

Malware Classification Leveraging NLP & Machine Learning for Enhanced Accuracy

Bishwajit Prasad Gond¹ , Rajneekant¹, Pushkar Kishore¹, and Durga Prasad Mohapatra¹

National Institute of Technology Rourkela, Odisha, India
bishwajitprasadgond@gmail.com, rajneekant200@gmail.com,
monumit46@gmail.com, and durga@nitrkl.ac.in

Abstract. This paper investigates the application of natural language processing (NLP)-based n -gram analysis and machine learning techniques to enhance malware classification. We explore how NLP can be used to extract and analyze textual features from malware samples through n -grams, contiguous string or API call sequences. This approach effectively captures distinctive linguistic patterns among malware and benign families, enabling finer-grained classification. We delve into n -gram size selection, feature representation, and classification algorithms. While evaluating our proposed method on real-world malware samples, we observe significantly improved accuracy compared to the traditional methods. By implementing our n -gram approach, we achieved an accuracy of 99.02% across various machine learning algorithms by using hybrid feature selection technique to address high dimensionality. Hybrid feature selection technique reduces the feature set to only 1.6% of the original features.

Keywords: API calls · Malware Classifier · n -grams · Portable executable

1 Introduction

In the relentless cat-and-mouse game of cybersecurity, where threats perpetually adapt and multiply, it has become increasingly crucial to deploy innovative techniques for malware detection and classification. Malwares, spanning a wide spectrum of categories such as adware, viruses, worms, spyware, downloaders, trojans, and backdoors, pose an escalating challenge to security experts worldwide. To combat this diverse landscape of threats, the incorporation of Natural Language Processing (NLP)-based n -gram [1] analysis, particularly n -grams based feature selection, has emerged as a promising avenue for enhancing malware classification. This paper explores the application of n -grams analysis, coupled with machine learning, to classify malwares into these above seven distinct categories.

Beyond traditional signature-based [2] and heuristic-based approaches [3], which often struggle to cope with polymorphic and obfuscated malware [4], our methodology leverages the unique linguistic footprints exhibited by different malware categories. Specifically, we focus on the text extracted from the Application Programming Interface (API) calls of Portable Executable (PE) files in Windows environments [5].

Our research focuses on fine-grained malware classification by leveraging n -grams and machine learning techniques to identify intricate patterns in API call sequences

of different malware variants. This novel approach promises accurate and nuanced classification, providing a robust defense against evolving threats.

Coupling n -grams with machine learning in malware classification is an inventive approach enhancing our ability to discern subtle behavioral and structural differences. Utilizing API call sequences in PE files broadens the scope beyond static analysis to dynamic behavior monitoring.

As the cybersecurity community seeks more effective and adaptive defenses, this research aims to contribute by exploring n -grams, machine learning techniques, and API call sequences for improved malware categorization accuracy and efficiency.

Dynamic analysis based on API calls tends to be more effective than static analysis [6]. This paper investigates using natural language processing (NLP) techniques for dynamic malware analysis based on n -gram API calls, leveraging ensemble and machine learning methods like boosting (XGBoost and LightGBM) and bagging (Random Forest) to enhance malware classification accuracy.

The rest of the paper is organized as follows: in Section 2 we covered the basic concepts. In Section 3, we review recent literature using API sequence and NLP in malware classification and detection. Section 4 outlines our NLP-based framework for malware detection, focusing on preprocessing and the NLP process. Section 6 details our experimental setup, including obtaining an API key, data extraction and malware classification. Section 7 covers the detail result analysis. In Section 8, we compare our work with state-of-the-art techniques in malware detection and classification. Finally, Section 9 concludes our work and outlines some future research directions.

2 Basic Concepts

In this section, we first discuss the static analysis of malware, followed by dynamic analysis. We also address the challenges associated with these approaches. Furthermore, we delve into sandbox environments and explore current trends in malware analysis.

2.1 Static Analysis

Static analysis dissects malware's binary code, file structure, strings, and metadata without execution. It identifies functions, suspicious code, attack vectors, and indicators of compromise (IOCs), aiding rapid threat detection and classification. However, it has limitations in uncovering behavior details, especially with obfuscation techniques like code encryption, packing, anti-analysis measures, polymorphism, and control flow obfuscation. Security experts often combine static analysis with dynamic analysis and reverse engineering for comprehensive understanding.

2.2 Dynamic Analysis

Dynamic analysis involves executing malware in a controlled sandbox environment to observe real-time behavior, crucial for understanding capabilities and developing countermeasures. As malware sophistication grows with polymorphic and metamorphic coding, dynamic analysis gains prominence. Sandboxes simulate real environments, capturing behaviors like file system, registry, network activities, enabling comprehensive monitoring.

Current trends focus on automation and machine learning for efficient large-scale analysis and accurate pattern recognition. Automated tools rapidly process vast samples, identifying common malicious behaviors. Machine learning techniques enhance threat detection accuracy, adapting to evolving tactics.

3 Related Works

. This section discusses research on malware detection and classification related to Natural Language Processing (NLP) and API Call.

Nakazato et al. [7] proposed an approach to detect and classify malware in their malwares using API sequence. They conducted dynamic analysis to automatically obtain the execution traces of malware and then classified malwares into clusters based on their behavior characteristics derived from Windows API calls in parallel threads. The authors utilized two NLP techniques, namely n -gram and TF-IDF, to infer the characteristics of malware samples. This proposed methodology successfully classified 90% of 2312 malware samples into maximum of 20 different clusters.

Another research on malware related to API sequences and machine learning was proposed by Chandrasekar et al. [8], who used a 3^{rd} order Markov chain to model Windows API call sequences. Their proposed malware detection system tested on a dataset containing 94 benign and 179 malware executables, achieved an accuracy of 90%.

Windows API calls are highly deterministic features for behavior-based malware detection, accurately reflecting program behavior during execution and effectively distinguishing malware from benign programs [9, 10]. API call information can identify evasive malware. Hence, many malware detection techniques utilize methods for extracting and leveraging API call information.

Ye et al. [10] developed an intelligent malware detection system, IMDS, using API features. They extracted API calls accessed by PE files through static analysis and used these features as input for classification algorithms, achieving a detection accuracy of 93.07%. Additionally, authors proposed a layered classification framework to determine malicious operations performed by malware samples using API call features. They applied various feature selection methods to obtain discriminative features, resulting in an accuracy of 98.6%.

Dabas et al. [11] introduced a malware detection method for Windows based on API calls, using three feature sets: (a) API calls usage, (b) frequency, and (c) sequences, enriched with TF-IDF to create the API integrated feature set. Malware samples are retrieved from the VirusShare data repository having 2500 malware and 2500 benign samples from freshly installed Windows 10 OS. It achieves 99.6% accuracy across ML algorithms, addressing high-dimensionality with hybrid feature selection, reducing feature set to 9% .

Dabas et al. [12] presented MalAnalyser, a lightweight Windows malware detection system based on frequent API call subsequences. The authors collected malwares from VirusShare platform having 2500 malware and 2500 benign samples from freshly installed Windows 10 OS. It uses GLBPSO and GA for feature selection, achieving up to 99.7% accuracy with 30% features, and 100% accuracy on GA-enriched features.

MalAnalyser outperforms similar approaches on a benchmark dataset with 99.72% accuracy.

Sharma et al. [13] proposed an approach which paper focuses on Windows malware detection using TF-IDF enriched API call information. They combined API call feature sets into an integrated set and applied TF-IDF for feature importance and the dataset was obtained from Virusshare. The TF-IDF enriched integrated API calls feature set achieves up to 99.91% accuracy for SVM and Logistic Regression algorithms.

4 Proposed Framework

In this section, we present our proposed framework for malware classification using NLP and machine learning techniques. Our proposed architecture (Fig. 1) involves:

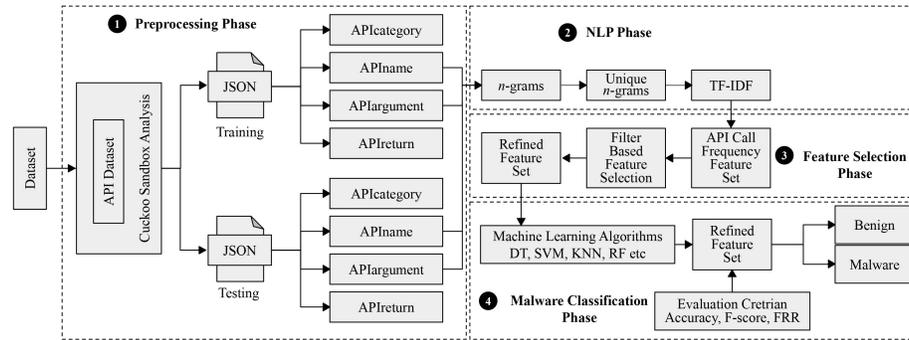


Fig. 1: Proposed Architecture for Malware Classification

1) acquiring malware hash from VirusShare¹, 2) querying VirusTotal² for JSON file with antivirus scans to determine malware class, 3) downloading distinct malware categories, 4) dynamic analysis in Cuckoo sandbox³ to extract API call sequences in JSON format, 5) partitioning JSON report into API name, argument, return, and category text files, 6) applying n -gram methods combining API names and arguments, 7) calculating TF-IDF using unique n -grams from all categories, 8) applying hybrid feature selection techniques to get refined feature set, 9) applying machine learning techniques and adjusting evaluation criterion on refined feature set. The four phases are:

- ❶ **Preprocessing Phase:** Steps 4 and 5.
- ❷ **NLP Phase:** Step 6 and 7.
- ❸ **Feature Selection Phase:** Step 8.
- ❹ **Malware Classification Phase:** Step 9.

Phase 1: Preprocessing Phase

In preprocessing phase, we perform the following (Fig. 2) activities:

1. **Preprocess the Data:** After obtaining the data for each class, we performed behavioral analysis using Cuckoo Sandbox, resulting in a behavioral report in JSON

¹ <https://virusshare.com>

² <https://www.virustotal.com>

³ <https://cuckoosandbox.org>

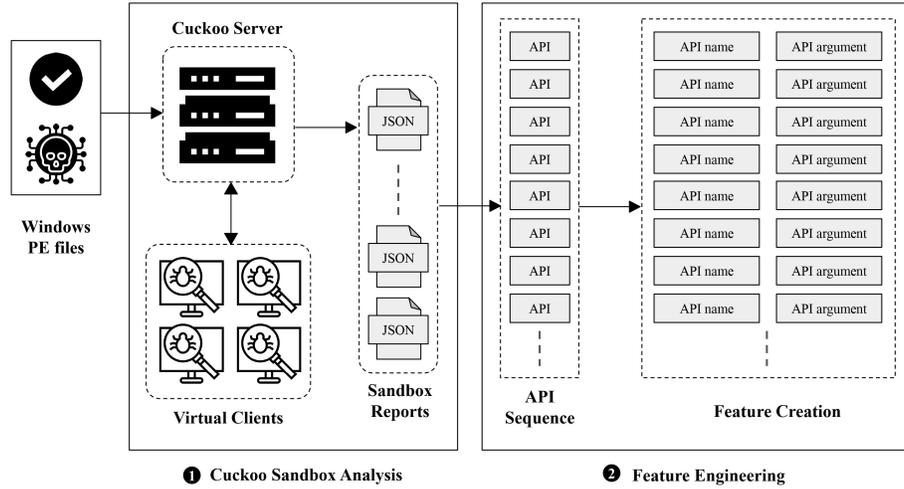


Fig. 2: Data Preprocessing and Feature Engineering

format. We then split the file into four parts: API category, API name, API argument, and API return. From this, we selected the API name and argument to create n-grams, where the API name is the first part and the API call argument is added using underscores. Finally, we obtained a CSV file containing n-grams for each category.

- API Dataset & Cuckoo Sandbox Analysis:** The process starts with taking a dataset that undergoes Cuckoo Sandbox Analysis which results in the behavioural report of Portable Executable in JSON format.
- Data Division:** The data is divided into training and testing sets in 80% and 20%, both processed through JSON format.
- API Elements Extraction:** Various API elements like APICategory, APIName, APIArgument, and APIreturn are extracted.

Phase 2: NLP Phase

In NLP phase, we perform the following activities:

- n-grams and Unique n-grams:** We extract n-grams and select Unique n-grams from the JSON.
- n-grams after Processing:** The followings are the examples of unigram, bigram and trigrams that we have used in this paper.
 - Unigram:** *LdrLoadDll_urlmon_urlmon.dll*
 - Bigram:** *NtAllocateVirtualMemory_na, LdrLoadDll_ole32_ole32.dll*
 - Trigram:** *LdrUnloadDll_SHELL32, LdrLoadDll_SETUPAPI_SETUPAPI.dll, LdrGetProcAddress_ole32_OleUninitialize*

3. **Vectorization with TF-IDF:** It tokenizes text, counts occurrences of each token, and computes TF-IDF weights, which reflects the importance of each token in a document relative to the entire corpus. Optional L2 normalization ensures consistent feature vector lengths. TF-IDF is applied on each csv files of unigram, bigram and trigram with Unique n -gram csv file to transform the text data.
4. **Refined Feature Set:** A refined feature set is obtained after filtering the unnecessary feature of each n -grams CSV file.

Phase 3: Feature Selection Phase

Out of the millions of features available, we have applied several hybrid feature selection and reduction methods to eliminate redundant features. These methods eliminate features containing numbers, special characters, or underscores, which are prevalent in both malware and benign classes within our feature set. The goal is to identify and retain only the most relevant features for classification. This process helps reduce the dimensionality of the dataset, which can improve the efficiency and effectiveness of machine learning algorithms. By selecting the most informative features and discarding irrelevant or redundant ones, we aim at enhancing the performance of the classification model. In feature selection phase, we perform the following activities:

1. **API Call Frequency Feature Set:** We explore the features derived from API call frequency to understand the system behavior and usage patterns.
2. **Filter Based Feature Selection:** We apply filter-based techniques (e.g., mutual information, correlation analysis) to select the most informative features.
3. **Refine Feature Set:** We eliminate the redundant or irrelevant features to ensure that the final set is discriminative and predictive.

Phase 4: Malware Classification Phase

In malware classification phase, we perform the following activities:

1. **Applying Machine Learning Algorithms:** Algorithms such as Decision Tree [14], SVM with Kernel: linear, sigmoid, polynomial 3°, polynomial 4° and RBF, KNN, Random Forest [15], XGBoost, LightGBM are applied to the refined feature set obtained from phase three.
2. **Use Evaluation Criteria:** We use the evaluation criteria such as accuracy, precision, recall and F_1 score, etc. to determine the effectiveness of these ML algorithms.
3. **Classify the Malwares :** This phase involves distinguishing between benign and malware entities based on the features selected and evaluated in the previous phases.

To gain an in-depth understanding of the architecture, one can refer to the source code of our malware detector available on GitHub⁴

Our research utilizes malware samples collected from VirusShare. We obtained an API key to download over 200,000 JSON files (each corresponding to a unique hash), but were limited to four files per minute due to API constraints. From these JSON files, we extracted essential information, including scan results from 70 antivirus programs, which served as the basis for subsequent malware classification. We then performed malware classification, a crucial step to identify and categorize the samples for further analysis.

⁴ <https://github.com/bishwajitprasadgond/MalwareClassification>

5 Working Example

Example Documents:

- A: "This is a sample text document."
- B: "Here is another text document."

TF-IDF Calculation

1. **Term Frequency (TF):** $TF(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$
 $f_{t,d}$ is the term frequency, $\sum_{t' \in d} f_{t',d}$ is total terms in d .
2. **Inverse Document Frequency (IDF):** $IDF(t, D) = \log \left(\frac{N}{|\{d \in D : t \in d\}|} \right)$
 N is total docs, $|\{d \in D : t \in d\}|$ is docs containing t .
3. **TF-IDF:** $TF-IDF(t, d, D) = TF(t, d) \times IDF(t, D)$

Example:

- $IDF(\text{"sample"}) = IDF(\text{"another"}) = IDF(\text{"text"}) = IDF(\text{"document"}) = 0.3$
- $TF(\text{"sample"}, A) = TF(\text{"sample"}, B) = 1/5 = 0.2$
- $TF(\text{"another"}, A) = 0, TF(\text{"another"}, B) = 1/5 = 0.2$
- $TF(\text{"text"}, A) = TF(\text{"text"}, B) = 1/5 = 0.2$
- $TF(\text{"document"}, A) = TF(\text{"document"}, B) = 2/5 = 0.4$

TF-IDF Vectors:

- Document A: [0.06, 0, 0.06, 0.12]
- Document B: [0.06, 0.06, 0.06, 0.12]

6 Experimental Setup

Our experimental setup aims at evaluating the effectiveness of NLP and machine learning techniques in malware classification. It consisted of the follows:

- **Host OS:** We used Ubuntu 18.04 LTS on a machine with an Intel i7 processor, 8GB RAM, and a 10TB HDD. Cuckoo Sandbox 2.0.7 was employed for malware analysis on the Ubuntu host.
- We collected two lakhs diverse set of malware samples from VirusShare representing seven categories.
- A separate Windows 10 environment was used with an Intel i7 processor, 128GB RAM, and 5TB storage to collect and analyze the dynamic analysis reports obtained from Cuckoo Sandbox.
- Unigram, bigram and trigram analysis was conducted on Windows 10 host to transform API call sequences into n -grams of each categories.

This setup allowed controlled experimentation to assess the potential of n -gram and machine learning techniques in enhancing the performance of malware classification.

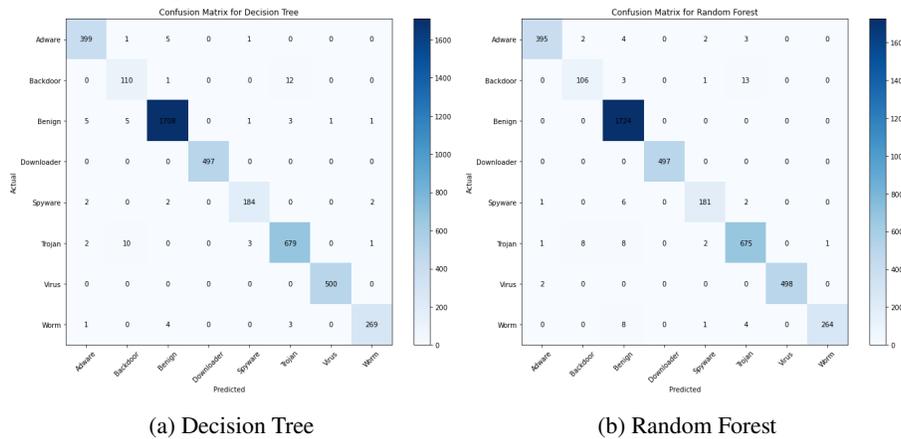
Our proposed approach was implemented using Python 3.10.9 programming language. Comprehensive information about our malware detector's experimental setup, our dataset, and source code can be accessed on GitHub⁴.

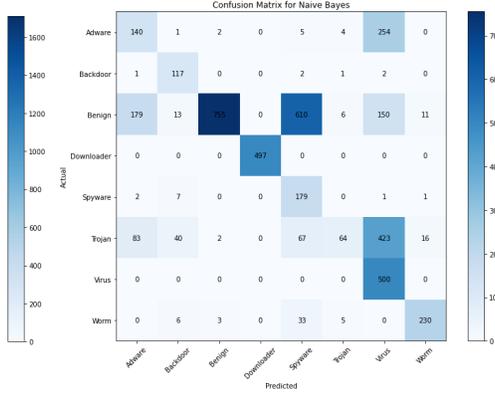
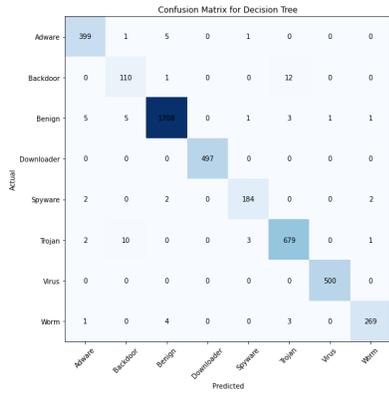
7 Results Analysis

Table 1: Datasets used

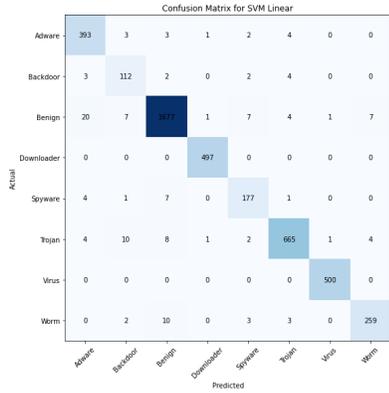
S.No	Types	Test Sample	Train Sample	Total Sample
1	Adware	406	1580	1986
2	Backdoor	123	551	674
3	Downloader	495	2002	2497
4	Spyware	190	756	946
5	Trojan	695	2873	3568
6	Worm	277	1080	1357
7	Virus	500	1892	2392
8	Benign	1724	6910	8634
	Total	4410	17644	22054

The dataset in Table 1 comprises of 22,054 malware samples categorized into 7 types (Adware, Backdoor, Downloader, Spyware, Trojan, Worm, Virus), alongside a Benign class. Training and testing samples are split in an 80:20 ratio per malware type, totaling 17,644 training samples and 4,410 testing samples. The Benign class has the most samples. This distribution ensures a balanced representation, facilitating thorough training and evaluation of machine learning models. The dataset for this research shown in Table 1 consists of malware samples categorized into 7 types: Adware, Backdoor, Downloader, Spyware, Trojan, Worm, and Virus, along with a class for Benign samples. The dataset is divided into training and testing samples, with varying ratios 80:20 for each malware type. The total number of samples in the dataset is 22,054, with 17,644 samples used for training and 4,410 samples used for testing. Each malware type has a different distribution of samples between the training and testing sets, with the Benign class having the highest number of samples. The dataset's distribution suggests a relatively balanced representation of malware types, allowing for comprehensive training and evaluation of machine learning models.



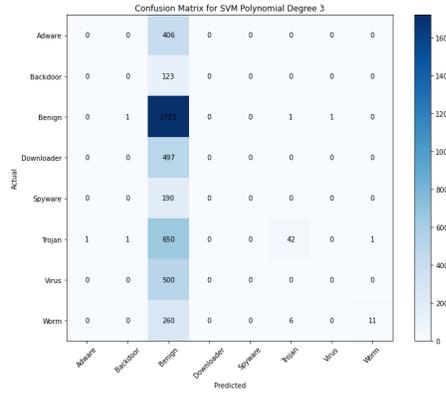


(c) k-Nearest Neighbors

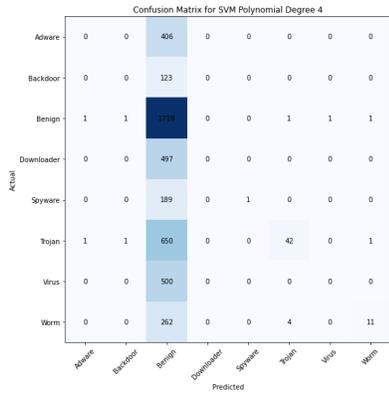


(e) SVM Linear

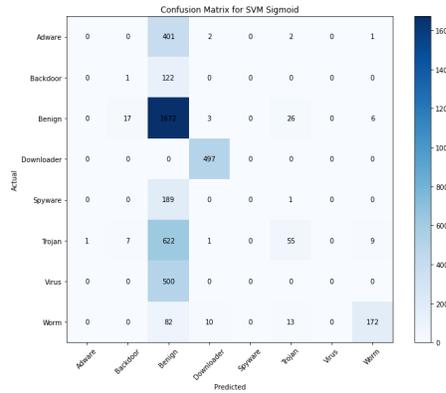
(d) Naive Bayes



(f) SVM Polynomial 3°



(g) SVM Polynomial 4°



(h) SVM Sigmoid

Based on the evaluation metrics in Table 2, it can be observed that the XGBoost classifier stands out as the best performer, achieving the highest accuracy, precision,

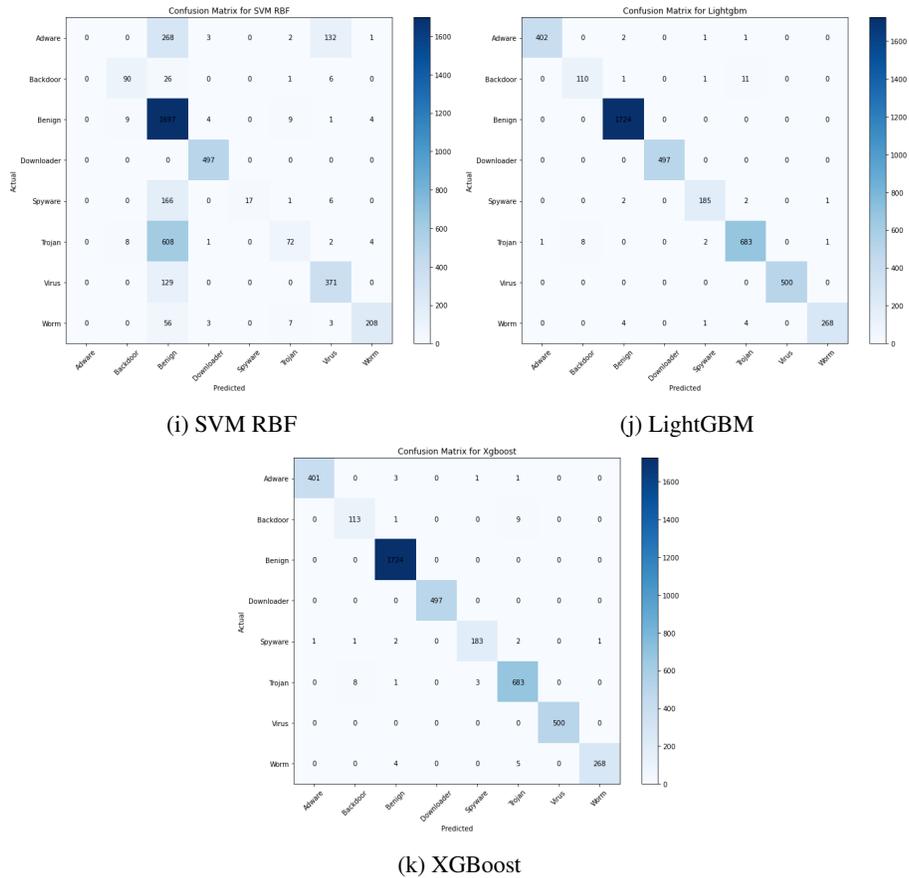


Fig. 3: Confusion Matrices for Malware Classification Using ML Techniques

recall, and F_1 score among all classifiers (99.02%, 98.35%, 97.74%, and 98.04%, respectively). Conversely, SVM Polynomial (degree 3 and 4) and SVM Sigmoid perform poorly, with accuracy rates around 40% and very low precision, recall, and F_1 scores, making them unsuitable for this task.

The poor performance of SVM Polynomial (degree 3 and 4) suggests potential overfitting due to complexity, while SVM Sigmoid struggles with capturing relationships in multi-class problems like malware classification. Naive Bayes exhibits lower performance due to underfitting from its simplicity. In contrast, XGBoost and LightGBM perform best, followed by Random Forest, Decision Tree, and SVM Linear, each demonstrating strengths in handling complex datasets and capturing important patterns. KNN performs decently but lags behind due to limitations in capturing complex data patterns as Confusion Matrices shown in Figure 3.

Table 2: Evaluation Metrics for malware classification using ML Techniques

		Performance Parameters			
S.No	Classifiers	Acc	F_1	Rec	Pre
1	Decision Tree	98.50	97.16	97.30	97.23
2	k-Nearest Neighbors	94.40	93.16	92.16	92.65
3	Naive Bayes	56.26	65.49	69.98	57.70
4	SVM Linear	97.01	94.77	95.93	95.32
5	SVM Polynomial 3 ^o	40.21	27.12	13.73	9.45
6	SVM Polynomial 4 ^o	40.19	39.19	13.78	9.57
7	SVM RBF	66.91	73.10	55.03	55.17
8	SVM Sigmoid	54.33	36.96	33.48	31.32
9	Random Forest	98.37	97.75	96.35	97.03
10	XGBoost	99.02	98.35	97.74	98.04
11	LightGBM	99.02	98.31	97.60	97.95

8 Comparison of our work with present state-of-the-art techniques

Our work focuses on improving malware classification using NLP-based n -gram analysis. Since we utilize a unique dataset, we lack direct comparisons with existing state-of-the-art techniques. Our approach harnesses the power of Natural Language Processing (NLP) and n -gram analysis, offering a distinctive perspective on malware detection. In the absence of any directly related work, we compare our work with some closely related work

Dabas et al. [11] achieved 99.6% accuracy in Windows malware detection using TF-IDF enriched API calls, reducing the feature set to 9%, but they used 2500 malware and 2500 benign samples. Dabas et al. [12] achieved up to 99.7% accuracy with 30% features and 100% accuracy on GA-enriched features, but they used 2500 malware and 2500 benign samples. Sharma et al. [13] achieved up to 99.91% accuracy for SVM and Logistic Regression algorithms by combining API call feature sets into an integrated set and applying TF-IDF, but they didn't disclose the percentage of features used. The detailed comparison of our work with some available related work is given in Table 3.

Table 3: Quantitative comparison of some malware detection techniques

No	Author	n -gram	IF-TDF	Feat. Red ⁿ	ML Tech	Datasets	Class ⁿ	Dect ⁿ	Acc
1	Dabas et al. [11]	✗	✓	9%	✓	VirusShare	✗	✓	99.6
2	Dabas et al. [12]	✗	✓	30%	✓	VirusShare	✗	✓	99.7
3	Sharma et al. [13]	✗	✓	n.d.	✓	VirusShare	✗	✓	99.9
4	Proposed Work	✓	✓	1.6%	✓	VirusShare	✓	✗	99.02

9 Conclusions and Future Work

In conclusion, the evaluation of machine learning algorithms identify XGBoost and LightGBM classifiers as the most proficient options among those considered. Both classifiers demonstrate exceptional performance across key metrics, achieving the highest accuracy, precision, recall and F_1 score values. XGBoost achieves remarkable results with an accuracy of 99.02%, precision of 98.35%, recall of 97.74%, and F_1 score of 98.04%, closely followed by LightGBM with comparable scores of 99.02% accuracy, 98.31% precision, 97.60% recall, and 97.95% F_1 score. These findings show the effectiveness and reliability of both XGBoost and LightGBM for classification tasks, making them the optimal choices for achieving high predictive performance and robustness in similar scenarios.

We outline the following possible extensions to enhance our work: 1) Exploring diverse feature selection approaches to enhance analysis versatility. 2) Applying various deep learning approaches to broaden the research scope and uncover new insights. 3) Utilizing genetic algorithm approaches for feature selection and creation to further expand potential insights from our investigation.

References

1. Robert Moskovitch, Dima Stopel, Clint Feher, Nir Nissim, and Yuval Elovici. Unknown malware detection via text categorization and the imbalance problem. In *2008 IEEE international conference on intelligence and security informatics*, pages 156–161. IEEE, 2008.
2. Jean Bergeron, Mourad Debbabi, Jules Desharnais, Mourad M Erhioui, Yvan Lavoie, and Nadia Tawbi. Static detection of malicious code in executable programs. *Int. J. of Req. Eng.*, 2001(184-189):79, 2001.
3. Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh. A survey on heuristic malware detection techniques. In *The 5th Conference on Information and Knowledge Technology*, pages 113–120. IEEE, 2013.
4. Joma Rajab Salim Alrzini and Diane Pennington. A review of polymorphic malware detection techniques. *International Journal of Advanced Research in Engineering and Technology*, 11(12):1238–1247, 2020.
5. Ce Li, Zijun Cheng, He Zhu, Leiqi Wang, Qiujuan Lv, Yan Wang, Ning Li, and Degang Sun. Dmalnet: Dynamic malware analysis based on api feature engineering and graph learning. *Computers & Security*, 122:102872, 2022.
6. Edward Raff and Charles Nicholas. A survey of machine learning methods and challenges for windows malware classification. *arXiv preprint arXiv:2006.09271*, 2020.
7. Junji Nakazato, Jungsuk Song, Masashi Eto, Daisuke Inoue, and Koji Nakao. A novel malware clustering method using frequency of function call traces in parallel threads. *IEICE transactions on information and systems*, 94(11):2150–2158, 2011.
8. Chandrasekar Ravi and R Manoharan. Malware detection using windows api sequence and machine learning. *International Journal of Computer Applications*, 43(17):12–16, 2012.
9. Weijie Han, Jingfeng Xue, Yong Wang, Lu Huang, Zixiao Kong, and Limin Mao. Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *Computers & Security*, 83:208–233, 2019.
10. Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. Sbmds: an interpretable string based malware detection system using svm ensemble with bagging. *Journal in computer virology*, 5:283–293, 2009.

11. Namita Dabas, Prachi Ahlawat, and Prabha Sharma. An effective malware detection method using hybrid feature selection and machine learning algorithms. *Arabian Journal for Science and Engineering*, 48(8):9749–9767, 2023.
12. Prachi., Namita Dabas, and Prabha Sharma. Malanalyser: An effective and efficient windows malware detection method based on api call sequences. *Expert Systems with Applications*, 230:120756, 2023.
13. Sharma, Prabha, and Nimita. Windows malware detection using machine learning and tf-idf enriched api calls information. In *2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA)*, pages 1–6. IEEE, 2022.
14. Gerard T LaVarnway. An introduction to cart: Classification and regression trees. In *Computer Science and Statistics: Proceedings of the 20th Symposium on the Interface*. Alexandria, VA: American Statistical Association, pages 298–301, 1988.
15. Leo Breiman. Random forest, vol. 45. *Mach Learn*, 1, 2001.