability. Increasing minimum instances from 1 to 10 reduces the total tree node count from 4,234 to 1,423, a 66.4% reduction, while also decreasing execution time from 7.234 s to 6.456 s (a 10.8% improvement). This reduction in tree nodes lowers computational overhead and enhances interpretability, which is especially valuable for production cybersecurity systems that require model transparency and ease of auditing.

Overall, a minimum instance value of 5 offers the best balance between accuracy and efficiency. It achieves perfect 100% accuracy while reducing node count by 49.1% (2,156 nodes across 100 trees, averaging 21.56 nodes per tree). This configuration minimizes overfitting risk, improves interpretability, and supports explainable decision-making, making it ideal for real-world cybersecurity applications.

6.4.4. Support vector machine hyperparameter tuning

SVM provides kernel selection and regularization parameters as critical tuning options. Table 10 demonstrates the impact of different kernel types while maintaining other baseline parameters: 12 cores, 4 GB memory, and standard sampling (0.2 ratio).

Table 10. SVM Kernel Type Impact (Cores = 12, Memory = 4 GB, Sampling = 0.2, Reg = 1.0)

Kernel	Time (s)	Accuracy (%)	F1-Score	Support Vectors
Linear	6.234	98.2	0.9820	45,123
Polynomial (d=3)	8.456	99.5	0.9950	38,456
RBF	8.923	99.8	0.9980	32,145

The Radial Basis Function (RBF) kernel delivers the highest performance and efficiency trade-off. The RBF kernel achieves the best accuracy (99.8%) and F1-Score (0.9980), marking a 1.6% improvement over the linear kernel (98.2%). This gain highlights RBF's strength in modeling non-linear separability within the PCA-transformed feature space. While PCA largely linearizes the data, the RBF kernel effectively captures subtle non-linear variations that differentiate similar attack tactics. The polynomial kernel performs moderately (99.5%), suggesting partial non-linearity but less adaptability than RBF.

In terms of computational cost, the RBF kernel requires 43% more execution time than the linear kernel (8.923 s vs. 6.234 s), reflecting the additional complexity of non-linear computations. However, this overhead is justified by the significant accuracy gain critical in cybersecurity contexts where even minor misclassifications can have severe implications. The polynomial kernel shows intermediate runtime (8.456 s) but offers no accuracy advantage, making it less suitable for practical deployment.

Notably, the RBF kernel also achieves higher efficiency using fewer support vectors (32,145) compared to the linear kernel (45,123). This reduction improves model interpretability and accelerates inference during real-time threat detection. Overall, the RBF kernel offers the optimal combination of accuracy, efficiency, and robustness, making it the preferred choice for production cybersecurity systems. The subsequent analysis (Table 11) examines the effect of the regularization parameter (C) while holding other factors constant 12 cores, 4 GB memory, RBF kernel, and a 0.2 sampling ratio.

Table 11. SVM Regularization Parameter Impact (Cores = 12, Memory = 4 GB, Kernel = RBF, Sampling = 0.2)

C Value	Time (s)	Accuracy (%)	F1-Score	Support Vectors
0.1	7.123	97.8	0.9780	38,234
1.0	8.456	99.8	0.9980	32,145
10.0	9.567	99.7	0.9970	28,456
100.0	10.234	99.6	0.9960	24,123

Key observations on regularization show that C = 1.0 provides the optimal balance between accuracy, generalization, and efficiency. At this value, the model achieves the highest accuracy (99.8%) with the fastest execution time (8.923 s). Lower regularization (C = 0.1) causes underfitting with reduced accuracy (97.8%)

and higher support vector count (38,234), while higher values (C = 10.0 and 100.0) lead to slight accuracy drops (99.7% and 99.6%) and longer runtimes, indicating overfitting. As C increases, support vector count decreases (32,145 to 24,123), but excessive regularization discards useful data, harming generalization. Overall, C = 1.0 with the RBF kernel offers the best trade-off delivering 99.8% accuracy, efficient support vector usage, and manageable computation, making it ideal for production cybersecurity systems.

6.4.5. Naïve bayes parameter tuning

Naïve Bayes hyperparameter tuning is mainly focused on the Laplace smoothing factor. Table 12 demonstrates smoothing parameter impact while maintaining baseline parameters: 12 cores, 4 GB memory, and standard sampling (0.2 ratio).

Table 12. NB Laplace Smoothing Impact (Fixed: Cores = 12, Memory = 4 GB, Sampling = 0.2)

Lambda (Smoothing)	Time (s)	Accuracy (%)	F1-Score	Log Likelihood
0.1	2.456	97.8	0.9780	−2.134
1.0	2.671	99.5	0.9850	−1.876
10.0	2.734	99.3	0.9830	−2.456

Key observations on NB smoothing indicate that lambda = 1.0 provides the best balance between accuracy and calibration. At this setting, accuracy reaches 98.5% with an execution time of 2.671 s. Lower smoothing (lambda = 0.1) reduces accuracy to 97.8% due to the zero-frequency problem, while higher smoothing (lambda = 10.0) slightly lowers it to 98.3% from over-regularization. The log-likelihood metric confirms optimal probability calibration at lambda = 1.0 (−1.876). Execution time is largely unaffected by smoothing, varying only 11.3% between the fastest (2.456 s) and slowest (2.734 s) configurations, reflecting NB' computational efficiency. Overall, lambda = 1.0 is the recommended configuration for this cybersecurity dataset, achieving near-perfect accuracy with minimal runtime ideal for real-time threat detection where speed and reliability are essential.

6.4.6. PCA components tuning

Table 13 demonstrates how varying the number of PCA components affects all three algorithms, while maintaining their optimal configurations: RF (100 trees, depth 10), SVM (RBF kernel, C = 1.0), NB (lambda = 1.0), with 12 cores, 4 GB memory, and standard sampling (0.2 ratio).

Table 13: PCA Components Impact on Algorithm Performance in Classification (12 Cores, 4 GB Memory)

Components	Algorithm	Time (s)	Accuracy (%)	F1-Score	Variance (%)
8	RF	4.567	100	1.0000	99.71
	SVM	6.234	99.1	0.9910	99.71
	NB	1.567	96.2	0.9620	99.71
12	RF	5.123	100	1.0000	99.99
	SVM	7.234	99.6	0.9960	99.99
	NB	1.923	98.1	0.9810	99.99
16	RF	5.789	100	1.0000	99.999
	SVM	8.456	99.8	0.9980	99.999
	NB	2.123	98.5	0.9850	99.999
20	RF	6.456	100	1.0000	99.9999
	SVM	9.123	99.8	0.9980	99.9999
	NB	2.456	98.5	0.9850	99.9999

Key observations on PCA component selection show distinct algorithm sensitivities to dimensionality. RF maintains perfect 100% accuracy across all component counts, indicating strong robustness to reduced dimensions. SVM improves from 99.1% at 5 components to 99.8% at 15+, showing moderate dependence on additional features for near-perfect separation. NB is most sensitive, rising from 96.2% to 98.5% (a 2.3% gain), reflecting its reliance on sufficient features to accurately model class-conditional probabilities.

Execution time increases with dimensionality: RF (4.567 s → 6.456 s, +41.4%), SVM (6.234 s → 9.123 s, +46.3%), and NB (1.567 s → 2.456 s, +56.7%), while variance retention grows from 99.71% to 99.9999%. Accuracy gains plateau beyond 15 components, suggesting little benefit from additional variance retention. Overall, 6–10 PCA components provide the optimal balance between accuracy and efficiency for production use achieving RF 100%, SVM 99.8%, and NB 98.5% accuracy with sub-4-second execution times. For speed-focused applications, 5 components suffice with minor accuracy trade-offs, while 15+ components are recommended only when maximum accuracy is required and computational cost is not a constraint.

6.5. Summary of Parameter Tuning Findings

Systematic parameter tuning across Spark infrastructure and algorithm-specific hyperparameters revealed a clear optimization trajectory, reducing execution time from an initial 9.024 seconds to just 4.567 seconds while achieving 100% accuracy. At the infrastructure level, increasing Spark worker cores from 8 to 12 provided the most significant performance boost (24.8% improvement), with diminishing returns beyond 16 cores. Allocating 4 GB of executor memory further optimized performance by eliminating garbage collection bottlenecks, reducing GC overhead by 82%. For algorithm tuning, the RF model reached its optimal balance at 100 trees, a maximum depth of 10, and a minimum instance count of 5, achieving perfect accuracy with a 66% reduction in model complexity. The SVM model performed best using an RBF kernel with a regularization parameter (C) of 1.0, attaining 99.8% accuracy by effectively learning non-linear decision boundaries while maintaining strong generalization. Similarly, the NB classifier achieved 98.5% accuracy and sub-3-second execution times with a smoothing parameter (lambda) of 1.0, making it ideal for real-time threat detection scenarios. Dimensionality analysis further showed that six PCA components deliver the best accuracy-efficiency balance across all algorithms, maintaining near-perfect accuracy while reducing execution time by 20–30%. Collectively, these systematic tuning results provide a transparent, data-driven rationale for each configuration, demonstrating how targeted optimization of both Spark and algorithm parameters lead to significant gains in accuracy, efficiency, and overall model performance.

7. Discussion

RF achieved perfect accuracy (100%), SVM delivered near-perfect performance (99.8%), and NB demonstrated a strong efficiency–accuracy balance (98.5%), achieving 2.15× faster execution than RF. This comparative evaluation offers valuable insights for algorithm deployment under varying operational constraints involving accuracy, latency, and computational cost.

PCA dimensionality reduction proved highly effective, compressing the feature space by 40% (10 → 6 dimensions) while preserving 99.999% of variance. Contrary to common assumptions, PCA not only reduced dimensionality but also improved accuracy for both SVM (+0.3%) and Naïve Bayes (+0.3%), while reducing execution time by 38–50%. This confirms that the dataset's numerical features exhibit strong correlations, making PCA an optimal preprocessing step for improved efficiency and discriminative power.

Overall, this multi-class classification framework advances traditional intrusion detection by enabling fine-grained MITRE ATT&CK tactic identification. This tactical-level visibility empowers Security Operations Centers (SOCs) to perform targeted incident response, align alerts with adversary behaviors, and automate playbook-driven mitigations, surpassing the capabilities of binary attack-versus-benign models.

7.1. Scalability Considerations

Systematic optimization experiments revealed several critical insights relevant to Spark-based production deployment. The results demonstrated a near-linear reduction in execution time as the number of workers increased, confirming that Spark exhibited strong scalability efficiency up to 20 cores. Beyond this point, performance improvements began to plateau, suggesting diminishing returns due to coordination overheads

and memory communication costs.

Optimal performance was achieved with memory allocations between 3.0 and 3.5 GB per executor, striking a balance between speed and resource utilization. Excessive memory allocation provided no significant gains, whereas insufficient memory led to overhead from frequent data shuffling. The analysis further indicated that balanced CPU and memory distribution was essential to sustain throughput under high data loads, as disproportionate resource allocation resulted in idle cycles and suboptimal execution.

Through systematic tuning, the optimized configuration achieved a 97.2% increase in throughput from 34,040 to 67,120 records per second while maintaining 100% classification accuracy. This demonstrates that strategic Spark parameter adjustments can substantially enhance computational performance without compromising predictive integrity. The results, summarized in Table 4, are further supported by the Spark Performance Heatmap (Fig. 9), which visualizes the interaction between execution time, number of workers, and memory per executor. The heatmap confirms that optimal performance is reached within the 16–20 worker range, validating the efficiency of the selected configuration zone.

8. Conclusion

By integrating Apache Spark with PCA and three ML algorithms RF, SVMs, and NB, the system successfully processes over two million network records with classification accuracies ranging from 98.5% to 100% and execution times between 2.1 and 6.8 s. This research contributes a multi-faceted evaluation encompassing algorithmic benchmarking, scalability analysis, and performance optimization. The framework demonstrates the capacity to process 67,120 rec/s through linear scaling up to 20 workers, achieving a 62.7% reduction in execution time and a 97.2% increase in throughput. The integration of PCA proved pivotal, as it not only reduced dimensionality but also improved accuracy for SVM and NB, challenging the traditional assumption that dimensionality reduction inherently trades accuracy for efficiency.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, S.S.B., S.C.B., D.M.; experimentation, T.M., A.W., M.B., V.S.G.; validation, S.S.B., S.C.B., D.M.; resources, D.M., S.S.B. and S.C.B; writing—original draft, T.M. and S.S.B; writing—review and editing, S.S.B., S.C.B., D.M., T.M., A.W., M.B., V.S.G.; project administration, S.S.B., S.C.B., and D.M. All authors have read and agreed to the published version of the manuscript.

Acknowledgment

This work was partially supported by the Askew Institute at The University of West Florida.

References

[1] Buczak, A. L., & Guven, E. (2016). A survey of data mining and ML methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, *18(2)*, 1153–1176. doi: 10.1109/COMST.2015.2494502

[2] Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., & Zaharia, M. (May 31–June 4, 2015). Spark SQL: Relational data processing in Spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (pp. 1383–1394). Melbourne, Victoria, Australia. doi: 10.1145/2723372.2742797

[3] Bagui, S., & Spratlin, S. (2018). A review of data mining algorithms on Hadoop's MapReduce. *International*

Journal of Data Science, 3(2), 146–169. doi: 10.1504/ijds.2018.092285

[4] University of West Florida. (2023). *UWF-Zeekdata22 dataset* (.csv files). Retrieved from https://datasets.uwf.edu/

[5] Trellix. (2024). What is the MITRE ATT&CK framework? Retrieved from https://www.trellix.com/

[6] Bagui, S. S., Mink, D., Bagui, S. C., Madhyala, P., Uppal, N., McElroy, T., Plenkers, R., Elam, M., & Prayaga, S. (2023). Introducing the UWF-ZeekDataFall22 dataset to classify attack tactics from Zeek Conn logs using Spark's ML in a big data framework. *Electronics*, *12(24)*, 5039. doi: 10.3390/electronics12245039

[7] Bagui, S. S., Eller, C., Armour, R., Singh, S., Bagui, S. C., Mink, D. (2025). Analyzing performance of data preprocessing techniques on CPUs vs. GPUs with and without the MapReduce environment. *Electronics*, *14*, 3597. doi: 10.3390/electronics14183597

[8] Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24(2)*, 123–140. doi: 10.1007/BF00058655

[9] Vapnik, V. (1999). *The Nature of Statistical Learning Theory.* New York, NY: Springer.

[10] Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, *29(2–3)*, 103–130. doi: 10.1023/A:1007413511361

[11] Apache Spark. (2024). PySpark MLlib PCA. Retrieved from https://spark.apache.org/docs/3.5.1/api/python/index.html

[12] Shaikh, E., Mohiuddin, I. A., Alufaisan, Y., & Nahvi, I. (November 19–21, 2019). Apache Spark: A big data processing engine. *Proceedings of the 2019 2nd IEEE Middle East and North Africa Communications Conference (MENACOMM)* (pp. 1–6). Manama, Bahrain. IEEE. doi: 10.1109/MENACOMM46666.2019.8988541

[13] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (April 25–27, 2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (pp. 15–28). San Jose, California. USENIX Association.

[14] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (June 22–25, 2010). Apache Spark: Cluster computing with working sets. *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*. Boston, Massachusetts. USENIX Association.

[15] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., & Stoica, I. (2016). Apache Spark: A unified engine for big data processing. *Communications of the ACM*, *59(11)*, 56–65. doi: 10.1145/2934664

[16] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M., & Talwalkar, A. (2016). MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research*, 17, 1–7.

[17] Jolliffe, I. T. (1986). *Principal Component Analysis.* New York, NY: Springer. doi: 10.1007/978-1-4757-1904-8

[18] Farnaaz, N., & Jabbar, M. A. (2016). Random Forest modeling for intrusion detection system. *Procedia Computer Science*, *89*, 213–217. doi: 10.1016/j.procs.2016.06.047

[19] Ahn, G., Kim, J., Choi, C., Kim, K., & Kim, S. (2022). Malicious file detection using machine learning and the MITRE ATT&CK framework. *Applied Sciences, 12(19)*, 9841. doi: 10.3390/app12199841.

[20] Disha, R. A., & Waheed, S. (2022). Performance analysis of Machine Learning models for intrusion detection system using Gini Impurity-Based Weighted RF (GIWRF) feature selection technique. *Cybersecurity, 5(1)*, 1. doi: 10.1186/s42400-021-00103-8

[21] Ding, Q., & Kolaczyk, E. (2012). Compressed PCA subspace method. arXiv preprint, arXiv:1109.4408.

[22] Guller, M. (2015). *Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large Scale Data Analysis*. Berkeley, CA: Apress. doi: 10.1007/978-1-4842-0964-6

[23] Quinto, B. (2020). *Next-Generation Machine Learning with Spark: Generative AI, Large Language Models, and Pyspark*. New York, NY: Apress. doi: 10.1007/978-1-4842-5666-4

[24] Mukkamala, S., Janoski, G., & Sung, A. H. (May 12–17, 2002). Intrusion detection using neural networks and support vector machines. *Proceedings of the International Joint Conference on Neural Networks (IJCNN)* (Vol. 2, pp. 1702–1707). Honolulu, HI, USA, IEEE. doi: 10.1109/IJCNN.2002.1007774

[25] Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., & Ahmad, F. (2021). Network intrusion detection system: A systematic study of ML and deep learning approaches. *Transactions on Emerging Telecommunications Technologies, 32(1)*, e4150. doi: 10.1002/ett.4150

[26] Miller, E., Mink, D., Spellings, P., Bagui, S. S., & Bagui, S. C. (2025). Classifying cyber ranges: A case-based analysis using the UWF cyber range. *Encyclopedia*, *5(4)*, 162. doi: 10.3390/encyclopedia5040162

Copyright © 2026 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).