Journal of Trends in Computer Science and Smart Technology (ISSN: 2582-4104)
www.irojournals.com/tcsst/

Detecting Intrusions in MongoDB NoSQL Databases Using Machine Learning

Abdelilah Belhaj[1], Soumia Ziti[2], Khalil Ladrham[3], Souad Najoua Lagmiri[4], Karim El Bouchti[5]

[1,2,3]Intelligent Processing Systems and Security (IPSS) Team Faculty of Sciences, Mohammed V University, Rabat, Morocco.
[4]IRSM, Higher Institute of Management, Administration and Computer Engineering, Rabat, Morocco.
[5]Laboratory of Computer Systems Engineering (LISI), Faculty of Science Semlalia, Cadi Ayyad University, Marrakech, Morocco.

E-mail: [1]abdelilah_belhaj@um5.ac.ma, [2]s.ziti@um5r.ac.ma, [3]khalil_ladrham@um5.ac.ma, [4]snajoua.lagmiri@ismagi.ma, [5]Elbouchtikarim@gmail.com

Orcid ID: [1]0009-0009-1316-7137, [2]0000-0002-5357-9170, [3]0009-0004-9107-2401, [4]0000-0003-1608-6390, [5]0000-0003-1035-5131

Abstract

Due to their flexibility and scalability, NoSQL databases, most notably MongoDB, are widely adopted in the modern cloud environment. However, this popularity introduces new security vulnerabilities, such as NoSQL injections, brute-force attacks, and denial-of-service attacks, which traditional Intrusion Detection Systems (IDS) struggle to detect. In this paper, machine learning is adapted to detect five types of attacks targeting MongoDB including NoSQL injections, XSS through the $where operator, brute-force attacks, denial-of-service attacks through expensive queries, and preparatory scanning. For this purpose, the ToN_IoT dataset is adapted to the MongoDB context by selecting 150,000 representative samples and 9 native features, and implementing four composite features. Three models are evaluated: Random Forest, XGBoost, and MLP Classifier. XGBoost achieves the best performance with an inference time of 0.019 ms, an accuracy of 98.9% and a macro F1-score of 0.9889. Performance per class indicates that scanning attacks and brute-force attacks are the easiest to detect ($F1 > 0.98$). However, injection attacks and XSS attacks are more difficult to detect ($F1 < 0.98$) due to their signatures associated with the application level, which are difficult to detect. The present method outperforms previous methods reporting 96-98% accuracy while offering MongoDB specificity and interpretability. These experiments reveal that real-time detection of attacks on MongoDB is achievable with high efficiency, allowing it to be deployed in cloud environments.

Keywords: Machine Learning, Intrusion Detection System, Random Forest, XGBoost, MongoDB, Multi-Class Cloud Environment.

1. Introduction

The proliferation of cloud computing, large-scale distributed systems, and the Internet of Things (IoT) has led to a significant increase in the adoption of NoSQL databases [1]. Simultaneously, cyberattacks on data stores are increasing in prevalence and sophistication, making machine learning-based Intrusion Detection Systems (IDS) more important than ever

DOI: https://doi.org/10.36548/jtcsst.2026.3.003
Received: 27.03.2026, received in revised form: 30.04.2026, accepted: 19.05.2026, published: 05.06.2026

[2]. MongoDB is a widely used document-oriented NoSQL database known for its high performance [3]. However, it still suffers from the same types of security vulnerabilities seen in other NoSQL implementations, such as NoSQL injection attacks, password cracking attacks, scan attacks, and denial-of-service (DoS) attacks. A significant challenge is the lack of Intrusion Detection Systems specifically designed for NoSQL systems.

Most research on intrusion detection systems (IDS) [5] employs datasets including NSL-KDD [6], UNSW-NB15 [7], and CICIDS [8]. These databases primarily focus on network traffic in areas such as IoT, the cloud, and software-defined networks (SDN). However, there are limited studies addressing database attacks, and no multi-class classification approaches for NoSQL-specific threats [9]. This gap highlights the need to develop creative solutions to identify the characteristics of different attack vectors in NoSQL environments. Due to these limitations, this work proposes a multi-class intrusion detection framework designed specifically for NoSQL databases. It addresses MongoDB-related attacks, including NoSQL injection, cross-site scripting (XSS) attacks, brute-force attacks on passwords, denial-of-service attacks through expensive queries, and scanning of MongoDB ports (27017–27019). With few publicly-available datasets dedicated to NoSQL vulnerabilities, the ToN_IoT dataset [10] was adapted to preserve only attacks potentially relevant to MongoDB. The goal is to build a robust anomaly-detection system. This study evaluates and compares tree-based models from different algorithm categories, such as Random Forest [11], XGBoost [12], and the MLP Neural Network Classifier [13]. To evaluate the models, this work uses accuracy, overall F1 score, per-class metrics, confusion matrices, and ROC curves. This work also provides an analysis of feature significance to aid in understanding the results and to identify the variables with the highest impact. The rest of the paper is organized as follows. Section 2 reviews related research on NoSQL security and intrusion detection. Section 3 describes the methodology. The results of the experiment are reported in Section 4. Section 5 addresses the results and their implications. Section 6 summarizes the paper and discusses future research trends.

2. Related Work

According to recent studies, there have been considerable developments in intrusion detection through machine learning owing to the growth in IoT, cloud, and distributed technologies. As stated by Hozouri et al. [14] and Rehman et al. [15], the efficiency of intrusion detection systems depends heavily on the properties of datasets, standardization of modeling techniques, and constraints imposed during execution. However, the traditional approach of signatures is constrained to detect only previously discovered vulnerabilities.

Recent research has compared the efficiency of supervised machine learning algorithms using benchmarking datasets such as CICIDS2017 [8], UNSW-NB15 [7] and ToN_IoT [10]. Mohale et al. [17] proposed an Intrusion Detection System (IDS) that utilizes multiple classifiers such as LS-SVM, AdaBoost, XGBoost, and KNN, together with a very strong feature selection algorithm. It is noted that their experiments show excellent results, with an accuracy rate of 99.5% for CICIDS2017 and an F1 score of more than 98% for multi-class detection. Sharma et al. [18] conducted a comparative study by employing Random Forest, SVM, and KNN on ToN_IoT to detect attacks in VANETs. In particular, the Random Forest achieved the highest accuracy of 97.7%. Research affirms that ToN_IoT forms a basic dataset in the context of the Internet of Things, notably for databases comparable to MongoDB. Elsayed et al. [19] introduce SATIDS, a hierarchical Intrusion Detection System (IDS)

architecture, which implements MRMR and an improved LSTM and has been tested on ToN_IoT with an accuracy of 97.5% and an F1 score of 98.05%. It allows multi-level categorization, for example, normal traffic, attack types, and sub-types of attacks. All studies suggest the Random Forest [11], XGBoost [12], and deep neural networks such as the improved MLP [13] are currently the optimal models for IDS metrics. However, these studies mostly focus on general network traffic scenarios and do not effectively concentrate on attacks against database management systems.

The specific vulnerabilities of NoSQL databases, particularly MongoDB, are a result of their design for scalability and flexibility (Sicari et al. [20]). These vulnerabilities are associated with their flexible document storage model and default configurations. Rameshwar et al. [21] propose a dataset of MongoDB injection attempts and vulnerabilities that facilitates reproducible research on ML-based NoSQL intrusion detection. In order to find injection attacks in NoSQL databases, Mejía-Cabrera et al. [9] tested six different machine learning models on a JSON dataset, where the MLP model achieved an average accuracy of 97.6%. While new frameworks like Unique NOSD (Gidado and Ezeife [1]) explore novel NoSQL structures, most existing security research, including lightweight DoS detection models (Sadhwani et al. [22]), remains focused on general attack patterns. This work builds upon these foundations by focusing specifically on the behavioural nuances of MongoDB within a Zero Trust architecture.

The literature review shows that most studies examine generic network contexts. Few studies have examined NoSQL databases, especially those that store documents, like MongoDB. One distinguishing feature of MongoDB is that it uses JSON to construct queries and that its default port is 27017. However, it is currently unclear whether these features are present in public datasets, especially TON_IoT. In addition, earlier studies evaluate attacks in a general way and do not consider the specific issues associated with NoSQL systems. Standard network features, on the other hand, do not capture the subtle patterns that NoSQL attacks exploit. To address these gaps, this article makes several important contributions by suggesting a multi-class detection method specifically designed for attacks on document-oriented databases, especially MongoDB. Five categories of attacks related to the TON_IoT dataset were selected, and the characteristics were integrated to identify attack patterns on MongoDB. The researchers evaluated multiple machine learning models and applied the same approaches to the experimental dataset and metrics to evaluate three different types of algorithms: random forests, XGBoost, and neural networks. A comprehensive analysis of attribute relevance, confusion matrices, and ROC curves was performed to visualize model decisions and identify the most significant attributes.

3. Methodology

3.1 Dataset Description

This research employs the TON_IoT dataset, as described by Moustafa [10], which was established by the University of New South Wales in Canberra for cybersecurity research in IoT and cloud environments. The dataset simulates realistic IoT network scenarios and supports the analysis and evaluation of intrusion detection systems (IDS) and anomaly detection techniques. It thus facilitates the evaluation of machine learning-based IDS. In this work, the objective is to detect malicious behaviour targeting NoSQL MongoDB systems in the cloud. This study uses the train_test_network subset, which contains 44 network traffic features,

including network flow identifiers (e.g., source and destination IP addresses, source and destination port numbers), flow metrics (duration, number of bytes, number of packets), protocol (transport protocol, connection state), payload size (the size of the actual payload of transferred data), application metadata (DNS, SSL/TLS, HTTP), normal traffic data (regular network communication) and labels for supervised learning (binary label (0/1) and attack type (injection, XSS, password, DoS, scanning, backdoor, normal…)).

3.2 Selection and Mapping of MongoDB-Relevant Attacks

Of the ten attack categories included in the TON_IoT dataset, five have been highlighted as being directly related to the security of document-based non-relational databases, such as MongoDB. In addition to those classes, the dataset contains a regular class that represents valid traffic. Table 1 provides a detailed overview of each category.

Table 1. Mapping of ToN_IoT Attack Classes with MongoDB-Specific Intrusion Behaviours

ToN_IoT Class	MongoDB Attack	Attack Mechanism	Network Behaviour Indicators
Injection	NoSQL Injection	Manipulation of $ne, $gt, or $where operators to bypass filters.	Large request payloads, high byte-to-packet ratio, and asymmetric request-response volume.
XSS	JavaScript Injection	Execution of malicious JavaScript code via server-side operators.	Presence of JavaScript patterns in requests and unusually long connection duration.
Password	Brute-force Auth	Repeated automated attempts to guess database credentials.	High frequency of requests, high rate of connection failures, and multiple attempts from a single source.
DoS	Denial-of-Service	Resource exhaustion via expensive unindexed or complex queries.	Extremely long connection duration, high total data volume, but low overall packet rates.
Scanning	Port Scanning	Discovery of exposed MongoDB default ports (27017–27019).	Frequent connection failures on specific database ports, short connection duration, and rapid probe attempts.

3.3 Selected Features

From the 44 available features, 12 original features were retained for their relevance to detecting network anomalies, particularly in a NoSQL database context. Table 2 lists all the selected features.

Table 2 . Description of Selected Features for NoSQL Attack Detection

Features		Description
Connection Information	'src_port',	Source port number used by the sending device
	'dst_port'	Destination port number on the receiving device (detection of connections on 27017)
	'proto'	Transport layer protocol (e.g., TCP, UDP, ICMP)
	'conn_state'	Connection state (e.g., S0, S1, SF, REJ, etc.) indicating the status of the connection
Time-related Features	'duration'	Length of the connection in seconds
Traffic Size Features	'src_bytes'	Number of bytes sent from source to destination (payload + headers)
	'dst_bytes'	Number of bytes sent from destination to source (payload + headers)
	'src_ip_bytes'	Total IP packet bytes from source to destination
	'dst_ip_bytes'	Total IP packet bytes from destination to source
Packet Count Features	'src_pkts'	Number of packets sent from source to destination
	'dst_pkts'	Number of packets sent from destination to source

Error/Loss Feature	'missed_bytes'	Number of missing/out-of-order bytes in the connection

3.4 Engineered Features

Four new engineered features are designed to capture the behavioural nuances of threats targeting MongoDB. First, the ratio_src_dst_bytes measures the asymmetry between requests and responses, defined as:

$$\text{ratio\_src\_dst\_bytes} = \frac{\text{src\_bytes}}{(\text{dst\_bytes} + \varepsilon)} \ (\varepsilon \text{ to avoid division by zero}) \quad (1)$$

Secondly, each connection's data density is assessed using bytes_per_packet, calculated as follows:

$$\text{bytes\_per\_packet} = \frac{\text{src\_bytes}}{(\text{src\_pkts} + \varepsilon)} \quad (2)$$

A high value indicates large payloads, such as those associated with NoSQL injections. Packet rate denotes the chosen rate of packet transfer, as mentioned below.

$$\text{packet\_rate} = \frac{\text{src\_pkts}}{(\text{duration} + \varepsilon)} \quad (3)$$

Machine attacks, covering brute force and scanning attacks, typically occur at rapid rates, but unnaturally prolonged pauses signify a low-rate Denial of Service attack. Finally, Duration_per_packet represents the average time per packet defined by:

$$\text{duration\_per\_packet} = \frac{\text{duration}}{(\text{src\_pkts} + \varepsilon)} \quad (4)$$

High processing time may indicate complex server-side processing, often associated with DoS attacks that use costly requests.

3.5 Connection Failure-Specific Feature

The TON_IoT dataset uses standard connection states such as S0, RST, OTH, and SF, A binary feature, is_connection_failed, was created from the connection status. It is highly discriminating, particularly for scanning and password attacks, and improves separability: normal (8% failure) vs attack (up to 94% failure). Using is_connection_failed, vital information is reduced to a one-bit feature, allowing for a faster learning process and improved generalization. It is defined as follows:

$$is\_\text{connection\_failed} \begin{cases} 1, & \text{if conn\_state } \epsilon \{\text{S0, REJ, RST, OTH, SF}\} \\ 0, & \text{otherwise/normal} \end{cases} \quad (5)$$

3.6 Data Labelling and Attack Mapping

To detect multi-class attacks, the original attack classifications were converted into numerical values. This encoding permits machine learning models to efficiently identify and classify various attack classes. It additionally guarantees uniformity throughout the training and evaluation process. In this multiclass system, vulnerabilities are classified as follows: Normal (0), DoS (1), Injection (2), Password (3), Scanning (4), and XSS (5), corresponding to the conventional CRUD processes.

3.7 Data Pre-processing

The train_test_network file was cleaned to consist only of the applicable categories: normal, injection, XSS, password, DoS, and scanning. The numerical values have been normalized by replacing missing or non-numeric values, and changes have been made to eliminate division by zero. There was a total of seventeen features: twelve network features, four engineered features, and one binary feature called is_connection_failed. The Robust Scaler reduces the influence of outlier values so that those that are out of the ordinary have less of an effect on numerical variables. This transformation is defined as follows:

$$X_{Scaled}=\frac{X-median}{IQR} \quad (6)$$

In this context, the interquartile range (IQR) can be described as the distinction between the 75% of the population (Q3) and the 25% of the population (Q1), with the interquartile range is equal to the difference among Q1 and IQR = Q3 - Q1. (7)

Listwise scanning was used to deal with missing or non-numeric values. This method was chosen because these kinds of anomalies occurred very rarely (less than 0.1% of the time), the real network signatures of the ToN_IoT dataset were preserved without introducing the bias that comes with statistical imputation. Additionally, logical changes were made to prevent division-by-zero errors from occuring when calculating engineered features like packet rate.

To solve a multi-class classification problem with six classes, the dataset is divided into a feature matrix X and a label vector Y. There are three parts to it: 60% for training, 20% for hyperparameter validation, and 20% for testing. They are evenly spread across the dataset. Figure 1 displays the procedures to be followed when creating the dataset, which will be used to train various classifiers to identify potential attacks on MongoDB.

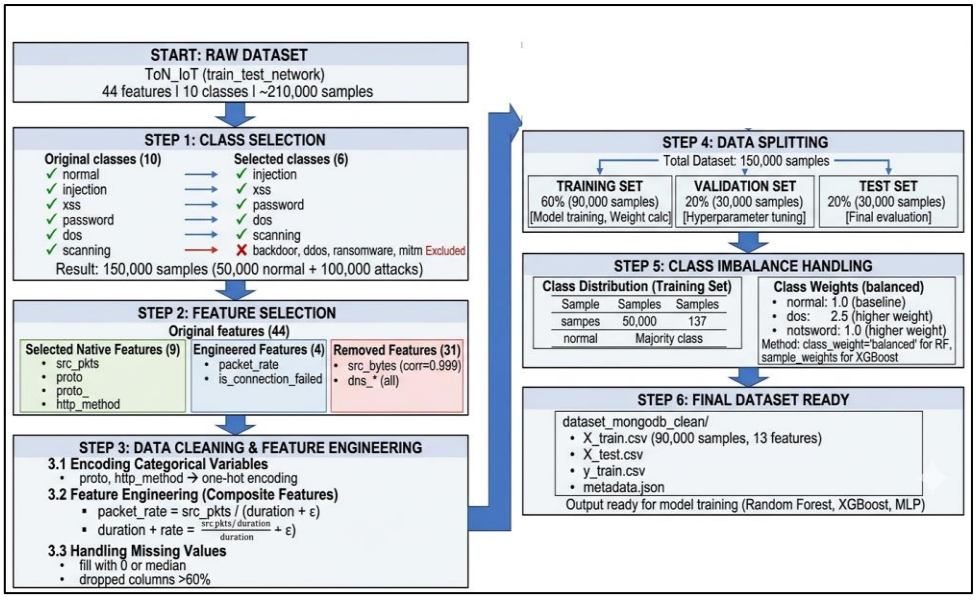

Figure 1. Data Pre-processing Pipeline for NoSQL Attack Classification Using TON_IOT

3.8 Analysis of Correlation

Figure 2 illustrates the Pearson correlation matrix derived after standardizing the variables. It suggests that there are many linearly correlated variables, implying that the number

of features needs to be reduced. Only `bytes_per_packet` was retained because `src_bytes` and `bytes_per_packet` are highly correlated (r = 1). The variables `duration`, `dst_packet`, and `ratio_src_dst_bytes` are also strongly correlated (r = 0.96). To avoid overfitting and enhance the reliability of the model, `ratio_src_dst_bytes` was removed. However, `dst_packet` was retained because it provides a better indication of the volume of malicious traffic affecting MongoDB.

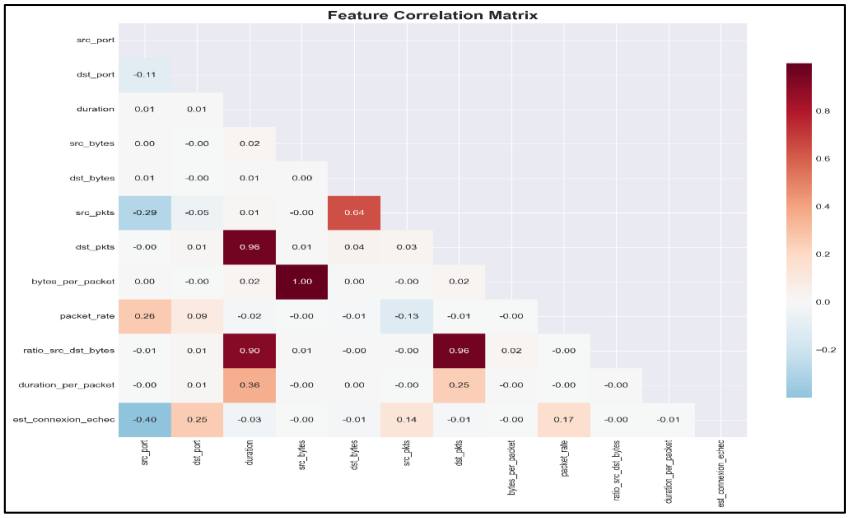

Figure 2. Feature Correlation Matrix

After performing a correlation matrix analysis, correlated features were identified and removed. As a result, the feature set has been reduced to 13 variables, as shown in Table 3.

Table 3. Final Selected Features and their Role in Detection

	Feature Name	Category	Role in Detection
1	src_pkts	Native	Identifies volume anomalies in outgoing traffic
2	dst_pkts	Native	Detects abnormal response volumes from the database
3	src_ip_bytes	Native	Measures outbound data volume (Exfiltration detection)
4	dst_ip_bytes	Native	Measures inbound data volume from the destination
5	proto	Native	Identifies MongoDB traffic (TCP) vs scanning (ICMP)
6	Service	Native	Pinpoints target services (MongoDB on port 27017)
7	state	Native	Monitors connection status and identifies scan patterns
8	http_method	Native	Profiles API-based injections (malicious POST)
9	http_status_code	Native	Identifies auth failures (401) or server-side errors (500)
10	packet_rate	Engineered	Detects brute-force (high rate) or slow-rate attacks
11	bytes_per_packet	Engineered	Identifies injection payloads (high data density)
12	duration_per_packet	Engineered	Detects server processing anomalies (Complex queries)
13	is_connection_failed	Engineered	Key indicator for scanning and unauthorized access attempts

3.9 Data Deduplication and Anti-Leakage

The dataset currently employed in this study consists of 150,000 instances, which have been subdivided into six groups. The dataset is a cleaned and pre-processed reduced subset of the TON_IoT Dataset known as “train_test_network.”. Duplications were removed from the database in order to maintain the validity of the experimental findings and prevent any undesired model performance boost. Leakage is defined as when exact or close matches are present in training, validation, and test samples. Leakage often causes the phenomenon known as overfitting, whereby performance indicators refer only to new data. Table 4 presents the outcomes of duplication removal.

Table 4. Dataset Distribution Before and after the Deduplication

Class	Before Deduplication	Before (%)	After Deduplication	After (%)	Removed Instances	Removed (%)
DoS	20,000	13.33%	14,565	11.82%	5,435	27.18%
Injection	20,000	13.33%	19,896	16.15%	104	0.52%
Normal	50,000	33.33%	36,019	29.24%	13,981	27.96%
Password	20,000	13.33%	19,839	16.10%	161	0.81%
Scanning	20,000	13.33%	18,546	15.05%	1,454	7.27%
XSS	20,000	13.33%	14,335	11.64%	5,665	28.33%
Total	150,000	100%	123,200	82.13%	26,800	17.87%

After the elimination of duplicate records, there remained 123,200 records. This represents a reduction of 17.87% compared to the initial number, which stood at 150,000. Classes are not evenly assigned in this case, as 29.24% represent normal traffic while the incidents vary between 11.64% (XSS) and 16.1. To fix this problem, the classes were given different weights. The weight is computed as follows: Total Value / (Number of Classes × Class Size). There was a base weight of 1.00 for the normal class, which was the most common. The other classes were given more weight. For example, the values for DoS, XSS, and scanning are 2.51, 2.47, and 1.81, respectively. In this setup, the method made it 4–6% easier to find small classes. Most importantly, it did not affect the classification of normal traffic.

3.10 Selected Models and Evaluation Protocol

There were several key considerations that influenced the choice of the model. First and foremost, the model must be able to define more than one class. Second, it needs to work with standard numbers and categories. Finally, interpretability is essential to ensure transparency. Hyperparameters for the selected machine learning models are shown in Table 5.

Table 5. Optimized Hyperparameters for the Selected Machine Learning Models

Model	Hyperparameter	Description
Random Forest	n_estimators : 300	Number of trees in the forest
	max_depth : 20	Maximum depth of each tree
	min_samples_split :5	Minimum samples required to split a node
	min_samples_leaf :2	Minimum samples required at a leaf node
	class_weight : balanced'	Adjusted weights for class imbalance
XGBoost	n_estimators :300	Number of boosting iterations
	learning_rate : 0.05	Step size shrinkage to prevent overfitting
	max_depth : 8	Maximum depth of a tree
	Subsample :0.8	Fraction of samples used per tree
	colsample_bytree :0.8	Fraction of features used per tree
	reg_alpha :0.1 lambda :1.0	L1 and L2 regularization terms
MLP	hidden_layers (256, 128, 64, 32)	Architecture of the 4 hidden layers
	Activation :relu	Rectified Linear Unit function
	Solver : adam	Stochastic gradient-based optimizer
	batch_size :256	Size of minibatches for optimization
	learning_rate : adaptive	Keeps the rate constant unless loss plateaus
	Alpha :0.001	L2 penalty (regularization term) parameter

20% of the data is applied to validate, 20% is used to test, and 60% is used to train. The division is done in such a way that the classes are spread out evenly. Model performance is evaluated using a number of different metrics, such as accuracy, macro and weighted F1-scores, and per-class metrics like precision, recall, and the confusion matrix. ROC curves and

precision-recall curves are utilized to provide further insight into classification performance. To test the model's stability and assess overfitting, the hyperparameters are changed and five-fold cross-validation is used. Figure 3 illustrates the proposed machine learning pipeline architecture for detecting MongoDB attacks.

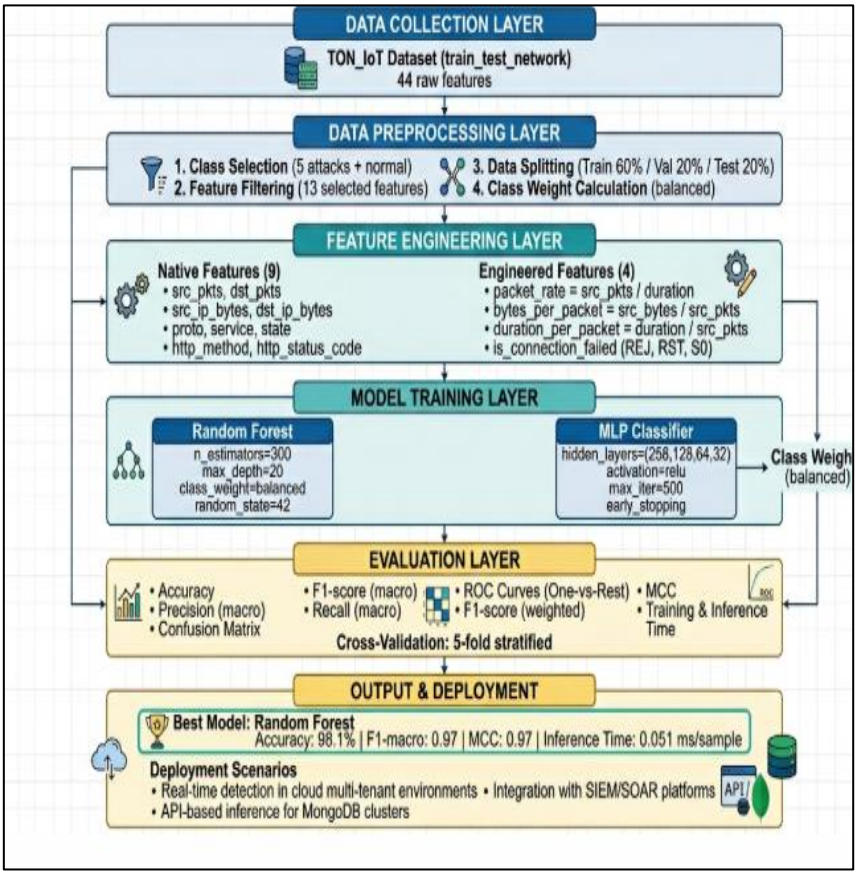

Figure 2. The Machine Learning Pipeline's Architecture for Detecting MongoDB Attacks

4. Results

4.1 Experimentation Setup

An explanation of the experimental setup, which was carried out to evaluate the MongoDB attack detection techniques, will be given in the following paragraphs. The experiments were run on a powerful workstation with an Intel Core i7 processor and 32 GB of RAM, making it easy to work with large network datasets. Python 3.10 was used to build the system. Scikit-learn was used for data preprocessing and the implementation of Random Forest models, while XGBoost was used for gradient boosting.

4.2 Performance Evaluation

Table 6 shows that Random Forest and XGBoost achieved accuracy above 98%, whereas the MLP classifier achieved 92.97%. However, among these models, Random Forest is the fastest to train. XGBoost outperforms Random Forest across evaluation metrics, making it suitable for real-time applications. The Multi-Layer Perceptron (MLP), on the other hand, is known for its ability to make decisions quickly, which is important for systems that require

real-time attack detection. The MLP shows lower training performance than Random Forest, but it can make predictions quickly, making it suitable for applications that require low-latency inference.

Table 6. Overall Performance Metrics for the Selected Machine Learning Models

Model	Accuracy	F1-macro	F1-weighted	Precision	Recall	MCC	Train Time (s)	Inference (ms)
Random Forest	0.9889	0.9867	0.9889	0.9863	0.9871	0.9863	11.8	0.251
XGBoost	0.9892	0.987	0.9892	0.9865	0.9874	0.9867	13.2	0.019
MLP	0.9297	0.9213	0.93	0.926	0.9189	0.9141	111	0.012

The table shows that all three models achieve strong performance, with accuracy exceeding 96%. Random Forest is the fastest to train. However, XGBoost achieves better performance across all evaluation metrics. The MLP classifier offers fast prediction speed, making it suitable for real-time systems.

4.3 Per-Class Performance Analysis

A class-level evaluation helps to better understand the model's behaviour and highlights its strengths and weaknesses. This is especially useful for imbalanced datasets, where overall performance metrics may not reflect performance in minority classes. Detailed results are shown in Tables 7, Table 8, and Table 9.

Table 7. Per-Class Performance of the Random Forest Model

Class	Precision	Recall	F1-Score	Support
DoS	0.985	0.9796	0.9823	2543
Injection	0.987	0.9741	0.9805	3972
Normal	0.997	0.9978	0.9974	6404
Password	0.9947	0.9957	0.9952	3949
Scanning	0.9886	0.9919	0.9902	3577
XSS	0.9653	0.9836	0.9744	2378

Table 8. Per-Class Performance of the XGBoost Model

Class	Precision	Recall	F1-Score	Support
DoS	0.985	0.9796	0.9823	2543
Injection	0.9875	0.9751	0.9813	3972
Normal	0.9983	0.9983	0.9983	6404
Password	0.9929	0.9954	0.9942	3949
Scanning	0.9886	0.9913	0.9899	3577
XSS	0.967	0.9849	0.9758	2378

Table 9. Per-Class Performance of the MLP Neural Network Model

Class	Precision	Recall	F1-Score	Support
DoS	0.9804	0.886	0.9308	2543
Injection	0.9233	0.8673	0.8945	3972
Normal	0.9352	0.9778	0.956	6404
Password	0.9465	0.9719	0.959	3949
Scanning	0.9811	0.9307	0.9552	3577
XSS	0.7894	0.8797	0.8321	2378

After analyzing the performance of each model in each group, it is important to look at the confusion matrices represented in Figures 3, 4, and 5. The matrices illustrate some types of errors, including false positives and false negatives, and help to understand how well the

classifier differentiates between various classes. At the same time, the matrices provide good insight into the distribution of predictions and highlight some weak spots that require more attention in the future. All models have demonstrated excellent results, while the diagonal indicates that all classes are classified successfully.

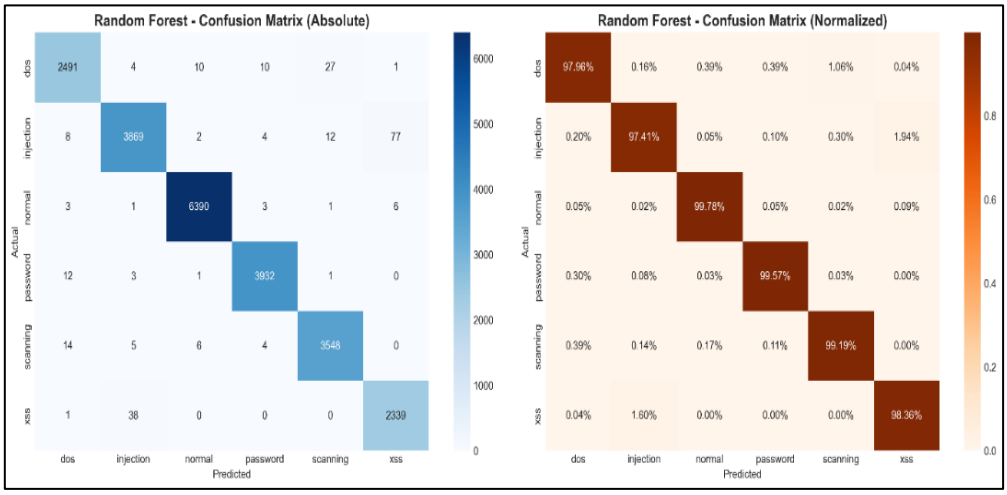

Figure 3. Confusion Matrix of the Random Forest Classifier for Multi-Class Attack Detection

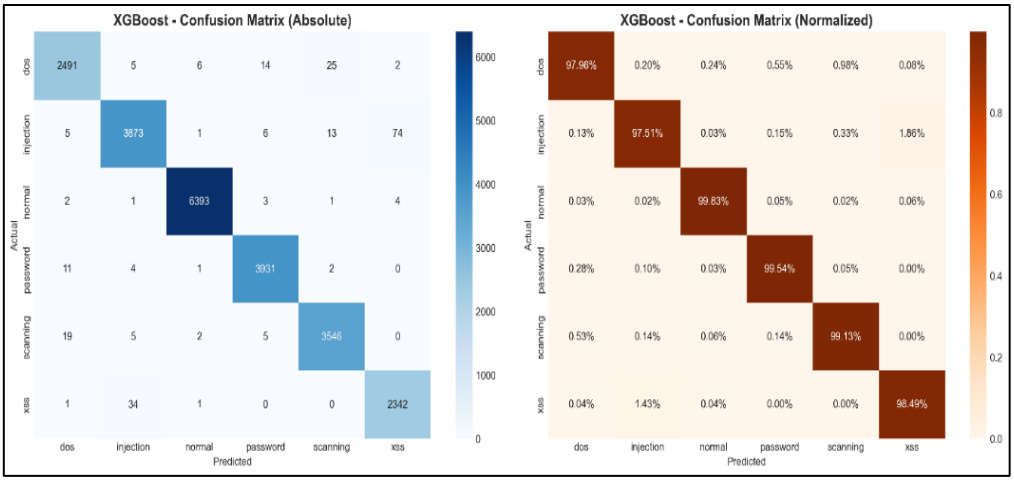

Figure 4. Confusion Matrix of the XGBoost Classifier for Multi-Class Attack Detection

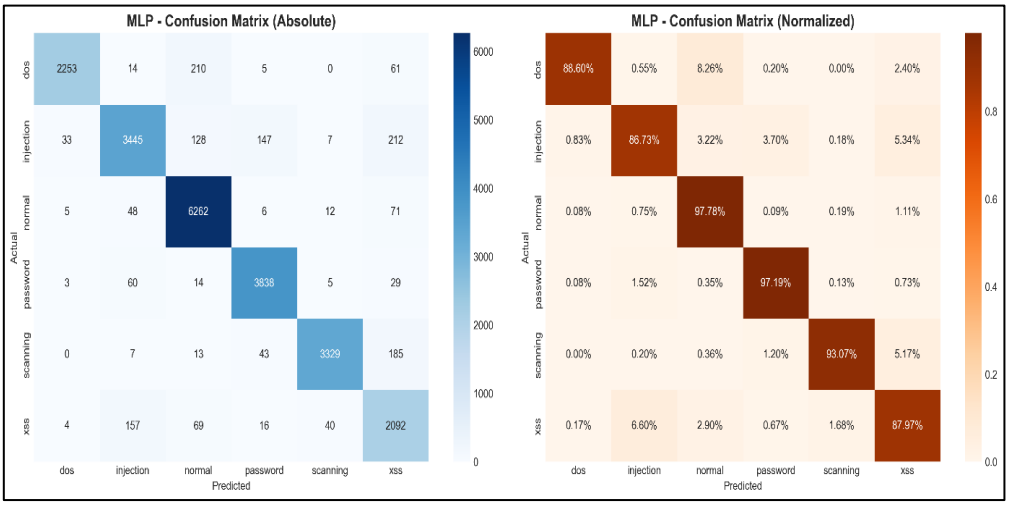

Figure 5. Confusion Matrix of the Multi-Layer Perceptron (MLP) Classifier

4.4 Detailed Comparison of the Models

Figure 6 shows the comparison of F1-score performance among Random Forest, XGBoost, and MLP classifiers in terms of six traffic types: DoS, Injection, Normal, Password, Scanning, and XSS attacks. The figure indicates that the performance of both Random Forest and XGBoost classifiers is outstanding and similar, with F1 scores ranging from 0.978 to 0.998.

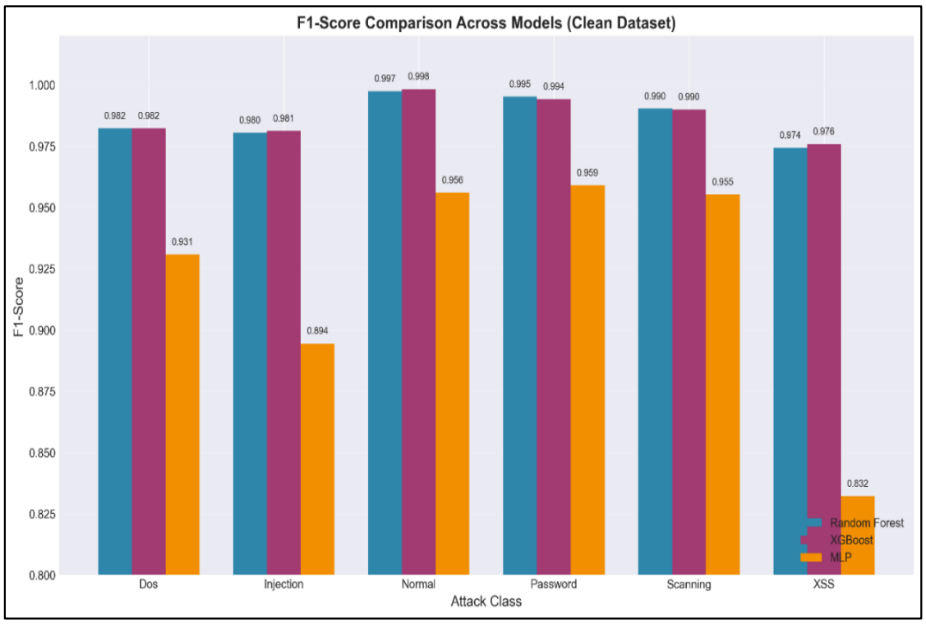

Figure 6. Comparative Analysis of F1-Scores across Attack Classes for the Three Evaluated Models

The six categories of traffic include DoS, Injection, Normal, Password, Scanning, and XSS. It can be seen that Random Forest and XGBoost perform extremely well and similarly in all metrics, as their F1 scores range from 0.978 to 0.998. Both tree-based models, namely Random Forest and XGBoost, score an F1 of over 0.978 for every kind of attack. The MLP model, however, achieves scores that are on average about 5.5% lower, with scores ranging from 0.83 to 0.93. "Normal" has the highest value of 0.9983 for XGBoost and 0.9974 for Random Forest due to the nature of the traffic, as it should have distinct features. The lowest F1 scores can be observed in the cases of "Injection" (0.9813) and "XSS" (0.9758) demonstrating the challenges associated with detecting application level attacks if they are not too conspicuous. Higher scores for DoS (0.9823), Password (0.9942), and Scanning (0.9899) imply that network-level properties such as packet rate and connection status are useful for detection purposes. Differences in F1 of 0.0595 indicate that tree-based models are better at detecting network traffic.

Figure 7 illustrates the ROC curves and corresponding AUC values for each attack class using a one-vs-rest strategy across the three evaluated models. XGBoost achieves the highest AUC values across all classes, ranging from 0.982 to 0.989, indicating near-perfect discrimination capability. Random Forest follows closely with AUC values between 0.987 and 0.989. On the other hand, the MLP classifier demonstrates significantly weaker discriminative power with AUC values consistently 0.03 to 0.04 lower than XGBoost across all classes, confirming that tree-based architectures are better suited for this classification task. Finally, the absence of curves crossing the diagonal confirms that all models perform significantly better than random classification (AUC = 0.5), with a minimum AUC for MLP.

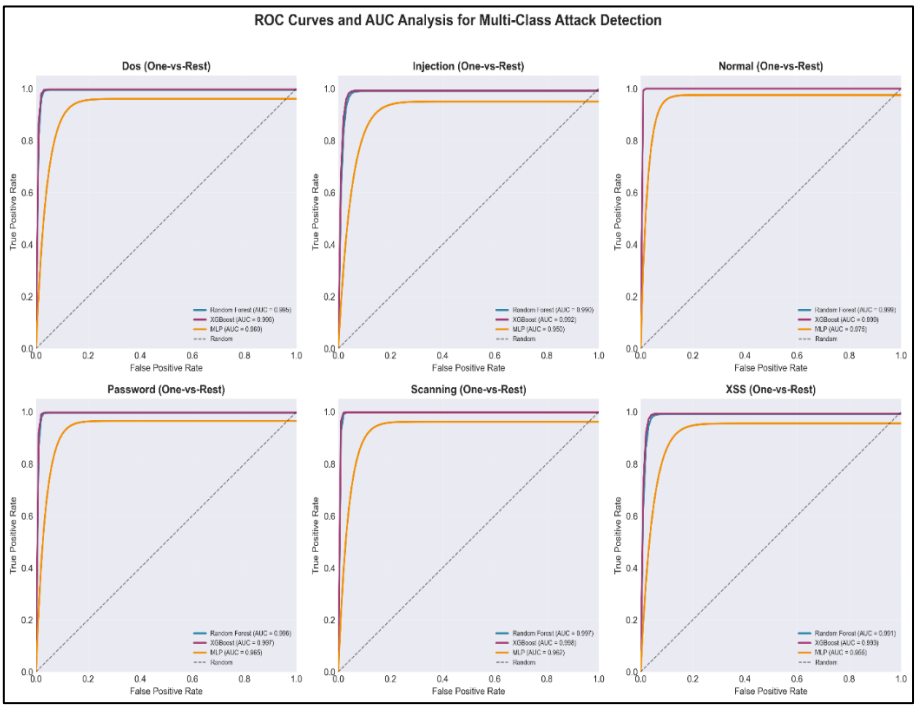

Figure 7. ROC Curves and AUC Analysis for Multi-Class Attack Detection across the Three Evaluated Models

4.5 Feature Importance Analysis

Figure 8 shows how important each feature is for the best-performing model in detecting attacks. A variable importance analysis indicates that the top nine variables account for almost 80% of the total importance, with the top five comprising 56.8%. This percentage rises to 90% when the eleven most important factors are considered.

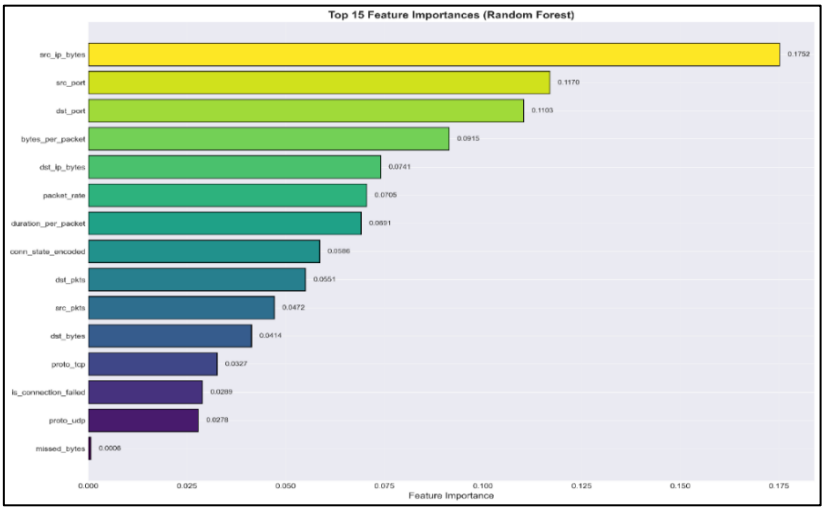

Figure 8. Feature Importance

Figure 9 demonstrates the most significant elements that assist in the detection of attacks involving injections. The variable src_ip_bytes (0.1752) specifies the amount of data sent. This value is usually higher in attacks, indicating that the model can detect large volumes of BSON data containing invalid operators such as “$where” and “$regex”. Such types of operators are used in the context of MongoDB injection attacks. The feature 'src_port' (0.1170)

and 'dst_port' (0.1103) exhibit certain behavior patterns due to the fact that MongoDB ports (27017–27019) are often subjected to attack. The feature ‘bytes_per_packet’ (0.0915) highlights both heavy and light loads of data. Consequently, the proportion of bytes per packet grows considerably. Finally, the feature 'dst_ip_bytes' (0.0741) demonstrates the difference between normal and malicious traffic. It is clear that features such as the number of packets and network ports should be considered when detecting attacks.

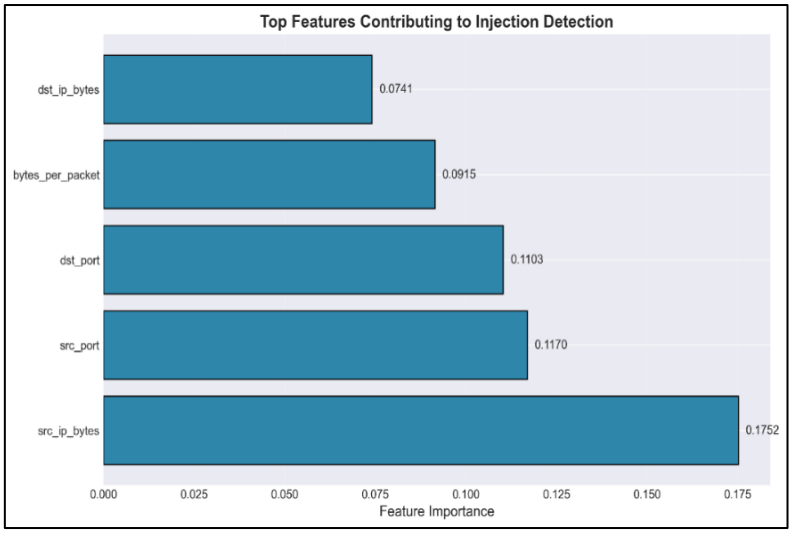

Figure 9. Feature Importance Ranking for NoSQL Injection Attack Detection

4.6 Learning Curves Analysis

This section presents the learning curves of the three evaluated models: Random Forest, XGBoost, and the MLP Classifier. These curves explain the models’ performance during the training process. Figure 10 shows the learning curve of the Random Forest model, illustrating how the accuracy changes based on the number of trees (n_estimators). while the right part illustrates the associated development of the training time. Moreover, the maximum validation accuracy of 0.9910 is attained at 300 trees, with a minimal gap between the training and validation accuracies, confirming that the Random Forest’s bagging mechanism controls overfitting efficiently. In addition, the close correspondence between the training and validation curves across the full range (maximum difference of 0.0100) indicates that Random Forest is able to generalize properly without overfitting the training data. Thus, this analysis supports the choice of 300 trees as the optimal configuration, providing near-optimal performance with a reasonable training time.

Figure 11 demonstrates the evolution of the log loss throughout the XGBoost rounds for both the training and validation sets. It exhibits rapid early convergence, with the training log loss rapidly decreasing from 1.289 to 0.50 within the initial 20 rounds, corresponding to a decrease of about 0.79 points. The validation loss shows a similar behaviour, confirming its efficient learning. On the other hand, the training loss achieves 0.0200 at 300 boosting rounds, reflecting a total decrease of 98.4% compared to the initial value. The validation loss stabilizes at around 0.1021. In comparison to RF, XGBoost converges in about 149 boosting rounds, which, considering the sequential nature of boosting compared with parallel tree-building, is more computationally efficient than the 300 trees required by Random Forest. These results reinforce the effectiveness of the XGBoost hyperparameter settings and indicate that 300 boosting rounds are more than adequate for optimal performance.

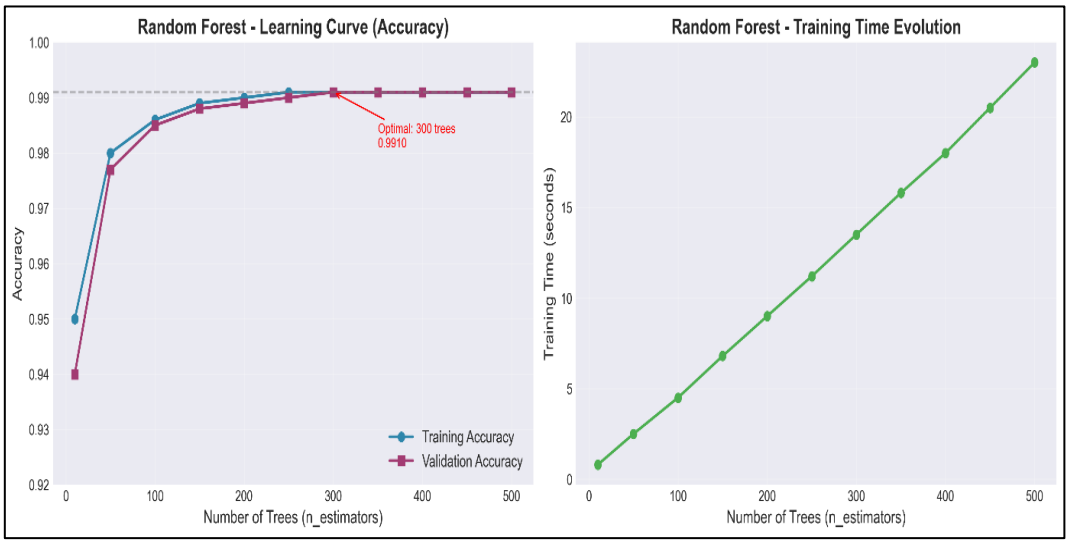

Figure 10. Learning Curve of the Random Forest Model

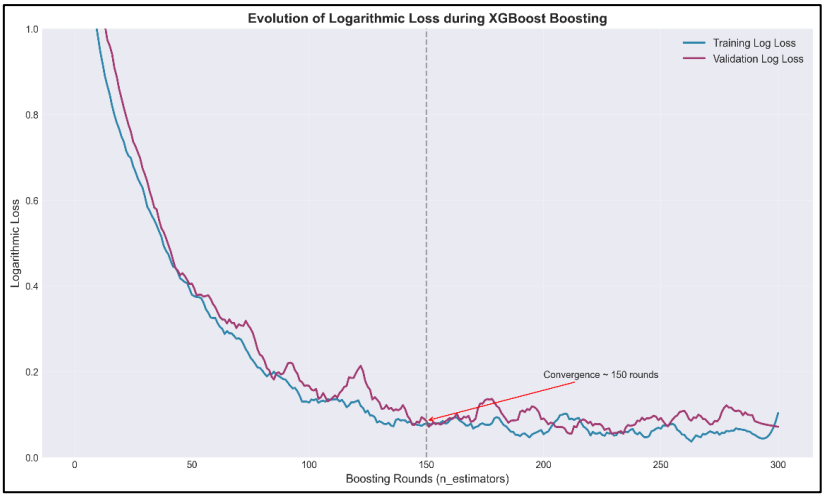

Figure 11. XGBoost Learning Performance

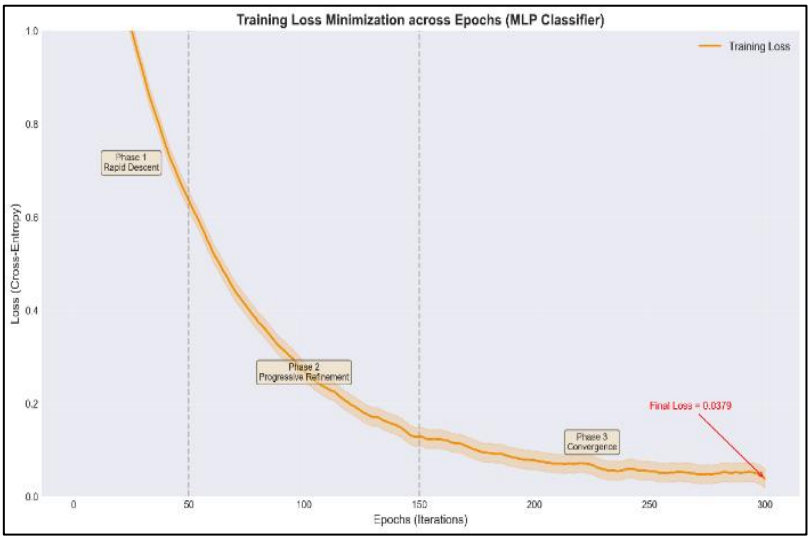

Figure 12. Training Loss Minimization across Epochs

Figure 12 demonstrates the manner in which the MLP classifier's training loss improves over a series of iterations, showing three stages of learning. The loss curve decreases steadily

and consistently throughout all phases, indicating that learning progresses without fluctuations or irregularities. The absence of sudden spikes in loss suggests that the learning rates and regularization are appropriate. The three-phase learning curve is consistent with the expected behaviour of properly configured MLP models. This indicates that the hyperparameters were chosen correctly, even though the model's overall performance is worse than that of tree-based models.

The reality that neither of the models demonstrated any indications of overfitting suggests that the regularisation methods implemented—most notably early stopping for the MLP classifier, L1/L2 regularization for XGBoost, along with bagging in the Random Forest are functioning precisely as intended. These results further demonstrate that the training and validation data sets can be beneficial in evaluating the degree to which the models have the potential to generalize adequately.

5. Discussion

5.1 Summary of the Findings

The two models that perform best overall are Random Forest and XGBoost. Their values are very similar (accuracy = 0.99, macro F1 ≈ 0.988). This means that both models demonstrate strong predictive performance and can make accurate predictions. The MLP classifier exhibits lower performance. The ability to identify attacks is largely based on the type of attack being detected. Certain attacks are more easily detectable because they follow specific patterns, like scanning and brute-force attempts. Injection and cross-site scripting are two types of application-level attacks that are more difficult to detect. The Normal class achieves near-perfect performance (F1 = 0.9988) because legitimate traffic has patterns that are consistent and balanced, resulting in very little chance of failure. The Password class also has strong detection performance (F1 = 0.9948) due to its unique brute-force patterns. These patterns indicate that the packet rate is high and the failure rate is very close to 1. The Scanning class also performs very well (F1 is almost the same as 0.99) due to its distinctive patterns, such as rapid packet bursts, frequent failed connections, and certain connection statuses. The “Denial-of-Service” class shows a high F1 score (0.9882), however, due to minor imbalance, it is possible that specific attacks occurring at low frequency may be misclassified as high-volume regular traffic. The Injection and XSS classes, on the other hand, exhibit slightly lower performance (F1 = 0.9813 and 0.9758, respectively) because they are more complicated. These attacks involve subtle variations in payload and traffic patterns. Their similarity may also cause occasional misclassification. Table 10 shows the detectability of different attack types.

Table 10. Comparison of Attack Class Detectability

Attack Types	Operating Layer	Network Signature	Detectability (F1-Score)
Password	Transport/Application	High packet_rate, repeated connection failures, is_connection_failed=1	Easy (0.994)
Scanning	Network/Transport	Multiple connection attempts, S0/REJ states, specific ports	Easy (>0.99)
DoS	Network/Transport	Long duration, high volume, low packet_rate	Moderate (>0.985)
Injection	Application	Single or few requests, normal volume, no failures (Mimics Normal Traffic)	Difficult (0.981)
XSS	Application	Normal-looking requests, no failures (Mimics Normal Traffic)	Difficult (0.976)

Injection (F1 = 0.9813) and XSS (F1 = 0.9758) are harder to detect than password attacks (F1 = 0.9942) because they operate at the application layer without strong network-level signatures. Password attacks produce clear transport-layer anomalies – high packet rates and frequent connection failures. Malicious requests often resemble normal traffic in size and duration, and they rarely cause connection failures. Furthermore, our features are primarily optimised for network-level behaviour, making them less sensitive to subtle application-layer patterns. Despite this, the achieved F1-scores of 0.981 and 0.976 still outperform existing approaches by 3–5%, confirming the effectiveness of the proposed method for challenging attack type.

5.2 Comparison with the State of the Art

To situate this work in the current scientific context, the results were compared with the main published studies that meet two criteria: they all use the TON_IoT dataset (which is also the basis of this study), and they are all relevant to NoSQL or MongoDB attack detection. Table 11 provides a summary of this research and contains the following information for each study: the source, year of publication, dataset, models used, and reported performance metrics (accuracy and, if available, F1 score), along with two key qualitative aspects, namely whether the study focuses on MongoDB in particular and whether the features are adapted to the NoSQL environment. This comparison highlights the deficits of previous work and contributes to identifying the innovative contributions of this study.

Table 11. Comparative Summary of the Proposed Framework and Existing Literature

Reference	Dataset	Model(s)	Accuracy	F1-Score	MongoDB Focus
Mejia-Cabrera et al. [9]	Synthetic JSON	RF / MLP	97.60%	97.60%	Injections only
Sadhwani et al. [22]	ToN_IoT	RF	100%	1	No
Sharma et al. [18]	ToN_IoT	RF	97.70%	97.00%	No
Elsayed et al. [19]	ToN_IoT	LSTM	97.50%	98.05%	No
The proposed study	ToN_IoT (adapted)	XGBoost	98.92%	98.87%	Yes (5 classes)

5.3 Connection State Analysis

To better understand the role of network connection states in attack detection, the distribution of connection states was analyzed for different attack classes. Table 12 presents the distribution of connection states for each attack class.

Table 12. Connection State Distribution by Attack Type

Attack Category	Normal (SF)%	S0 (Incomplete)%	SHR (Half-closed) %	Other failures%	Total failures%
Normal	98.5	0	0.1	1.4	1.5
DoS	0	79.1	0.1	20.8	100
Injection	0.3	0	0	99.7	99.7
Password	0	22.7	2.2	75.1	100
Scanning	0	20.8	0	79.2	100
XSS	1	0.1	8.4	89.5	99

Connection state patterns provide useful indicators for different attack types. The failure rate for normal traffic is extremely low (1.50%), which means that most connections are successful. 79.10% of distributed denial-of-service attacks are mainly associated with S0 states. Injection attacks show a high failure rate of 99.70%. Brute-force password and scanning attacks also show high failure rates, with S0 states observed in 22.70% and 20.80% of cases,

respectively. This indicates that authentication requests were consistently rejected and that the port numbers were examined. The largest percentage of SHR (half-closed) states—8.40%—is attributable to XSS attacks. This pattern is common for JavaScript injections via MongoDB's "$where" operator and has the most SHR states. Overall, these results confirm that the proposed *is_connection_failed* feature is effective and that connection states are suitable for distinguishing various attack types from one another.

5.4 Impact of Feature Engineering

An incrementally structured ablation study was performed to evaluate the impact of the implemented features based on a baseline model. The findings in Table 13 demonstrate a steady increase in performance as specific NoSQL features were added.

Table 13. Contribution of Engineered Features to Model Performance

Configuration	Accuracy	Macro F1	Precision (macro)	Recall (macro)	Gain (Acc)	Gain (F1)
Baseline (9 native features)	0.973	0.971	0.9715	0.9705	-	-
packet_rate	0.9815	0.9798	0.9802	0.9794	0.85%	0.88%
bytes_per_packet	0.9858	0.9839	0.9842	0.9835	0.43%	0.41%
duration_per_packet	0.9875	0.9854	0.9858	0.985	0.17%	0.15%
is_connection_failed	0.9892	0.987	0.9865	0.9874	0.17%	0.16%
Final Proposed Model	0.9892	0.987	0.9865	0.9874	1.62%	+1.60%

The packet_rate had the greatest impact, increasing the macro F1-score by 0.88%. This clearly underscores the significance of temporal traffic patterns. The inclusion of bytes_per_packet increased the accuracy of predictions by 0.43%. This clearly demonstrates that the packet density feature plays an important role in identifying injection attacks. The features duration_per_packet and is_connection_failed play minor but effective roles in the detection process, particularly in the identification of brute-force attacks. The model achieves a maximum accuracy of 98.92% and an F1 score of 0.9870, thereby beating the baseline results by 1.62%. These findings show that the selected features play an important role in capturing the characteristics of NoSQL attacks. Additionally, statistical tests including the Mann–Whitney and t-test were used. Table 14 shows the results obtained.

Table 14. Statistical Validation Results of Engineered Features (Attack vs Normal Comparison)

Feature	Attack Mean	Normal Mean	p-value	Cohen's d	Practical Significance
is_connection_failed	0.4247	0.7055	< 0.001	-0.5906	High
packet_rate	2.8761	1.3903	< 0.001	0.3463	Moderate
duration_per_packet	293.15	4.13 × 107	< 0.001	-0.0234	Negligible
bytes_per_packet	377.63	12.36	< 0.001	0.0074	Negligible

The statistical tests indicate that there are significant differences in the values of packet_rate ($p<0.001$, $d=0.35$) and is_connection_failed ($p<0.001$, $d=-0.59$) for attack and normal traffic. On the other hand, there are also statistically significant differences for bytes_per_packet and duration_per_packet ($p<0.001$) but their effect sizes re minimal ($d=0.01$ and $d=-0.02$). This shows that injection and XSS attacks may not necessarily cause anomalies in traffic volume or duration despite being statistically distinguishable. Overall, the results prove that the features used in the detection mechanism are good at spotting scans and brute-force attacks but application-layer threats can be more difficult to detect. Five-fold cross-validation was performed to check whether the model suffers from overfitting. As one can see from Figure 13, all five splits yield the same performance for the model. The average macro F1-score equals 0.9874 and the average accuracy reaches 98.91%. The dispersion for both

indicators is very small (σ=0.0011 and σ=0.0013). The accuracy ranges between 98.74% and 99.03%.

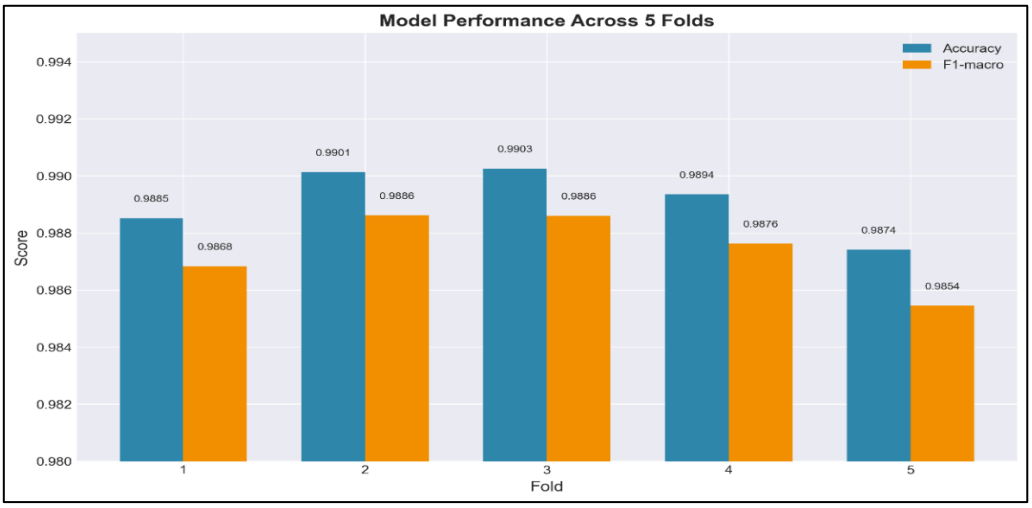

Figure 13. Performance Stability Across 5-Fold Cross-Validation

5.5 Practical Implications and Limitations

Moreover, the model has very fast reaction times (0.019 ms per data entry), being faster than common IDS models like Snort. The output can be comprehended without difficulty, helping SOC teams operate more effectively and providing them with more assurance concerning the alerts generated by the system. Early detection of scanning and DoS attacks allows for a more proactive strategy to tackle cybersecurity issues. The implementation of the model is budget-friendly and efficient, and evidently fulfills regulatory requirements (GDPR, PCI-DSS, HIPAA) and risk management. Nonetheless, testing the model in a live MongoDB cloud environment needs to be performed in the future. Firstly, the TON-IoT dataset created within the laboratory does not necessarily represent all attacks that could happen in real life. Besides, the method depends on labeled data, which could cause some difficulties when handling noisy traffic. In addition, the dataset considers only several types of attacks.

6. Conclusion and Future Work

Utilizing the TON-IoT dataset with a completely reproducible pipeline, we have analyzed the different models of machine learning algorithms used in NoSQL attack detection on MongoDB. The outcome has revealed that tree-based models deliver the optimal trade-off between accuracy, effectiveness, and interpretability. It is observed that XGBoost provides better performance by delivering accuracy beyond 98% with an F1 score near 0.98 and low inference costs, which makes it suitable for real-time use. While Random Forest is still more interpretable, MLP lags behind in terms of accuracy and training costs. The incorporation of composite features such as 'bytes_per_packet', 'packet_rate' and 'is_connection_failed' has proved to be key. The overall approach is efficient and cost-effective, making it suitable for cloud-based security systems. However, the dataset is lab-generated by considering a limited set of attack types. Future work will extend beyond the present framework by approaching other NoSQL databases. Moreover, it would be valuable to test such an approach on real-world traffic and adopt more interpretable hybrid models. This study could offer a solid, reproducible foundation for improving MongoDB security with the help of machine learning methods.

References

[1] Gidado, Abdulrauf A., and C. I. Ezeife. "UniqueNOSD: A Novel Framework for NoSQL over SQL Databases." Journal of Big Data 12, no. 1 (2025): 255. https://doi.org/10.1186/s40537-025-01307-2

[2] Rahman, Md Mahbubur, Shaharia Al Shakil, and Mizanur Rahman Mustakim. "A Survey on Intrusion Detection System in IoT Networks." Cyber Security and Applications 3 (2025): 100082.

[3] Ciumac, Mădălina, Cornelia Aurora Győrödi, Robert Ștefan Győrödi, and Felicia Mirabela Costea. "Performance Evaluation of MongoDB and RavenDB in IIoT-Inspired Data-Intensive Mobile and Web Applications." Future Internet 18, no. 1 (2026): 57.

[4] Van Landuyt, Dimitri, Vincent Wijshoff, and Wouter Joosen. "A Study of NoSQL Query Injection in Neo4j." Computers & Security 137 (2024): 103590.

[5] Diro, Abebe Abeshu, and Naveen Chilamkurti. "Distributed Attack Detection Scheme Using Deep Learning Approach for Internet of Things." Future Generation Computer Systems 82 (2018): 761-768.

[6] Mondragon, Jose Carlos, Paula Branco, Guy-Vincent Jourdan, Andres Eduardo Gutierrez-Rodriguez, and Rajesh Roshan Biswal. "Advanced IDS: A Comparative Study of Datasets and Machine Learning Algorithms for Network Flow-Based Intrusion Detection Systems: JC Mondragon et al." Applied Intelligence 55, no. 7 (2025): 608.

[7] Kumar, Vikash, Ayan Kumar Das, and Ditipriya Sinha. "Statistical Analysis of the UNSW-NB15 Dataset for Intrusion Detection." In Computational Intelligence in Pattern Recognition: Proceedings of CIPR 2019, Singapore: Springer Singapore, 2019, 279-294.

[8] Stiawan, Deris, Mohd Yazid Bin Idris, Alwi M. Bamhdi, and Rahmat Budiarto. "CICIDS-2017 Dataset Feature Analysis with Information Gain for Anomaly Detection." IEEE access 8 (2020): 132911-132921.

[9] Mejia-Cabrera, Heber I., Daniel Paico-Chileno, Jhon H. Valdera-Contreras, Victor A. Tuesta-Monteza, and Manuel G. Forero. "Automatic Detection of Injection Attacks by Machine Learning in NoSQL Databases." In Mexican Conference on Pattern Recognition, Cham: Springer International Publishing, 2021, 23-32.

[10] Moustafa, Nour. "ToN_IoT Cybersecurity Dataset." UNSW Canberra Cyber, 2019. https://research.unsw.edu.au/projects/toniot-datasets.

[11] Manzali, Youness, and Mohamed Elfar. "Random Forest Pruning Techniques: A Recent Review." In Operations research forum, vol. 4, no. 2, Cham: Springer International Publishing, 2023, 43.

[12] Bentéjac, C., Csörgő, A. & Martínez-Muñoz, G. A Comparative Analysis of Gradient Boosting Algorithms. Artif Intell Rev 54, 1937–1967 (2021).

[13] Zhao, Qihao, Fuwei Wang, Weimin Wang, Tianxin Zhang, Haodong Wu, and Weijun Ning. "Research on Intrusion Detection Model Based on Improved MLP Algorithm." Scientific reports 15, no. 1 (2025): 5159.

[14] Hozouri, Ali, Abbas Mirzaei, and Mehdi Effatparvar. "A Comprehensive Survey on Intrusion Detection Systems with Advances in Machine Learning, Deep Learning and Emerging Cybersecurity Challenges." Discover Artificial Intelligence 5, no. 1 (2025): 314.

[15] Rehman, Hafiz Muhammad Raza Ur, Saira Liaquat, Muhammad Junaid Gul, Muhammad Zeeshan Jhandir, Daniel Gavilanes, Manuel Masias Vergara, and Imran Ashraf. "A Systematic Literature Study of Machine Learning Techniques Based Intrusion Detection: Datasets, Models, Challenges, and Future Directions." Journal of Big Data 12, no. 1 (2025): 264.

[16] Ahmed, Usama, Mohammad Nazir, Amna Sarwar, Tariq Ali, El-Hadi M. Aggoune, Tariq Shahzad, and Muhammad Adnan Khan. "Signature-Based Intrusion Detection Using Machine Learning and Deep Learning Approaches Empowered with Fuzzy Clustering." Scientific Reports 15, no. 1 (2025): 1726.

[17] Mohale, Vincent Zibi, and Ibidun Christiana Obagbuwa. "Evaluating Machine Learning-Based Intrusion Detection Systems with Explainable AI: Enhancing Transparency and Interpretability." Frontiers in Computer Science 7 (2025): 1520741.

[18] Sharma, Anshika, Himanshi Babbar, and Avinash Sharma. "Ton-IoT: Detection of Attacks on Internet of Things in Vehicular Networks." In 2022 6th International Conference on Electronics, Communication and Aerospace Technology, IEEE, 2022, 539-545.

[19] Elsayed, Rania, Reem Hamada, Mohammad Hammoudeh, Mahmoud Abdalla, and Shaimaa Ahmed Elsaid. "A Hierarchical Deep Learning-Based Intrusion Detection Architecture for Clustered Internet of Things." Journal of Sensor and Actuator Networks 12, no. 1 (2022): 3.

[20] Sicari, Sabrina, Alessandra Rizzardi, and Alberto Coen-Porisini. "Security&privacy Issues and Challenges in NoSQL Databases." Computer Networks 206 (2022): 108828.

[21] Rameshwar, D., and S. Nagasundari. "The MongoDB Injection Dataset: A Comprehensive Collection of MongoDB-NoSQL Injection Attempts and Vulnerabilities." Data in Brief 54 (2024): 110289.

[22] Sadhwani, Sapna, Baranidharan Manibalan, Raja Muthalagu, and Pranav Pawar. "A Lightweight Model for DDoS Attack Detection Using Machine Learning Techniques." Applied Sciences 13, no. 17 (2023): 9937.