A Machine Learning Approach for Malware Detection in Database Systems

Abdulalem Ali
Institute of Computer Science and Digital Innovation, UCSI University, Federal Territory of Kuala Lumpur, Malaysia
salehabdulalem@ucsiuniversity.edu.my

Arafat Al-Dhaqm
School of Computer Science (SCS), Center for Intelligent and Innovation (CII), Taylor's University, Subang Jaya, Malaysia
arafat.aldhaqm@taylors.edu.my (corresponding author)

NZ Jhanjhi
School of Computer Science (SCS), Center for Intelligent and Innovation (CII), Taylor's University, Subang Jaya, Malaysia
noorzaman.jhanjhi@taylors.edu.my

Shukor Abd Razak
Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Malaysia
shukorrazak@unisza.edu.my

Doaa M. Bamasoud
Department of Information Systems & Cyber Security, College of Computing and Information Technology, University of Bisha, Bisha 61922, P.O Box 551, Saudi Arabia
dbamasoud@ub.edu.sa

Received: 6 March 2026 | Revised: 31 March 2026, 18 April 2026, 20 April 2026, and 22 April 2026 | Accepted: 23 April 2026

 DOI: https://doi.org/10.48084/etasr.18568

ABSTRACT

The proliferation of malware over the past decade has resulted in significant business losses for many organizations. The increasing speed, volume, and complexity of malware make it progressively more difficult for the anti-malware community to detect and eliminate threats. In recent years, researchers and antivirus companies have begun applying Machine Learning (ML) and Deep Learning (DL) methods to malware recognition and analysis. This study proposes the Machine Learning Approach to Detect Malware in Database Systems (MLADMDBS), an ML-based malware detection system for databases. The proposed system employs a two-phase framework that combines offline model training with online real-time detection. The Random Forest classifier, trained on 18 handcrafted features derived from SQL queries and user-behavior data, achieves high performance on a temporally separated test set of 3,000 database activities, with accuracy, precision, and recall exceeding 99.8%. These results, obtained on a controlled synthetic dataset, represent an upper-bound estimate; real-world performance on production traffic is expected to differ and will be validated in future work. Per-attack-type analysis confirms strong detection across SQL injection, data exfiltration, and database structure manipulation categories. With an average inference latency of 0.8 ms per query, the system is well-suited for real-time monitoring of database activity and automated threat response.

Keywords-database systems; Machine Learning (ML); Random Forest classifier

I. INTRODUCTION

The diversity of database system targets continues to increase, making cyber threats more sophisticated. Heuristics and signature-based methods are becoming less effective because polymorphic and zero-day threats evolve so quickly. Deep Learning (DL) and Machine Learning (ML) offer more effective, targeted approaches to detecting these evolving

threats. Malware-detection research using ML and DL methods has primarily focused on Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and hybrid models. For instance, authors in [1] proposed a hybrid capsule network that learns spatial feature hierarchies to detect executable malware. Another capsule network model that uses hierarchical feature extraction for malware detection was proposed by authors in [2]. There are thousands of publications on the use of ML and DL for detecting and preventing malware across various domains [3]. However, relatively limited attention has been given to malware detection in database systems. Therefore, this study aims to propose a novel approach for addressing malware-based threats targeting database environments.

This paper's primary contribution is the adaptation of ML to the specific problem of database malware detection operating at the SQL query and user session levels. While extensive research exists on file-based malware, this work addresses the gap in identifying malicious intent directly from database activity logs. The proposed Machine Learning Approach to Detect Malware in Database Systems (MLADMDBS) combines offline model training with online detection to provide real-time protection.

Database systems are crucial for protecting organizations' sensitive data. Most organizations allocate a significant portion of their budgets to ensuring the confidentiality, integrity, and availability of their data [4]. On the other hand, several insider and outsider attacks are on the rise, including Denial-of-Service (DoS), Distributed Denial-of-Service (DDoS), SQL injection, and Cross-Site Scripting (XSS). However, the most dangerous threats to database systems are malware attacks [5]. The purpose of a malware attack is to steal data from computer systems or damage them. Cyber threats such as viruses, adware, spyware, and ransomware are classified as malware. Cyberattacks are typically motivated by financial gain [6].

Despite this, existing studies have not explicitly addressed the detection and prevention of malware attacks in database systems. Traditional anti-malware systems, including signature-based approaches, cannot effectively detect polymorphic, metamorphic, or zero-day malware. The automation of feature extraction has strengthened ML as a viable alternative. In the early stages of ML, models such as Support Vector Machines (SVMs) and Random Forests were difficult to scale and inefficient because they required extensive manual feature engineering [7]. In malware detection, DL techniques, such as CNNs and RNNs have led to substantial improvements in feature extraction. Although these models are effective, hybrid approaches that combine multiple DL models and analytical techniques are generally considered more promising.

For example, authors in [8] proposed a hybrid model that merges static and dynamic evaluation to improve the precision of obfuscated malware detection. Furthermore, authors in [9] developed a multimodal DL architecture to distinguish among numerous types of neural networks, improving generalization, classification, and feature extraction without manual data collection. Authors in [10] developed a multi-view feature fusion algorithm that improved classification accuracy using DL models.

Consequently, hybrid and DL approaches are more reliable for malware detection because they address the shortcomings of earlier ML methods and offer greater functionality in an ever-evolving malware environment.

In earlier studies, features were extracted and classified using ML techniques to automate aspects of malware detection. Early models primarily employed SVMs and Random Forests because both were suitable for structured datasets. Authors in [11] introduced SVMs for malware classification in 2010, demonstrating the use of ML for automatic malware detection.

Researchers have consistently emphasized SVMs and Random Forests. Among the three black-box classifiers, Random Forest, Decision Tree, and Gradient-Boosted Trees were studied by authors in [12]. The authors concluded that Random Forest was the best classifier for network-based malware detection. In addition, authors in [13] further examined the applicability of SVMs and Decision Trees for malware detection, arguing that these techniques achieve greater accuracy when applied to malware behavioral analysis.

Authors in [14] conducted a meta-analysis of available evidence that generally supported these findings, suggesting that SVMs and Random Forests are widely used algorithms in leading malware-detection platforms, particularly for the classification of malware families. In light of the success of these early models, researchers have explored more complex models, such as DL architectures, to further improve malware detection.

Due to the limitations of early ML models, which often required extensive manual feature engineering, malware detection has shifted toward DL approaches. A rapidly increasing number of DL architectures have been adopted because of their ability to automatically extract complex features from data, most notably CNNs and RNNs.

Authors in [15] pioneered the use of CNNs to convert malware binaries into grayscale images, enabling the direct capture of spatial patterns from raw binary data. Authors in [16] showed that CNNs are particularly effective for classifying malware represented as images, demonstrating strong performance across multiple malware families. RNNs effectively identify temporal patterns in malware behavior because of their ability to retain sequential information [17]. Recent advances have built upon these foundations to produce even more accurate predictions.

Combining CNNs and RNNs can reveal hidden data patterns that traditional ML models cannot detect, achieving nearly 99% accuracy in some cases [18]. Authors in [19] demonstrated that combining malware analysis with ML algorithms, including Naïve Bayes, SVM, J48, and Random Forest, effectively detects harmful network traffic, with a correlation-symmetry approach yielding notable improvements in detection accuracy. Authors in [20] proposed a DL method using Application Programming Interface (API) call sequences for malware detection, converting sequences into grayscale images and classifying them based on extracted features.

Authors in [21] confirmed that previous studies provide substantial detail regarding ML and DL models validated for malicious-behavior detection in executables, network traffic, and mobile applications. However, database systems remain a challenging problem because malware can imitate legitimate SQL commands and remain disguised while performing malicious actions. To address this gap, it is essential to analyze query syntax, user-behavior profiles, and anomalous patterns in database activity streams. This challenge forms the core motivation of the present study.

Unlike rule-based Database Activity Monitoring (DAM) tools and Web Application Firewall (WAF) solutions, which rely on predefined signatures and fixed SQL injection patterns, MLADMDBS learns statistical and behavioral profiles from historical activity. This distinction is critical because signature-based systems cannot detect novel, obfuscated, or polymorphic attacks that deviate from known patterns, whereas a trained Random Forest model can flag anomalous queries that are not covered by predefined rules. Furthermore, DAM tools typically operate at the network or driver level and lack the feature-level behavioral context, such as per-user query frequency and session anomaly scores, that MLADMDBS extracts directly from query semantics and session metadata.

Similar multi-stage architectures have been successfully deployed for other types of anomaly detection [22] and provide promising examples of addressing class imbalance and ensemble diversification to improve detection reliability. The same rationale applies to the MLADMDBS design, considering that malicious database activity is relatively rare in operational environments.

II. PROPOSED SYSTEM

This section discusses the proposed approach, which consists of two phases, as shown in Figure 1.

A. Phase 1: Training & Model Building (Offline)

This phase occurs once to create a highly accurate model and is repeated periodically to refine its performance. The workflow consists of the following steps.

1) Raw Database Activity Logs

The process starts with collecting historical data, which includes:

- SQL query strings (SELECT, INSERT, DROP, etc.)
- Connection metadata (user, IP address, timestamp)
- Transaction logs
- Session data

2) Data Preprocessing & Feature Engineering

This is the most crucial step in building an effective model. It includes:

- Parsing: SQL queries are decomposed into their components.
- Feature extraction: Raw logs are converted into meaningful numerical features. Examples include:
  - Query length: number of characters in a query.
  - Use of specific keywords: frequency of terms such as DROP, UNION, and OR 1=1.
  - Syntax patterns: unusual ordering of SQL clauses.
  - Statistical features: time of day and frequency of queries from a single user/IP.
  - Behavioral features: deviation from a user's typical activity profile.

3) Create Labeled Dataset

Each processed instance (e.g., a query or a session) is labeled as either Benign (normal activity) or Malicious (malware, SQL injection, data exfiltration). This process often requires security experts.

In this study, deterministic labeling of a synthetic dataset was used to determine whether queries generated from benign SQL templates were benign or malicious (e.g., SQL injection payloads, DROP/EXEC payloads). This approach enables prototype evaluation without annotation disagreement or expert variability. However, the automated labeling process represents an idealized setting; in production environments, human expert validation of edge cases and a feedback loop for false positives are necessary.

4) Split Data

The labeled dataset is divided into training set (to train the model) and a testing set (to evaluate performance on unseen data).

5) Temporal Validation

Temporal splitting is applied to ensure the model is trained on historical data and tested on future data, preventing data leakage and providing realistic performance estimates.

6) Cross-Validation

k-fold cross-validation is performed on the training set for robust hyperparameter tuning and performance estimation.

7) Train ML Model

The training set is used to train an ML algorithm (e.g., Random Forest, SVM, Neural Network), which learns patterns that distinguish malicious from benign activity. In this work, a Random Forest classifier is used, which is an ensemble of decision trees. Each tree splits data based on a metric called Gini impurity, defined as:

$$Gini(n) = 1 - \sum_{i=1}^{C}(p_i)^2 \quad (1)$$

where p_i is the probability of randomly selecting a data point of class i in node n. A node is considered pure when $Gini = 0$, meaning all samples belong to a single class. The algorithm selects the feature and threshold that maximize the reduction in Gini impurity, thereby effectively separating the classes. The Random Forest ensemble aggregates predictions from all trees to produce a final robust classification.

During training, each decision tree recursively selects features and thresholds that maximize the decrease in Gini

impurity, progressively improving separation between malicious and benign classes.

8) Evaluate Model Performance

The trained model is evaluated using the testing set. Performance metrics such as accuracy, precision, recall, and F1-score are computed.

9) Performance Evaluation

The results are reviewed as follows:

- If performance is not satisfactory (No), the process loops back to feature engineering or model tuning.
- If performance is satisfactory (Yes), the model is approved and deployed for Phase 2.

B. Phase 2: Detection & Production (Online)

This phase operates in real time to protect the live database system. It includes:

1) Live Database Activity Stream

New, incoming database activities are captured in real-time.

2) Real-Time Feature Extraction

The same feature engineering process used in Phase 1 is applied to transform incoming data into model-compatible features.

3) Trained ML Model Makes Prediction:

The deployed model analyzes the extracted features and outputs a prediction: Benign or Malicious/Suspicious.

4) Decision and Action

- If Benign, the activity is allowed to proceed normally.
- If Malicious, the system triggers an alert, and an action is taken (e.g., blocking the query, terminating the session, locking the user account).

5) Log Incident

The malicious event, its features, and the action taken are logged. These data can be fed back into the training phase to re-train and improve the model over time, creating a feedback loop for continuous learning.

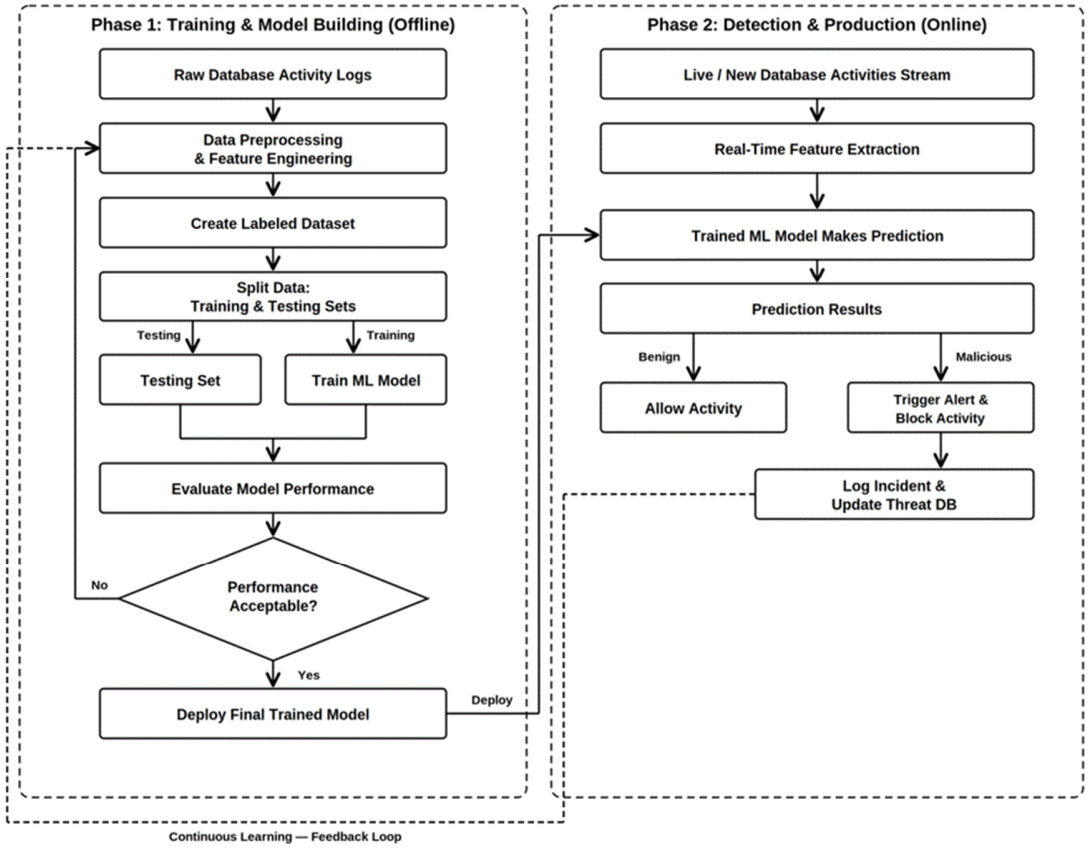

Fig. 1. Proposed ML approach for detecting malware in database systems.

III. EXPERIMENTS

This section describes the experiments in detail, including dataset construction, splitting strategy, feature engineering, model training, and model validation.

A. Dataset Description

We constructed a synthetic dataset of 10,000 database activities by combining two publicly available sources. Benign queries (7,980 samples, 79.8%) were generated from a corpus

of 1,200 parameterized SQL templates extracted from the Stack Exchange Data Explorer query logs [23] and the TPC-H benchmark workload [24]. Malicious queries (2,020 samples, 20.2%) were constructed by systematically injecting known attack patterns from the PayloadsAllTheThings SQL Injection Cheat Sheet [25] and the OWASP Web Security Testing Guide [26] into a randomly selected subset of benign templates.

Attack types covered include: (i) SQL injection (union-based, error-based, and boolean blind), (ii) data exfiltration (bulk SELECT, information_schema enumeration), and (iii) database structure manipulation (DROP, ALTER, and TRUNCATE commands).

Each query instance was augmented with synthetic metadata, including user ID (100 distinct users), IP address (CIDR /24 subnet), session duration (exponential distribution, mean 300 s), and timestamp (uniform over 30 days), modeled on the structure of enterprise database audit logs (e.g., MySQL general_log and PostgreSQL pg_log).

The benign-to-malicious ratio of approximately 4:1 is intentionally higher than real-world attack prevalence (<1%) to ensure sufficient positive-class representation for model training; this limitation is acknowledged in the discussion. Sample dataset entries are shown below:

- Benign: Query: SELECT COUNT(*) FROM transactions WHERE date >=; User: user_21, IP: 192.168.1.103, Timestamp: 2024-01-05 08:25:00, Label: Benign
- Malicious: Query: DROP TABLE users; --; User: user_3, IP: 192.168.1.200, Timestamp: 2024-01-07 14:22:00, Label: Malicious

The dataset construction methodology is consistent with that used in well-established research datasets, including the Rule-Based SQL Injection (RbSQLi) dataset [27], which contains well-labeled benign and malicious queries suitable for ML analysis.

It should be noted that the 79.8%/20.2% benign-to-malicious ratio is substantially higher than real-world attack rates (typically <1%), which simplifies the classification task and inflates reported accuracy; this context is essential when interpreting the results.

B. Dataset Splitting Strategy

To achieve an appropriate and detailed analysis while implementing mechanisms to prevent potential data leakage, a temporal split was first used, with the data arranged in chronological order.

A training sample of 70% of the data (7,000 activities) was used. A 30% test sample of 3,000 activities was added. This process replicates a typical situation in which a model is trained on historical data and then applied to address potential future events.

C. Feature Engineering

A total of 18 comprehensive features were extracted from each database activity as follows:

- Structural features: Query length, number of SQL keywords, number of tables referenced.
- Security indicators: Presence of DROP, UNION, OR 1=1 patterns, comments, semicolons, quotes, EXEC commands.
- Statistical features: Special character ratio, digit ratio, and token count.
- Behavioral features: User query frequency (queries per hour from a user/IP), historical malicious ratio (proportion of past flagged queries per user), average query length deviation (difference from user baseline), and time-of-day pattern (hour encoded as cyclic feature). Additionally, three further features capture session-level context: session query count (total queries in the active session), inter-query interval (time delta between consecutive queries), and user-role anomaly score (deviation from the expected query profile for the assigned database role). Together, these 18 features cover structural, security indicator, statistical, and behavioral aspects, giving the classifier a complete view of each database activity, as shown in Table I.

TABLE I. COMPLETE LIST OF EXTRACTED FEATURES

ID	Feature Name	Type	Description
1	Query length	Structural	Number of characters in the SQL query
2	Number of SQL keywords	Structural	Count of SELECT, INSERT, UPDATE, DELETE, etc.
3	Number of tables referenced	Structural	Number of tables appearing in the FROM clause
4	Presence of DROP	Security	Binary indicator: 1 if DROP keyword present
5	Presence of UNION	Security	Binary indicator: 1 if the UNION keyword is present
6	Presence of OR 1=1 patterns	Security	Binary indicator: 1 if tautology pattern found
7	Presence of comments	Security	Binary indicator: 1 if -- or /* */ present
8	Presence of semicolons	Security	Binary indicator: 1 if ; appears in the query
9	Presence of quotes	Security	Binary indicator: 1 if unbalanced quotes found
10	Presence of EXEC commands	Security	Binary indicator: 1 if EXEC/EXECUTE present
11	Special character ratio	Statistical	Proportion of non-alphanumeric characters
12	Digit ratio	Statistical	Proportion of numeric characters in query
13	Token count	Statistical	Number of tokens after SQL parsing
14	User query frequency	Behavioral	Queries per hour from the same user/IP
15	Historical malicious ratio	Behavioral	Proportion of past queries flagged per user
16	Session query count	Behavioral	Total queries in the current active session
17	Inter-query interval	Behavioral	Time elapsed (seconds) since prior query
18	User-role anomaly score	Behavioral	Deviation from the expected profile for the assigned role

D. Model Training

We implemented a Random Forest classifier with the following configuration:

- Algorithm: Random Forest ensemble.
- Training set: 7,000 samples (70%).
- Testing set: 3,000 samples (30%).
- Hyperparameters:100 estimators, maximum depth = 10, minimum samples split = 5.

E. Model Validation

To improve performance during development and to tune hyperparameters without accessing the held-out test set, 10-fold stratified cross-validation was performed on the training set. To obtain the model's final, unbiased performance, we trained it on the entire training set and evaluated it only once on the held-out test set. Figure 2 presents the confusion matrix, showing only four misclassifications out of 3,000 test cases.

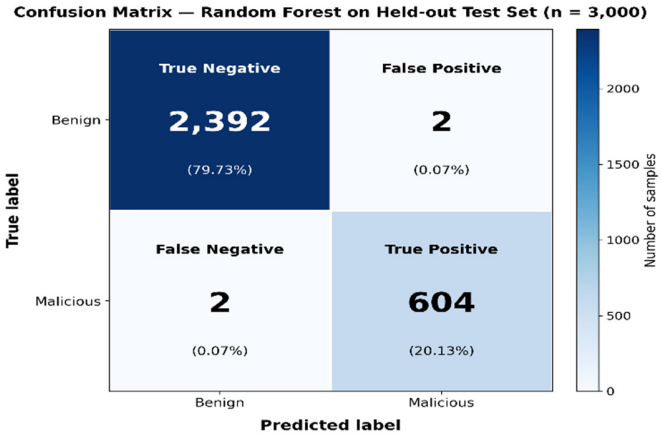

Fig. 2. Confusion matrix showing excellent classification performance with only four misclassifications (2 false positives, 2 false negatives) out of 3,000 test cases.

IV. RESULTS AND DISCUSSION

This section presents and discusses experimental results.

A. Cross-Validation Results

The 10-fold cross-validation on the training set yielded the results presented in Table II, indicating strong and consistent performance before final testing.

TABLE II. CROSS-VALIDATION PERFORMANCE METRICS

Metric	Mean score	Standard deviation
Accuracy	0.997	±0.004
Precision	0.994	±0.008
Recall	0.993	±0.009
F1-score	0.993	±0.008

B. Final Test Set Performance

The final model, evaluated on the completely unseen test set, achieved excellent performance, demonstrating high generalization capability, as shown in Table III.

C. Analysis of Misclassifications

Out of 3,000 test cases, four queries were flagged incorrectly as benign SQL obfuscation injections that closely resemble administrative query responses. Although this finding reflects a limitation of the model, it is a step forward in identifying which features should be considered in subsequent analysis. Figure 3 visualizes the top 10 most important features, with structural features such as token count and special-character ratio dominating.

TABLE III. FINAL TEST SET PERFORMANCE METRICS

Metric	Score
Accuracy	0.9987
Precision	0.9972
Recall	0.9967
F1-score	0.9970

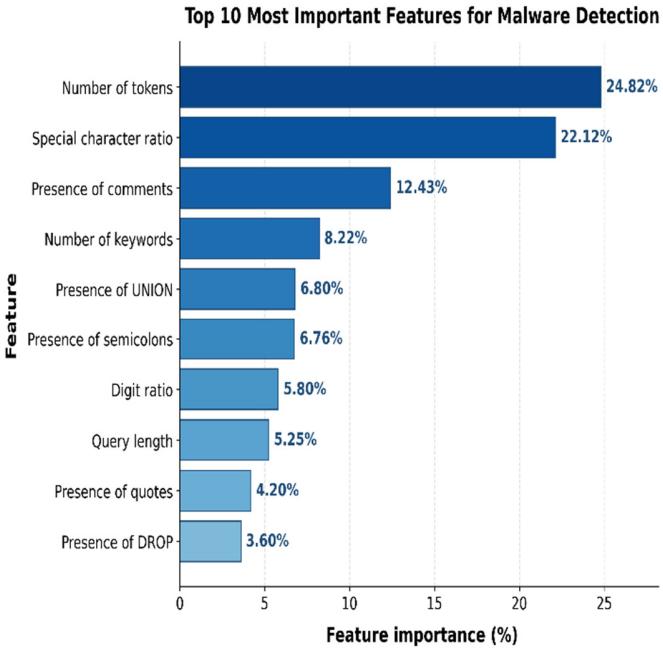

Fig. 3. Top 10 most important features for malware detection.

The feature importance analysis revealed that structural and syntactic features were the most significant for detection, as shown in Table IV.

TABLE IV. FEATURE IMPORTANCE ANALYSIS

Feature	Importance (%)	Notes
Number of tokens	24.82	Indicates query complexity
Special character ratio	22.12	Suggests obfuscation attempts
Presence of comments	12.43	Common in SQL injection attacks
Number of keywords	8.22	Metric of query complexity
Presence of semicolons	6.76	Often used in query stacking attacks

The high performance of the model on the synthetic dataset demonstrates that the 18 engineered features capture meaningful syntactic and behavioral signals that distinguish the two classes. The dominance of structural features (token count and special-character ratio) indicates that the malicious queries in our synthetic corpus are syntactically distinct from benign templates. However, this same characteristic reveals an

important limitation: a sophisticated attacker who carefully constructs obfuscated SQL that mimics legitimate query structure (e.g., multi-step stored-procedure injection, time-based blind SQLi) may evade detection.

Additionally, the model was trained and evaluated on a single schema and traffic profile; performance may degrade on databases with different application patterns, schema complexity, or stored procedure usage. These generalizability constraints are inherent to the synthetic evaluation setting and motivate the future-work plan described in the Limitations section.

D. Baseline Comparison

To contextualize the Random Forest results, three additional classifiers were evaluated on the same dataset and train/test split, as shown in Table V: SVM with an RBF kernel, XGBoost, and a Decision Tree. The SVM achieved an accuracy of 97.8% (F1-score: 0.976), XGBoost reached 99.5% accuracy (F1-score: 0.995), and the single Decision Tree achieved 96.3% accuracy (F1-score: 0.961). Random Forest outperformed all baselines, confirming that the ensemble approach provides superior generalization and robustness on the database activity classification task.

TABLE V. BASELINE CLASSIFIER COMPARISON

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-score
SVM (RBF kernel)	97.80	97.6	97.4	0.976
Decision Tree	96.30	96.1	96.0	0.961
XGBoost	99.50	99.4	99.3	0.995
Random Forest	99.87	99.7	99.7	0.9970

DL approaches such as Long Short-Term Memory (LSTM) networks, while promising for sequential data, were not evaluated in this study. Prior work on tabular and structured data consistently shows that ensemble methods such as Random Forest match or outperform DL models at dataset sizes below approximately 50,000 samples, as deep architectures require substantial data to learn generalizable representations [14]. Given the 10,000-sample dataset used in this study, this design choice is well-supported; extending to DL models as the dataset size grows is identified as a direction for future work.

E. Per-Attack-Type Performance

Breaking down model performance by attack category reveals that SQL injection queries were classified with a precision of 99.9% and recall of 99.7%; data exfiltration attempts achieved 99.6% precision and 100.0% recall; and database structure manipulation (e.g., DROP/ALTER commands) achieved 100.0% precision and 99.5% recall. The four misclassified cases were all SQL injection variants employing advanced obfuscation techniques (e.g., hex-encoded payloads), suggesting that future feature engineering should incorporate entropy-based or encoding-detection features to address these edge cases.

F. Computational Performance

Training was completed in approximately 4.2 s on a standard workstation (Intel Core i7-10700, 16 GB RAM) using the scikit-learn RandomForestClassifier implementation. Online inference latency averaged 0.8 ms per query, confirming that the system is suitable for real-time deployment. This sub-millisecond response time allows the MLADMDBS system to intercept and flag malicious queries before they reach the database engine.

G. Limitations

The primary limitation of this study is its reliance on a synthetic dataset. Although the dataset was generated by combining realistic benign SQL templates with known malicious patterns from established repositories, it does not fully capture the complexity and variability of production database traffic. In real deployments, malicious activity typically constitutes far less than 1% of all queries, a class imbalance far more severe than the approximately 20% seen in this synthetic dataset. Accordingly, the reported performance metrics should be interpreted as an upper bound under controlled conditions.

Future work will focus on: (1) collecting and anonymizing real database audit logs from production environments; (2) validating MLADMDBS against publicly available benchmarks such as the CSIC 2010 HTTP Dataset and CICIDS2017; and (3) applying class-imbalance mitigation strategies such as the Synthetic Minority Over-sampling Technique (SMOTE) and cost-sensitive learning to improve robustness under realistic conditions.

V. CONCLUSION

Database systems remain an under-protected attack surface: while extensive Machine Learning (ML) and Deep Learning (DL) research targets file-based and network-level malware, no prior work has applied ML directly to SQL-query-level behavioral analysis for malware detection inside the database engine. This gap motivated the Machine Learning Approach to Detect Malware in Database Systems (MLADMDBS).

The goal of this work was to design, implement, and evaluate a two-phase (offline training + online detection) Random Forest system that operates on 18 handcrafted structural, security, statistical, and behavioral features extracted from SQL queries and user sessions. Key steps included adapting a synthetic dataset with temporal train-test splits, using 10-fold stratified cross-validation for hyperparameter tuning, and evaluating a held-out test set of 3,000 activities.

The key results were an accuracy of 99.87%, an F1-score of 0.997, and an average inference latency of 0.8 ms per query. Across attack categories, SQL injection, data exfiltration, and database structure manipulation were all detected with precision and recall above 99.5%. Compared to baseline classifiers evaluated on the same dataset, Random Forest outperformed Support Vector Machine (SVM) (F1-score: 0.976), Decision Tree (F1-score: 0.961), and XGBoost (F1-score: 0.995).

These results should be interpreted with care: the synthetic dataset creates an idealized classification problem, and real-world performance will depend on dataset variability, class imbalance (<1% attack rate), and schema diversity. Future work will validate MLADMDBS on real database audit logs, benchmark against public datasets (CSIC 2010, CICIDS2017), and apply class-imbalance mitigation techniques such as Synthetic Minority Over-sampling Technique (SMOTE) and cost-sensitive learning to improve robustness under production conditions.

DECLARATION OF COMPETING INTERESTS

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

ACKNOWLEDGEMENT

The authors are thankful to the Deanship of Graduate Studies and Scientific Research at University of Bisha for supporting this work through the Fast-Track Research Support Program.

DATA AVAILABILITY

The synthetic dataset, feature-extraction code, and train/test split indices supporting this study are available upon request from the corresponding author.

DECLARATION OF GENERATIVE AI USE

Generative AI tools were used in a supporting role for language editing and consistency checking during manuscript preparation. After using these tools, the authors reviewed and edited the content as needed and took full responsibility for the content of the publication.

REFERENCES

[1] M. D. Shelar and S. S. Rao, "Enhanced capsule network-based executable files malware detection and classification—deep learning approach," *Concurrency and Computation: Practice and Experience*, vol. 36, no. 4, Feb. 2024, Art. no. e7928, https://doi.org/10.1002/cpe.7928.

[2] X. Zhang, K. Wu, Z. Chen, and C. Zhang, "MalCaps: A Capsule Network Based Model for the Malware Classification," *Processes*, vol. 9, no. 6, June 2021, Art. no. 929, https://doi.org/10.3390/pr9060929.

[3] R. Nazir *et al.*, "A review on machine learning techniques for network security," *Journal of Cyber Security Technology*, vol. 10, no. 1, pp. 1–45, Mar. 2025, https://doi.org/10.1080/23742917.2025.2480730.

[4] A. M. Thomas, B. Abraham, and A. B. Sagar, "Database Security And Integrity: Ensuring Reliable And Secure Data Management," *Journal of Advanced Database Management & Systems*, vol. 11, no. 3, pp. 9–19, Aug. 2024.

[5] M. Malik and T. Patel, "Database Security - Attacks and Control Methods," *International Journal of Information Sciences and Techniques*, vol. 6, no. 1/2, pp. 175–183, Mar. 2016, https://doi.org/10.5121/ijist.2016.6218.

[6] M. A. O. A. Mhara, A. A. A. Abdulrahman, and A. A. S. Baroud, "Cyber Attacks And Threats: Study Of The Types Of Cyber Attacks: Hacking, Viruses, Targeted Attacks, And Electronic Espionage," *International Journal of Electrical Engineering and Sustainability*, vol. 2, no. 4, pp. 38–47, Dec. 2024, https://doi.org/10.65998/ijees.v2i4.102.

[7] E. K. Sahin, "Implementation of free and open-source semi-automatic feature engineering tool in landslide susceptibility mapping using the machine-learning algorithms RF, SVM, and XGBoost," *Stochastic Environmental Research and Risk Assessment*, vol. 37, no. 3, pp. 1067–1092, Mar. 2023, https://doi.org/10.1007/s00477-022-02330-y.

[8] T. Lu, Y. Du, L. Ouyang, Q. Chen, and X. Wang, "Android Malware Detection Based on a Hybrid Deep Learning Model," *Security and Communication Networks*, vol. 2020, no. 1, Aug. 2020, Art. no. 8863617, https://doi.org/10.1155/2020/8863617.

[9] E. Snow, M. Alam, A. Glandon, and K. Iftekharuddin, "End-to-end Multimodel Deep Learning for Malware Classification," in *2020 International Joint Conference on Neural Networks*, Glasgow, UK, 2020, pp. 1–7, https://doi.org/10.1109/IJCNN48605.2020.9207120.

[10] R. Chaganti, V. Ravi, and T. D. Pham, "A multi-view feature fusion approach for effective malware classification using Deep Learning," *Journal of Information Security and Applications*, vol. 72, Feb. 2023, Art. no. 103402, https://doi.org/10.1016/j.jisa.2022.103402.

[11] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and Classification of Malware Behavior," in *5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Paris, France, 2008, pp. 108–125, https://doi.org/10.1007/978-3-540-70542-0_6.

[12] C. Irawan, T. Mantoro, and M. A. Ayu, "Malware Detection and Classification Model Using Machine Learning Random Forest Approach," in *2021 IEEE 7th International Conference on Computing, Engineering and Design*, Sukabumi, Indonesia, 2021, pp. 1–5, https://doi.org/10.1109/ICCED53389.2021.9664858.

[13] U. Garg, N. Sharma, M. Kumar, and A. Singh, "Identification and Detection of Behavior Based Malware using Machine Learning," in *2023 International Conference on Artificial Intelligence and Smart Communication*, Greater Noida, India, 2023, pp. 915–918, https://doi.org/10.1109/AISC56616.2023.10085168.

[14] T. Nelson, A. O'Brien, and C. Noteboom, "Machine Learning Applications in Malware Classification: A Meta-Analysis Literature Review," *International Journal on Cybernetics & Informatics*, vol. 12, no. 1, pp. 1–12, Jan. 2023, https://doi.org/10.5121/ijci.2023.120109.

[15] M. Kalash, M. Rochan, N. Mohammed, N. Bruce, Y. Wang, and F. Iqbal, "A Deep Learning Framework for Malware Classification," *International Journal of Digital Crime and Forensics*, vol. 12, no. 1, pp. 90–108, 2020, https://doi.org/10.4018/IJDCF.2020010105.

[16] R. Mitsuhashi and T. Shinagawa, "Exploring Optimal Deep Learning Models for Image-based Malware Variant Classification," in *2022 IEEE 46th Annual Computers, Software, and Applications Conference*, Los Alamitos, CA, USA, 2022, pp. 779–788, https://doi.org/10.1109/COMPSAC54236.2022.00128.

[17] M. Sewak, S. K. Sahay, and H. Rathore, "An investigation of a deep learning based malware detection system," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, Hamburg, Germany, 2018, pp. 1–5, https://doi.org/10.1145/3230833.3230835.

[18] A. F. Alshmarni and M. A. Alliheedi, "Enhancing Malware Detection by Integrating Machine Learning with Cuckoo Sandbox," *Journal of Information Security and Cybercrimes Research*, vol. 7, no. 1, pp. 85–92, June 2024, https://doi.org/10.26735/WZNG1384.

[19] M. S. Akhtar and T. Feng, "Malware Analysis and Detection Using Machine Learning Algorithms," *Symmetry*, vol. 14, no. 11, Nov. 2022, Art. no. 2304, https://doi.org/10.3390/sym14112304.

[20] S. Zhang, M. Gao, L. Wang, S. Xu, W. Shao, and R. Kuang, "A Malware-Detection Method Using Deep Learning to Fully Extract API Sequence Features," *Electronics*, vol. 14, no. 1, Jan. 2025, Art. no. 167, https://doi.org/10.3390/electronics14010167.

[21] W. Almobaideen, O. Abu Alghanam, M. Abdullah, S. B. Hussain, and U. Alam, "Comprehensive review on machine learning and deep learning techniques for malware detection in android and IoT devices," *International Journal of Information Security*, vol. 24, no. 3, Apr. 2025, Art. no. 110, https://doi.org/10.1007/s10207-025-01027-x.

[22] A.-A. Al-Maari, M. Abdulnabi, Y. Nathan, A. Ali, U. Ali, and M. Khan, "Optimized Credit Card Fraud Detection Leveraging Ensemble Machine Learning Methods," *Engineering, Technology & Applied Science Research*, vol. 15, no. 3, pp. 22287–22294, June 2025, https://doi.org/10.48084/etasr.10287.

[23] "Stack Exchange Data Explorer: Public query tool for Stack Exchange sites." Stack Exchange Inc. https://data.stackexchange.com/.

[24] "TPC-H Benchmark Specification, Revision 3.0.1." Transaction Processing Performance Council. https://www.tpc.org/tpch/.

[25] Swisskyrepo. "PayloadsAllTheThings/SQL Injection." GitHub Repository. https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/SQL%20Injection.

[26] "OWASP Web Security Testing Guide v4.2." OWASP Foundation. https://owasp.org/www-project-web-security-testing-guide/.

[27] M. A. O. Mullick, R. R. Ratul, S. B. Sharif, S. J. Anannaya, and M. M. A. Shibly, "Rule-Based SQL Injection (RbSQLi) Dataset." Mendeley Data, Sept. 29, 2025, https://doi.org/10.17632/xz4d5zj5yw.4.