AI-Driven Threat Detection: Exploring Machine Learning for Early Malware Identification

Adeyemi Isaiah Fagbade

Market Insights Team,
Saskatchewan Inidian Gaming Authority,
Saskatoon, SK, Canada.

adeyemi.fagbade@ndus.edu

***Abstract*—The rapid evolution of malware and the emergence of sophisticated threats, such as nascent polymorphic malware, have made traditional signature-based detection methods increasingly ineffective. Polymorphic malware constantly modifies its signature traits to avoid being identified by traditional signature-based malware detection models. To identify malicious threats or malware, this report investigates the application of machine learning (ML) and artificial intelligence (AI) to enhance early-stage malware detection and examine performance evaluation. The primary goal is to assess the potential of various machine learning models—spanning supervised and deep learning approaches—in identifying novel malware patterns and detecting anomalies within large datasets. A Kaggle dataset containing both malware and benign samples is used to assess the several classifiers' effectiveness. By focusing on critical aspects such as feature selection, model accuracy, and real-time detection efficiency, this study aims to develop a comprehensive framework for scalable, adaptable, and robust threat detection.**

To provide a well-rounded perspective, the report evaluates the performance of different ML classifiers using standardized metrics, including accuracy, True positive rate (TPR), and false negative rate (FNR). Results indicate that several algorithms, such as Random Forest Method (RFM) [85%], Decision Trees (DT) [82%], Sequential Neural Networks (SNN) [84.3%], Linear Regression LR [81%] and Support Vector Machines (SVM): Linear [81%], Poly [82.5%], RBF[82.6%] demonstrate high detection accuracy, with SNN reaching [84.3%] on the tested dataset. The outcomes emphasize the potential of AI-driven approaches in overcoming the limitations of traditional methods, offering a promising pathway toward enhanced cybersecurity solutions that can proactively respond to evolving malware threats.

Keywords—malware, machine learning, artificial intelligence, supervised

1. INTRODUCTION

The internet has revolutionized society, offering numerous benefits, but it has also enabled cyberattacks targeting individuals and corporations. Antivirus companies continuously strive to improve methods for identifying potential threats like malware. Traditional malware detection techniques have historically played a crucial role in cybersecurity, yet they face significant challenges in combating sophisticated modern threats [1].

Malware detection typically relies on two primary methods: static and dynamic analysis. **Static analysis**, often synonymous with signature-based detection, examines software without executing it. This approach analyzes the structure, behavior, and content of files to identify malicious patterns or code snippets [16]. While static analysis is efficient and quick, it struggles against polymorphic and obfuscated malware, which can modify their code to evade detection. **Dynamic analysis**, by contrast, involves running suspicious programs in controlled environments, such as virtual machines or sandboxes, to monitor their behavior in real time [2]. This adaptive method can detect malicious activities even when sophisticated evasion techniques are used. However, dynamic analysis is resource-intensive, requiring significant computational power and expertise. Additionally, some advanced malware can detect sandbox environments and modify their behavior to appear benign, further complicating detection. While traditional methods remain foundational, their limitations highlight the need for innovative approaches to address the evolving complexity of cyber threats.

1.1 RESEARCH PROBLEM:

Malware detection can be accomplished through two main approaches: static and dynamic analysis. However, a key challenge in malware detection is optimizing feature selection. Current detection methods often rely on numerous features, potentially making the analysis process unnecessarily complex and time-consuming. Research suggests that using fewer, more carefully selected features could maintain or even improve detection accuracy while streamlining the analysis process.

1.2 RESEARCH GOAL

The goal is to develop detection methods that can:

- Effectively identify previously unknown malware variants
- Design and analyze feature selection methods and operate with a reduced feature set
- Maintain high detection accuracy
- Assess model scalability and adaptability for detection performance.

2.0 LITERATURE REVIEW

Detecting malware is increasingly challenging due to the evolving techniques employed by malware developers to evade detection and remain concealed. Traditional antivirus software and manual analysis are often inadequate against emerging threats, necessitating the continuous adaptation of security measures.

Recent advancements have explored machine learning (ML) and deep learning as promising approaches to malware detection. Studies by Jin et al. [6] and Sethi et al. [7] introduced methods like autoencoders and decision trees, emphasizing the effectiveness of feature selection and sandboxing for dynamic malware analysis. Similarly, research by Darem et al. [9] employed concept drift detection and sequential deep learning, achieving notable accuracy improvements. Building on these efforts, the current study evaluates a broader range of ML algorithms and validates their performance using a comprehensive malware dataset.

In the paper by Tabish et al. [10], the authors utilized various ML techniques for multi-class classification to detect seven categories of malware, including benign files. Their novel approach demonstrated the ability to detect obfuscated and packed malware, addressing the challenge posed by intentionally rewritten code designed to evade detection. Using a dataset of 15,036 files (5,016 benign and 10,020 malicious), the study trained its models on 80% of the dataset, leveraging statistical features derived from byte-sequence n-grams of executable files.

Another study, Alazab et al. [11], focused on API functions for feature representation. The research achieved a precision of 97.6% with a false-positive rate of 0.025 using a Support Vector Machine with a normalized polynomial kernel. Motivated by the limitations of traditional antivirus systems, this ML framework achieved an impressive 98.5% accuracy rate in detecting previously unknown malware.

The paper *Survey on the Usage of Machine Learning Techniques for Malware Analysis by Bazrafshan et al. [12]* identifies three primary methods for detecting malicious software: signature-based, heuristic-based, and behavior-based approaches. It also explores concealment techniques used by malware to evade detection but does not delve into dynamic or hybrid approaches. The study provides an overview of ML applications in malware analysis, particularly in Windows environments, highlighting that poorly trained models can lead to inefficient algorithms and limited predictive capabilities.

These findings underscore the importance of robust ML models and comprehensive datasets in advancing malware detection and addressing the evolving cyber threat landscape.

2.1 MALWARE TYPES:

- **Backdoor**: A type of malware that bypasses normal authentication processes to gain unauthorized access to a system. It allows remote control over application resources, such as file servers and databases, enabling attackers to execute system commands and update malicious software without detection.
- **Rootkit**: A malicious program that grants attackers unauthorized access with elevated privileges, such as administrative rights. Rootkits are designed to conceal their presence, making them difficult to detect and remove, often operating undetected within the system.
- **Keylogger**: A form of malware that records every keystroke entered by a user. It captures sensitive information such as passwords, credit card details, and other confidential data, which can then be exploited by the attacker.
- **Ransomware**: A type of malware that encrypts the victim's data, rendering it inaccessible, and demands a ransom payment in exchange for the decryption key. Once infected, the system is typically locked, with the attacker's instructions displayed, preventing the user from accessing files or functionality until payment is made.

3.0 METHODOLOGY

3. 1 MACHINE LEARNING METHOD

Machine learning (ML), a branch of artificial intelligence, enables systems to analyze vast datasets, identify patterns, and make informed decisions or predictions. This adaptability makes ML particularly effective in malware detection. By training on extensive datasets containing both benign and malicious files, ML models can uncover subtle and complex features that distinguish safe software from harmful code, making them valuable for combating evolving malware threats [13-14].

For instance, if a user's account is compromised, it may exhibit unusual network behavior, such as unexpected data transfers or connections to suspicious servers. ML algorithms can detect such anomalies and flag them for review. Similarly, at the system level, ML can identify irregular activities like privilege escalations or unusual changes in resource usage, providing an additional layer of defense.

Deep learning, a specialized subfield of ML, shows exceptional promise in malware detection. Models like sequential neural networks (SNNs) can learn hierarchical representations of data, capturing intricate patterns and correlations that traditional methods might lack. For example, a deep learning model can identify recurring sequences of system calls common in malware but rare in legitimate software—patterns too complex for signature-based detection methods to recognize.

3.2 ADVANTAGES OF ML IN MALWARE DETECTION

1. **Feature Extraction and Selection**: ML algorithms can automatically identify and select key features from malware samples, such as API calls, file structures, code patterns, or behavioral traits. This automation improves detection accuracy and reduces false positives by focusing on distinguishing characteristics of malware.
2. **Enhanced Detection Accuracy**: ML models excel at identifying complex and previously unknown malware variants by analyzing large datasets. Unlike traditional methods that rely on static signatures of known threats, ML algorithms learn to detect the malicious patterns underlying malware

behavior, making them effective against polymorphic viruses that alter their code to evade detection.

3.3 ADVANCED ML TECHNIQUES FOR MALWARE DETECTION

- **StringExtraction**:
  String extraction is a technique used to isolate and analyze text segments from files to identify potentially malicious patterns. It involves extracting relevant text components from a file's content for analysis. ML algorithms can then process these extracted strings to identify patterns or features indicative of malware. This method is especially useful for analyzing large datasets, significantly improving the accuracy and efficiency of malware detection [11 & 16]. By leveraging techniques like string extraction and employing deep learning models, ML provides a robust framework for identifying sophisticated malware, offering advantages over traditional, static methods and addressing the challenges posed by advanced and adaptive malicious threats.

This report characterizes and illustrates the transformative potential of machine learning in enhancing malware detection and prevention. The adopted techniques follow the illustration in Fig1 and involves 6 main stages:

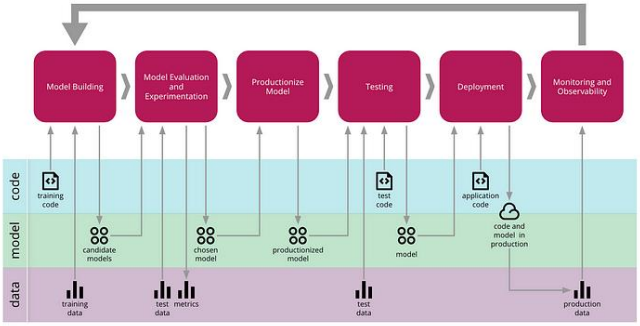

Fig 1: Main ML process.

Step 1. Data Collection:

We collected and organized a substantial dataset suitable for analysis by loading it from files into memory. Specifically, we utilized a Kaggle dataset containing both benign and malicious samples and began by cleaning and preprocessing the data to eliminate any anomalies. This data set included numerous files with log data for various types of malwares, as noted in Akhtar et al. [13-14]. The extracted log features were then used to train several models. Overall, the dataset comprised over 15,036 data points from different sources, featuring 215 columns and 15,036 rows.

Step 2. Data Pre-processing:

Here, we began by cleaning, preparing, and transforming the data for analysis. This involved standardizing the data, so it fell within the same ranges and formats, which included removing duplicates and handling missing values. The original data were stored as binary code in the file system, consisting of unprocessed executable files. We processed these files ahead of the analysis to ensure they were in a suitable format.

Step 3. Feature Extraction and Selection:

Twentieth-century datasets often contained tens of thousands of features. However, as feature counts have grown, overfitting has become a significant challenge in machine learning models (Selamat & Ali [15]). Extracting data features to reveal patterns and relationships is a critical step, as feature quality directly impacts model performance. To address overfitting, a smaller subset of features was selected from a larger pool to maintain accuracy with fewer variables. This study aimed to refine a dataset of dynamic and static features by retaining the most relevant ones and discarding those with minimal impact on analysis [15]. After extracting features, feature selection was conducted to enhance accuracy, simplify the model, and reduce overfitting by identifying the most useful features. Researchers have previously used various classification strategies to detect malicious code, with feature ranking proving effective in selecting features for malware detection models [16-17]. Finally, the data was divided into training and test sets, with the training set used to build the model and the test set for evaluation.

Step 4. Model Training:

The machine learning algorithm is trained on the extracted features to learn the patterns and relationships in the data.

Step 5. Model testing and Evaluation:

The performance of the model is evaluated on a separate test dataset to test its accuracy and generalizability.

Step 6. Model Deployment:

The trained model is deployed to make predictions or decisions on new data. At this stage, the best model is selected (either after the defined number of iterations or as soon as the needed result is achieved.

3.4 METHODS AND ANALYSIS:

The report follows a structured methodology involving data collection, preprocessing, feature extraction, and model evaluation as shown below, Fig2:

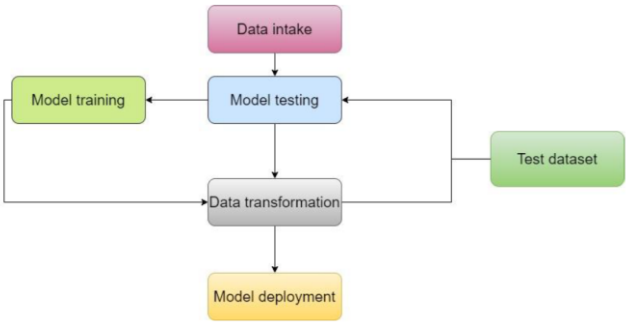

Fig 2: Modeling workflow.

3.5 DATASET

We commence our investigation using a dataset sourced from Kaggle, which contains both benign and malicious Android application samples. To ensure the integrity and reliability of our analysis, we first perform data cleaning and preprocessing steps to eliminate any anomalies or inconsistencies within the dataset. Following this, we focus on feature extraction by analyzing critical indicators such as Android application permissions and API calls. We then employ feature selection techniques to identify the most relevant features that significantly contribute to distinguishing between benign and malicious applications.

Subsequently, we train and evaluate multiple machines learning classifiers, including Extreme Gradient Boosting (XGBoost), Random Forest, Support Vector Machine (SVM), and Decision Tree models. To validate the robustness and generalizability of these models, we utilize 10-fold cross-validation during the training and testing phases. The performance of each classifier is rigorously assessed using a range of metrics: accuracy, the Area Under the Receiver Operating Characteristic Curve (AUC), false positive rate, and false negative rate. This comprehensive evaluation allows us to determine the effectiveness of each model in correctly classifying Android applications and in minimizing misclassification errors.

4.2 Result & Discussion:

After obtaining the processed dataset, we performed standardization and initial data treatment to ensure consistency and reliability. Immediately following these preprocessing steps, we conducted a comprehensive data analysis to prepare for training the predictive models. Figure 3 displays a sample of the features selected after applying the feature extraction algorithm. With these relevant features extracted and selected, we applied various algorithms to the refined dataset to develop and train the prediction models.

```
print("Total missing values : ",sum(list(data.isna().sum())))
data.info()
```

```
Total missing values :  0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15036 entries, 0 to 15035
Columns: 216 entries, transact to class
dtypes: int64(214), object(2)
memory usage: 24.8+ MB
```

```
corrdata = balanced_data.corr()
corrdata['class'][abs(corrdata['class'])>0.5]   #corr greater than 0.5 indicates good relation
```

```
transact                        -0.610301
onServiceConnected              -0.598012
bindService                     -0.596115
attachInterface                 -0.592081
ServiceConnection               -0.594336
android.os.Binder               -0.582618
SEND_SMS                         0.532169
Ljava.lang.Class.getCanonicalName -0.509296
class                            1.000000
Name: class, dtype: float64
```

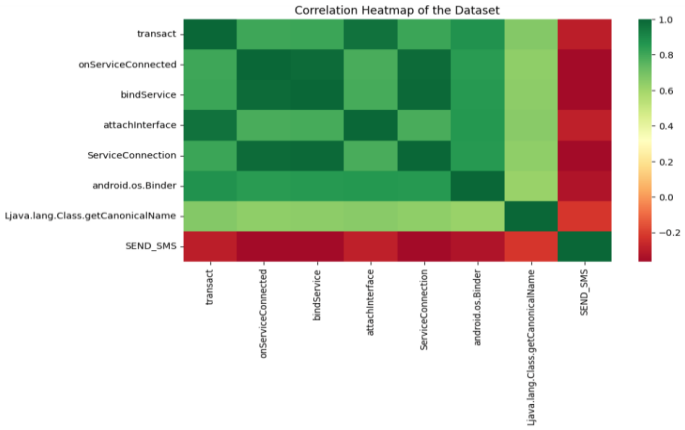

Fig. 3: Sample of the selected features.

3.5.1 EVALUATION CRITERIA

To evaluate the performance of the algorithms applied in this study, the following criteria are utilized:

- **Confusion Matrix**

The confusion matrix serves as a critical tool for assessing the performance of classification algorithms. It provides a summary of the number of correct and incorrect predictions made by a classifier, categorized by their respective count values. This matrix facilitates the calculation of key metrics, such as false positive and false negative rates, which are instrumental in performance evaluation.
The structure of a typical confusion matrix is as follows:

	Actually Positive (1)	Actually Negative(0)
Predicted Positive (1)	True Positive (TP)	False Positive (FP)
Predicted Negative (0)	False Negative (FN)	True Negative (TN)

Table 2: Confusion Matrix

- **False Positive Rate:** The false positive rate quantifies the proportion of negative samples that have been incorrectly classified as positive. It is computed using the formula:

$$FPR = FP/(FP + TN)$$

- **False Negative Rate:** The false negative rate measures the proportion of positive samples that yield negative test outcomes. It is calculated as:

$$FNR = FN / (FN + TP)$$

- **Accuracy:** Accuracy reflects the proportion of correctly classified samples (both positive and negative) relative to the total number of samples. It is expressed as:

$$ACC = (TP + TN) / (TP + FP + FN + TN)$$

For the preliminary analysis, several supervised machine learning algorithms were employed to investigate and characterize malware. The algorithms used include the

Random Forest Method (RFM), Decision Trees (DT), Sequential Neural Networks (SNN), Linear Regression (LR), and Support Vector Machines (SVM) with various kernels such as Linear, Polynomial, and Radial Basis Function (RBF).

The training outcomes of the Random Forest Method (RFM) models are summarized below, highlighting the performance metrics derived from the confusion matrix. Further detailed analysis of these results provides insights into the efficacy of each classifier in detecting and categorizing malware.

```
batch_size = 32
epochs = 10

history = model.fit(XX_train, yyy_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)

y_pred = (model.predict(XX_test) > 0.5).astype(np.int32)
accuracy = accuracy_score(yyy_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Epoch 1/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8019 - loss: 0.4062 - val_accuracy: 0.8220 - val_loss: 0.3564
Epoch 2/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8285 - loss: 0.3421 - val_accuracy: 0.8352 - val_loss: 0.3516
Epoch 3/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8319 - loss: 0.3459 - val_accuracy: 0.8352 - val_loss: 0.3508
Epoch 4/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8268 - loss: 0.3474 - val_accuracy: 0.8345 - val_loss: 0.3511
Epoch 5/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8301 - loss: 0.3400 - val_accuracy: 0.8352 - val_loss: 0.3506
Epoch 6/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8265 - loss: 0.3447 - val_accuracy: 0.8352 - val_loss: 0.3507
Epoch 7/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8382 - loss: 0.3234 - val_accuracy: 0.8352 - val_loss: 0.3500
Epoch 8/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8326 - loss: 0.3364 - val_accuracy: 0.8352 - val_loss: 0.3487
Epoch 9/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8151 - loss: 0.3520 - val_accuracy: 0.8352 - val_loss: 0.3492
Epoch 10/10
427/427 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8345 - loss: 0.3296 - val_accuracy: 0.8359 - val_loss: 0.3523
119/119 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Accuracy: 0.84
```

```
# Print a classification report which includes precision, recall, F1-score, and support for each class.
print("Classification Report:")
print(classification_report(test_y, y_pred))

conf_matrix = confusion_matrix(test_y, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

accuracy = accuracy_score(test_y, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.74      0.83      1860
           1       0.79      0.95      0.86      1931

    accuracy                           0.85      3791
   macro avg       0.86      0.84      0.84      3791
weighted avg       0.86      0.85      0.84      3791

Confusion Matrix:
[[1373  487]
 [  95 1836]]
Accuracy: 0.85
```

Fig. 4: Implementation of Random Forest Algorithm (RFM) and Sequential Neural Network (SNN)

- The accuracy for the test and the training dataset is shown above in Fig.4 and for SNN, it is 0.835 and 0.843 respectively.
- Similarly, for RFM, the model accuracy for the testing dataset is about 0.85.

The confusion matrix for RFA is given below in Fig.5.

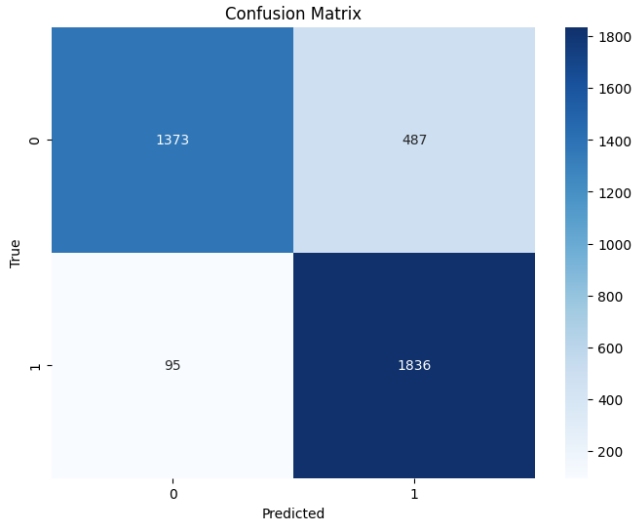

Fig. 5: Confusion Matrix for RFA.

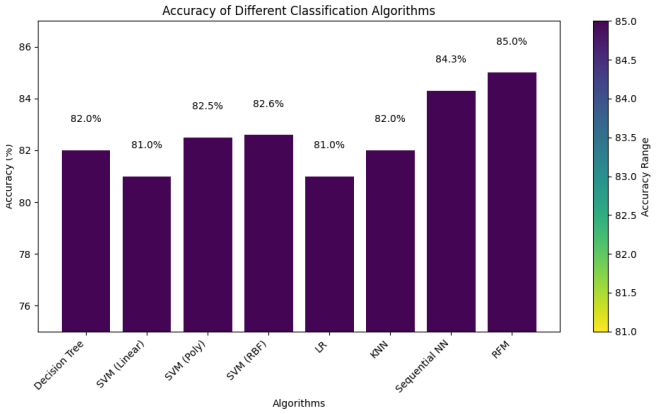

Fig. 6: Model accuracy and performance comparison.

Models	Accuracy(%)	TPR(%)	FPR(%)
SNN	84	0.9415	0.2656
RFM	85	0.9508	0.2618

Table 1: Best Methods result comparisons.

The rapid growth of malicious software continues to pose a critical threat to global cybersecurity. This issue emerged prominently in the 1990s, coinciding with the exponential increase in interconnected computer systems, which facilitated the proliferation of malware [23]. This surge in malware distribution has necessitated the development of various protective measures. However, these conventional security mechanisms often fail to keep pace with the increasingly sophisticated tactics employed by malware developers to evade detection.

In response to these challenges, recent research has shifted its focus toward leveraging machine learning (ML) algorithms for malware detection. ML-based approaches offer the potential to identify patterns and anomalies that traditional methods might overlook. This report introduces a protective framework that evaluates eight (8) distinct ML algorithms for malware detection, ultimately selecting the most effective model based on performance metrics. Our findings reveal that the Random Forest algorithm demonstrates superior performance compared to other methods. As shown in Table 1, Random Forest achieves the highest detection accuracy, with an 85% success rate, while maintaining the lowest false positive rate (FPR) of 0.26 on a reduced dataset. These results highlight the algorithm's effectiveness in balancing detection precision with minimizing false alarms, making it a promising solution for contemporary malware threats.

4.3 DISCUSSION:

In Figure 3, the dataset's initial 215 features are presented. However, after applying a feature selection process using the Filter method and incorporating significant correlation parameters, the feature set was reduced to eight. This feature selection technique effectively removed redundant and irrelevant features, enhancing the prediction accuracy. The implementation was carried out in a Python notebook utilizing various libraries designed for machine learning algorithms. Prior to analysis, data preprocessing techniques were employed to prepare the dataset. These included encoding categorical variables and addressing missing values, ensuring the data was organized into a format suitable for machine learning analysis. Figures 4 and 5 display the results of applying the Random Forest Method (RFM) and Sequential Neural Network (SNN) algorithms, respectively.

The training dataset was used to train all applied algorithms, which were then evaluated using the test dataset. The SNN model achieved a training accuracy of 0.835 and a testing accuracy of 0.843. In contrast, the RFM algorithm, which is a robust ensemble method consisting of multiple decision trees, demonstrated superior performance. The concept of Random Forest involves growing multiple decision trees based on independent subsets of the dataset. At each decision node, a random selection of features is considered, and the best split is determined. The training accuracy of RFM was 0.87, with a testing accuracy of 0.85.

Figure 5 provides the confusion matrix derived from the RFM method, while Figure 6 offers a comparative visualization of the performance of all applied algorithms. According to the experimental results, the accuracy of all classifiers exceeds 80%, with RFM achieving the highest accuracy of 85%. Additionally, RFM outperforms other models in terms of false positive and false negative rates. Following RFM, the SNN algorithm exhibited the second-highest accuracy of 84% with a false positive rate of 0.26. The Support Vector Machine (SVM) classifier ranks next in terms of accuracy. The overall analysis indicates that RFM is the most accurate and reliable algorithm among those evaluated, with an accuracy of 85% and an area under the curve (AUC) value of 0.95. Consequently, RFM can be considered the optimal classifier for this dataset. The sequential neural network follows closely in performance. Figure 6 and Table 1 provide a detailed comparison of the accuracy and performance metrics of all the algorithms, confirming RFM as the best-performing model in this study.

5. CONCLUSION

This study highlights the increasing focus of researchers on employing machine learning (ML) algorithms for malware detection. We proposed a selective evaluation framework that examined eight ML algorithms to determine the most effective approach for malware identification. The experimental results indicate that all the evaluated algorithms performed well in terms of detection accuracy, with the following accuracy rates: Decision Trees (DT) at 82%, Support Vector Machines (SVM) with Linear (81.04%), Polynomial (82.5%), and Radial Basis Function (RBF) kernels (82.6%), Linear Regression (81%), k-Nearest Neighbors (KNN) at 82.4%, Sequential Neural Networks (SNN) at 84.3%, and Random Forest Method (RFM) at 85.4%.

Among the classifiers, RFM demonstrated the highest detection accuracy (85.4%) and the most effective performance in minimizing false positives, achieving a false positive rate (FPR) of 0.2618%. Similarly, SNN exhibited strong detection capability with an accuracy of 84.3% and a comparable FPR of 0.2656%. These results underscore the efficacy of RFM and SNN in detecting malware with high precision and minimal false alarms.

In addition to these findings, the research underscores the potential of supervised static analysis based on Portable Executable (PE) file information. By integrating carefully curated data and leveraging advanced ML algorithms, this approach shows significant promise for accurately identifying and characterizing malware. The superior performance of the RFM method, in particular, highlights its utility as the most reliable classifier among those evaluated, offering robust detection accuracy and effective mitigation of false positives. These results establish a strong foundation for future research in improving malware detection systems through ML-driven approaches.

Acknowledgement:

We would like to express our sincere appreciation for the invaluable assistance provided by ChatGPT in enhancing our project documentation and improving the overall quality of our written materials. Our team initially conceptualized the ideas and frameworks that underpin this project; however, we recognized the need for precision and clarity in our documentation. By leveraging AI applications, particularly ChatGPT, we were able to fine-tune our content effectively.

==Grammar corrected using ChatGPT for acknowledgement😬

References

1. S. Sharma, C.R. Krishna, S. K. Sahay," Detection of advanced malware by machine learning techniques". In Proceedings of the SoCTA, Jhansi, India, pp. 22–24 December 2017.

2. K. Zhao, D. Zhang, X. Su, W. Li, W. Fest "A feature extraction and selection tool for android malware detection". In Proceedings of the 2015 IEEE Symposium on Computers and Communication (ISCC), Larnaca, Cyprus, pp. 714–720 6–9 July 2015.

3. P.R.K. Varma, K.P. Raj, K.V.S. Raju, "Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms". In

Proceedings of the 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, pp. 294–299, 10–11 February 2017.

4. C. Neelam, S. Arora, G. Gupta, "Android malware Detection Using Improvised Random Forest Algorithm". Global journal for research analysis: ISSUE-3, volume-9, MARCH – 2020. DOI: 1036106/gjra, PRINT ISSN No. 2277 – 8160.

5."Android Malware Dataset for Machine Learning" https://www.kaggle.com/shashwatwork/android-malwaredataset-for-machine-learning.

6. X. Jin, X. Xing, "A Malware Detection Approach Using Malware Images an Autoencoders" IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)–2020.DOI:10.1109/MASS50613.2020.00009.

7. K. Sethi, R. Kumar, "A Novel Machine Learning Based Malware Detection and Classification Framework" International Conference on Cyber Security and Protection of Digital Services, pp. 1-13, 3-4June 2019.

8. D. Wu, P. Guo, "Malware Detection Based on Cascading XGBoost and Cost Sensitive", International Conference on Computer Communication and Network Security (CCNS), 2020 IEEE, DOI:

10.1109/CCNS50731.2020.00051.

9. A. Abdulbasit, F. Darem, A. Ghaleb, "An adaptive Behavioral-Based Incremental Batch Learning Malware Variants Detection Model Using Concept Drift Detection and Sequential Deep Learning" IEEE Access, Volume 9,14 July 2021, DOI:10.1109/ACCESS.2021.3093366.

10. M. Tabish, Z. M. Shafiq, M. Farooq, "Malware detection using statistical analysis of byte-level file content". In Proceedings of the ACM SIGKDD Workshop on Cyber Security and Intelligence Informatics, pp 23–31. ACM, 2009.

11. M. Alazab, V. Sitalakshmi, W. Paul, A. Moutaz, "A Zero-day malware detection based on supervised learning algorithms of API call signatures". In proceedings of the ninth Australasian data mining conference, 121, 171-182. Australian Computer Society. DOI: 10.5555/248 3628.2483648, 2011.

12. Z. Bazrafshan, H. Hashemi, S.M.H. Fard, A. Hamze, "A survey on heuristic malware detection techniques Conference: Information and Knowledge Technology (IKT), (2013).

13. M.S. Akhtar, T. Feng, "Malware Analysis and Detection Using Machine Learning Algorithms. *Symmetry*, *14*(11), 2304, 2022. https://doi.org/10.3390/sym14112304

14. M. Damshenas, A. Dehghantanha, R. Mahmoud, "A survey on malware propagation, analysis and detection. *Int. J. Cyber-Secur. Digit. Forensics*, Vol. *2*, pp. 10–29, 2013.

15. N. Selamat, F. Ali, "Comparison of malware detection techniques using machine learning algorithm". *Indones. J. Electr. Eng. Comput. Sci.* Vol. *16*, pp. 435, 2019.

16. F. Hamid, "Enhancing malware detection with static analysis using machine learning". *Int. J. Res. Appl. Sci. Eng. Technol.* Vol. *7*, pp. 38–42, 2019.
17. K. Prabhat, G.P. Gupta, R. Tripathi, "A trustworthy privacy-preserving secured framework for sustainable smart cities by leveraging blockchain and machine learning". *J. Syst. Archit.* Vol. *115*, 101954, 2021