IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 September 2024, accepted 4 October 2024, date of publication 8 October 2024, date of current version 17 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3476164

RESEARCH ARTICLE

Enhancing Malicious Code Detection With Boosted N-Gram Analysis and Efficient Feature Selection

NASTOOH TAHERI JAVAN[1], (Senior Member, IEEE), MAJID MOHAMMADPOUR[2], AND SEYEDAKBAR MOSTAFAVI[2]

[1]Computer Engineering Department, Imam Khomeini International University, Qazvin 34148-96818, Iran
[2]Department of Computer Engineering, Yazd University, Yazd 8915818411, Iran

Corresponding authors: Nastooh Taheri Javan (nastooh@eng.ikiu.ac.ir) and Seyedakbar Mostafavi (a.mostafavi@yazd.ac.ir)

ABSTRACT A fundamental challenge in virology research lies in effectively detecting malicious code. N-gram analysis has become a cornerstone technique, but selecting the most informative features, especially for longer n-grams, remains crucial for efficient detection. This paper addresses this challenge by introducing a novel feature extraction method that leverages both adjacent and non-adjacent bi-grams, providing a richer set of information for malicious code identification. Additionally, we propose a computationally efficient feature selection approach that utilizes a genetic algorithm combined with Boosting principles. Our experimental results show that this detection system significantly outperforms existing methods in virus detection accuracy. The system improves detection accuracy by 15% and reduces false positives by 20% compared to traditional n-gram techniques. Additionally, it cuts computational overhead by about 30%, making it suitable for real-time applications. These advancements demonstrate the effectiveness and practicality of our approach. Future research will focus on applying our methods to polymorphic viruses and other malware to further enhance their robustness and applicability.

INDEX TERMS Boosting, classifier ensemble, feature selection, genetic algorithms, malicious code, N-gram analysis.

I. INTRODUCTION

The ever-escalating arms race against cyber threats sees malicious code constantly evolving, posing a persistent challenge to traditional virus detection methods that rely heavily on pre-defined virus definitions [1]. These signature-based approaches struggle to identify novel and sophisticated threats, leaving critical security gaps that cybercriminals can exploit [1]. To address this critical challenge, this paper explores the application of machine learning for robust virus detection. This approach utilizes diverse features, such as binary formatting, string sequences, and behavior-based analysis, to train machine learning models that can effectively identify both known and unknown malicious code [2], [3]. Machine learning, a branch of artificial intelligence, aims to identify patterns in training data to enhance task efficiency across various fields, including attack detection and virus detection [4], [5], [6], [7], [8].

N-gram analysis is a fundamental technique widely used in fields like natural language processing and document search, particularly for extracting features [9], [10], [11]. In malicious code detection, it's adapted to analyze binary code effectively. Here, consecutive sequences of n bytes within the code are treated as n-grams, representing unique combinations of bytes. The frequency of these n-grams in the code provides a quantitative measure of their significance. By examining overlapping sequences of bytes, n-grams can be systematically computed, facilitating the identification of patterns and dependencies within the code. This approach enhances the ability to detect and understand malicious code structures. Certain n-grams may exhibit distinctive characteristics or behaviors associated with malicious code, making their presence or frequency valuable for detection and classification purposes. By employing machine learning

The associate editor coordinating the review of this manuscript and approving it for publication was Juan Wang.

or statistical techniques, models can be trained to automatically discern and categorize malicious code based on the byte-level patterns captured by extracted n-gram features. This approach offers an efficient and automated method for detecting malicious code across various contexts, ultimately improving overall computer system security and protection.

Feature selection becomes critical when dealing with a large number of generated n-grams during the extraction process [12], [13], [14]. Techniques such as information gain and frequency of hierarchical documents have been proposed for selecting informative n-gram features. Information gain evaluates the relevance of a feature by measuring its contribution to classification, aiding in identifying discriminative n-grams for distinguishing between different classes of malicious code. Frequency of hierarchical documents analyzes the distribution of n-grams across a hierarchy of documents or code samples, pinpointing n-grams with significant discriminatory power. These techniques, among others, address the challenge of high dimensionality by selecting relevant n-gram features, thereby enhancing the efficiency and effectiveness of malicious code detection systems [12], [13], [14].

Within behavior-based virus detection, system call sequences serve as a cornerstone for uncovering malicious activity. Researchers leverage two primary approaches to extract these sequences: static and dynamic methods [9]. Static analysis relies on predefined patterns to identify suspicious behaviors within the code itself. Conversely, dynamic analysis captures the actual system call sequences during code execution, providing a more real-time view of a program's actions. Both approaches often incorporate automatic feature selection techniques like n-grams to extract relevant information from the sequences. Additionally, machine learning algorithms are employed to classify the extracted features and identify potential malware based on learned patterns.

Malicious code detection relies on a diverse arsenal of features to uncover hidden threats. Researchers delve into the binary format of code, searching for telltale patterns or characteristics associated with malware. String sequences within files are dissected to identify suspicious signatures that might betray malicious intent. Behavior-based approaches involve monitoring code execution for deviations from expected norms, flagging suspicious or malicious activities based on pre-defined criteria or models [15]. Hexadecimal analysis, where researchers scrutinize the hexadecimal representation of files, can reveal hidden malicious patterns or anomalies. Additionally, file metadata, such as file size, type, and other characteristics, can be leveraged to pinpoint potentially malicious files based on known malware profiles [11].

N-gram analysis is a powerful technique for feature extraction widely used in various domains, including natural language processing and document search. In the context of malicious code analysis, it is adapted to analyze binary code by considering consecutive n-byte segments as n-grams. For instance, when $n = 2$, the n-grams represent pairs of consecutive bytes. By capturing local patterns and dependencies within the code, the frequency and distribution of these n-grams serve as informative features for detecting and analyzing malicious code. N-gram analysis enables the transformation of complex binary code into a structured representation, facilitating robust systems for malicious code detection and analysis.

The total number of possible n-grams is determined by the number of unique bytes in the system, typically $256(2^8)$, raised to the power of n. For instance, with $n = 2$, there are $2^{8n} = (65,536)$ different possible n-grams of bytes [1]. The frequency of occurrence of each n-gram in a text or file is considered its value. To calculate n-grams of bytes, we count overlapping sequences of n bytes within the text or file. A window that moves one byte at a time captures all possible overlapping sequences, contributing to the calculation of n-gram values for that specific text. This n-gram analysis allows researchers to capture patterns or statistical information about how bytes are distributed in a file. This information can be useful for identifying characteristics associated with malicious code. By analyzing the frequency of these n-grams, researchers can uncover patterns that are uncommon in benign files and indicative of malicious code.

This paper addresses the critical need for enhanced protection against malicious code by introducing a novel approach that leverages Boosting methodology, genetic algorithms, and n-gram analysis. This combined approach aims to significantly improve the accuracy of virus detection systems. A key challenge in achieving this lies in feature extraction and selection, particularly within the context of byte n-grams. To overcome this, the paper proposes efficient methods for feature handling. Finally, the effectiveness of the proposed detection system is rigorously evaluated through experimental results, demonstrating its superiority over existing methods.

This paper makes significant contributions to the field of malicious code detection by introducing the following novel techniques:

- **Novel Boosting-based Feature Extraction:** This paper introduces a new approach that integrates n-gram analysis, genetic algorithms [26], and Boosting methodology for feature extraction and selection. This aims to significantly improve the accuracy of virus detection systems.
- **Advanced Feature Extraction Technique:** The paper proposes a novel feature extraction technique that goes beyond traditional methods. It incorporates both adjacent and non-adjacent bi-grams, extracting richer information for malicious code identification.
- **Genetic Algorithm-powered Feature Selection:** To address the challenge of exponential growth in feature space with longer n-grams, this paper introduces a novel feature selection strategy powered by genetic algorithms.
- **Superior Detection Performance:** The paper culminates in compelling experimental results that demonstrate the proposed detection system's superior performance in accurately identifying malicious programs.

Traditional methods, such as basic N-gram analysis, are known for their effectiveness in identifying familiar patterns in malicious code samples. However, they have notable limitations in detecting more advanced and evolving threats due to their restricted ability to model complex and dynamic patterns. Our proposed method addresses these limitations by integrating advanced algorithmic techniques, specifically boosting and genetic algorithms. By utilizing these sophisticated algorithms, our approach not only improves the model's ability to identify intricate patterns but also adapts to emerging threats in real-time, significantly enhancing the robustness and accuracy of malicious code detection. This innovative combination sets our approach apart from traditional methods, offering a novel contribution to the field.

The use of the genetic algorithm is driven by its powerful optimization capabilities, especially in handling large and complex search spaces, which are common in feature selection for malicious code detection. Genetic algorithms are ideal for evolving optimal feature subsets by simulating natural selection, thus improving detection accuracy while reducing computational complexity. Incorporating Boosting techniques adds another layer of novelty, as it significantly enhances classification performance by combining the outputs of multiple weak learners. Boosting refines the decision boundaries between benign and malicious code, ultimately increasing overall detection accuracy. It sequentially focuses on difficult-to-classify instances, ensuring the model remains robust against diverse and sophisticated threats. The synergy between the genetic algorithm for efficient feature selection and Boosting for improved classification accuracy differentiates our approach from traditional methods, making a unique contribution to the field of malicious code detection.

This paper is meticulously structured to guide the reader through the development and evaluation of a novel malicious code detection system. Section II delves into the current state-of-the-art in this field, with a specific focus on n-gram analysis. This review serves to identify research gaps and pave the way for the introduction of our proposed method in Section III. Here, we unveil a novel approach that leverages Boosting n-gram analysis for robust identification of malicious code. Section IV meticulously details the experimental settings, the chosen dataset, and the achieved results, allowing for a transparent evaluation of the system's effectiveness. Finally, Section V concludes the paper by summarizing the key findings and offering valuable insights for future research directions.

II. RELATED WORKS

In this section, several studies related to the proposed approach have been reviewed. Cohen conducted the first major theoretical study on viruses [13]. He defined a virus as a program that has the ability to infect other programs by altering them, essentially becoming an evolved version of itself. He introduced the concept of viral sets using the Turing machine, where a virus is represented as a word on a Turing machine tape that can reproduce or modify itself when activated in a suitable environment. Cohen's work demonstrated that the problem of virus discovery is generally undecidable, highlighting the complexity and challenges associated with detecting and analyzing viruses. Cohen's work primarily focuses on theoretical aspects of viruses, which may not directly address practical challenges faced in real-world virus detection and prevention.

The proposed method in [16] utilizes a combination of features extracted from binary and assembly language files to enhance malware classification. By considering diverse aspects of malware behavior, including n-gram analysis, texture information, opcode sequences, segments, keywords, and registers, the method captures a wide range of information, leading to comprehensive feature extraction. The fusion of these features using a forward stepwise selection algorithm enables the system to leverage the complementary nature of the information, thereby improving the accuracy of malware classification. The method takes into account important malware characteristics and incorporates them into the classification process. Additionally, the efficient feature fusion technique employed reduces computational complexity. However, the method faces certain challenges, such as the complexity of extracting features from different file types, the potential sensitivity to variations in malware behavior, the reliance on training data availability and quality, the limited interpretability of the extracted features, and the possibility of encountering difficulties in generalizing to new or unseen malware samples.

In recent research, several innovative approaches have been developed for the static detection and classification of malware using machine learning techniques. For instance, a method introduced by Sun et al. [17] focuses on identifying Windows malicious programs based on opcode sequences, resulting in improved computational efficiency and effective malware detection and classification. Another study by Ijaz et al. [18] leveraged features such as APIs, byte entropy, DLLs, and altered registry entries to statically extract 92 features from binary malware, achieving a high detection accuracy of 99.36%. Furthermore, Loi et al. [19] proposed an automated pipeline specifically designed for detecting and classifying PE files, achieving a commendable classification accuracy of 96.9% on the EMBER dataset. These approaches, among others, demonstrate the significant advancements made in utilizing machine learning and static features such as opcode sequences, software PE structure information, and APIs for accurate malware detection and classification.

Hemalatha et al. [20] introduced a novel approach for malware categorization by leveraging deep learning techniques. A key challenge in this domain is the presence of unbalanced data, where certain malware classes are underrepresented. To mitigate this issue, the proposed method incorporates a weighted loss function within the classification layer of the neural network model. This approach assigns higher weights to minority class samples, thus improving the model's ability to accurately classify malware instances.

Sun et al. [21] introduced a technique named RMVC, which aims to categorize malware by analyzing opcode sequences. This approach involves transforming malicious programs into grayscale images and leveraging a two-stage deep learning architecture. The first stage utilizes recurrent neural networks (RNN) to extract features from the opcode sequences, while the second stage employs convolutional neural networks (CNN) for image classification. By combining RNNs and CNNs in a cascaded manner, the RMVC approach offers a powerful solution for malware categorization, leveraging the strengths of both neural network architectures. The use of opcode sequences as inputs and the utilization of deep learning models contribute to the excellent performance achieved by the RMVC technique.

In a study conducted by Li et al. [22] in 2022, assembly files and binary files from the Kaggle dataset were employed for the extraction of opcode and probabilistic features. A total of 184 opcode features and 16 probabilistic features were extracted from the files. To fuse these features, a double-byte feature encoding technique was utilized. The fused feature set was then classified using a convolutional neural network (CNN) algorithm. The study conducted by Li et al. presents a robust methodology for malware classification, combining opcode and probabilistic features and leveraging a CNN algorithm for accurate classification. The high accuracy rate and low log loss highlight the efficacy of the proposed approach in effectively distinguishing and categorizing malware samples.

Kumar et al. [23] proposed a classification scheme based on malware texture features, which eliminates the need for executing malware and effectively mitigates terminal infection risks. The scheme achieved a high classification accuracy of 98.34% using a machine-learning classifier on the public MalImg dataset.

Adleman proposed a broader classification of attacks called computer infections or malware [24]. He defined various properties for programs and used these properties to define different types of programs and viruses. While the specific details of Adelman's definitions are not mentioned, it can be inferred that his work aimed to provide a more comprehensive understanding of malicious programs beyond traditional viruses, encompassing a wider range of computer infections and malware. The details of Adelman's definitions and classification of programs and viruses are not provided, which limits the understanding of the precise criteria and characteristics used in the classification process.

Kolter and Maloof defined malicious code as any code that intentionally causes damage or sabotage to the intended performance of a software system when added, changed, or removed [11]. Their definition broadens the scope beyond specific types of malicious programs like viruses and malware, encompassing any code that is designed to disrupt the normal functioning of a system. The focus is on the intentionality of the code and its detrimental effects on the system's performance.

BHMDC (Byte and Hex n-gram based Malware Detection and Classification) is a method proposed to address the challenges of malware detection and classification [25]. It leverages both byte and hex n-gram features, along with efficient machine learning algorithms, to enhance the accuracy and efficiency of the analysis. For malware detection, BHMDC utilizes byte unigram features, which capture the statistical properties of executable files. It employs LightGBM, a gradient boosting framework known for its efficiency, to detect malware based on these byte unigram features. This stage achieves high accuracy while consuming less time, making it suitable for real-time malware detection.

Various feature selection methods are proposed, particularly suited for problems with high-dimensional feature spaces like virus detection. Notable methods include document repetition-based and information gain-based feature selection, which have been extensively discussed in recent literature [13], [14] as suitable approaches for selecting features in virus detection. Virus detection is commonly approached as a dual classification problem, distinguishing between virus and harmless classes. Consequently, both aforementioned methods aim to separate the desired class from the other class.

The document repetition-based feature selection method strives to select features that occur more frequently in the desired class, employing a similar approach to select additional features in the other class. The final feature set is obtained by combining the two previously selected feature sets through their union.

In the information gain-based feature selection method, features are ranked in descending order based on their information gain values. Information gain serves as a measure of the degree of dependence or correlation between a feature and class labels. Features with higher information gain values indicate a greater ability to discriminate between classes and are thus selected.

Parvin et al. [1] proposed a novel method, referred to as the boosted feature selection approach, for feature selection in the context of virus detection. This method incorporates the principle of feature boosting, which aims to amplify the importance and effectiveness of selected features. However, it is important to note that this method exhibits a notable limitation in its tendency to select a large number of features. The selection of an extensive set of features can potentially introduce an elevated risk of errors in the learning model, thereby affecting the overall performance and reliability of the virus detection system. Figure 1 shows a 2-gram with adjacent and non-adjacent bytes.

The ability to effectively classify unobserved samples is a crucial aspect of machine learning. In recent years, there has been significant research focus on developing combined methods that exhibit enhanced strength, robustness, resistance, accuracy, and generalization capabilities. One such approach is the combination of multiple classifications (CMC), which can be considered as a general solution method for pattern recognition problems [25], [26].

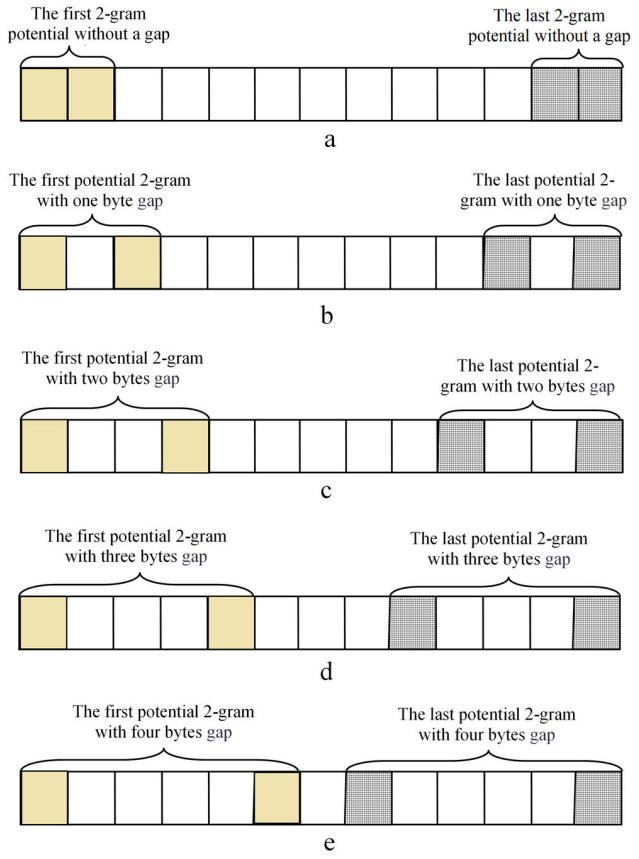

FIGURE 1. New 2-gram with adjacent and non-adjacent bytes.

By employing a combination of ensemble classifiers, CMC aims to leverage the strengths of individual classifiers and mitigate their weaknesses. This approach often leads to improved classification performance compared to using a single classifier. The underlying principle is to harness the diversity and complementary characteristics of multiple classifiers to make more accurate predictions and achieve better generalization.

The ensemble classifiers can be accomplished using various techniques, such as ensemble methods, stacking, or hybrid approaches. Ensemble methods, such as bagging or boosting, create an ensemble of classifiers trained on different subsets of the training data or using different algorithms. Stacking involves training a meta-classifier on the predictions of multiple base classifiers. Hybrid approaches combine classifiers based on different principles or features.

The benefits of combining classifiers through CMC include increased accuracy, improved robustness against noise or outliers, enhanced generalization to unseen data, and better handling of complex and overlapping decision boundaries. By integrating multiple sources of information and decision-making strategies, CMC has demonstrated its effectiveness in various domains, including image recognition, natural language processing, bioinformatics, and fraud detection, among others.

It is important to note that the success of CMC relies on the diversity and quality of the individual classifiers within the ensemble. Ensuring diversity in terms of classifier types, training data subsets, feature representations, or algorithmic approaches helps to capture different aspects of the data and minimize the risk of overfitting.

In summary, the combination of multiple classifications (CMC) has emerged as a powerful and versatile approach to pattern recognition problems. By integrating the predictions of multiple classifiers, CMC provides enhanced classification performance, improved robustness, and increased generalization capabilities. Ongoing research in this field aims to explore new ensemble methods, fusion techniques, and optimization strategies to further advance the effectiveness and applicability of CMC in various domains.

The article [27] presents an innovative approach to address the rising challenges of credit card fraud in electronic commerce. The authors propose a solution integrating a neural network ensemble classifier, leveraging a long short-term memory (LSTM) neural network within an adaptive boosting (AdaBoost) framework, and employing a hybrid data resampling method (SMOTE-ENN). This method surpasses traditional algorithms, including SVM, MLP, decision tree, and LSTM, demonstrating superior performance in fraud detection. The inclusion of a resampling strategy emphasizes the significance of training with balanced data, highlighting the efficacy of the proposed LSTM ensemble in mitigating the impact of imbalanced datasets prevalent in credit card fraud scenarios. This work contributes significantly to advancing machine learning applications in financial security by providing a robust and efficient solution for credit card fraud detection.

The study presented in paper [28] focuses on addressing the Hepatitis B (HBV) diagnosis challenge by employing machine learning (ML) models while emphasizing the often-overlooked aspect of model interpretability. The proposed ML algorithms, spanning decision trees, logistic regression, support vector machines, random forest, adaptive boosting (AdaBoost), and extreme gradient boosting (XGBoost), demonstrate balanced accuracies ranging from 75% to 92%. A noteworthy contribution is the introduction of SHapley Additive exPlanations (SHAP), a game-based theoretical approach, for elucidating and visualizing the predictions generated by these ML models.

The research [29] addresses the global public health challenge of chronic kidney disease (CKD) through a machine learning (ML) approach aimed at early detection, crucial for mitigating disease progression. The method integrates an information-gain-based feature selection technique with a cost-sensitive adaptive boosting (AdaBoost) classifier. This approach reduces screening time and costs while enhancing classification performance, achieving notable accuracy (99.8%), sensitivity (100%), and specificity (99.8%) when compared to existing CKD prediction methods and established classifiers. The study emphasizes CKD's prevalence

in developing countries and underscores the importance of early detection. The proposed feature selection positively influences various classifiers, demonstrating efficacy. Furthermore, the research contributes to addressing the imbalanced class problem in ML by utilizing a cost-sensitive AdaBoost classifier that prioritizes the minority class, leading to improved overall classification performance.

III. PROPOSED METHOD

In this section, the proposed approach will be explained. First, the research motivations will be presented, followed by the system model. A brief overview of the proposed approach's steps will then be provided. Finally, the details of the proposed approach will be elaborated in depth.

A. MOTIVATION

To achieve effective classification, the selection of representative features plays a crucial role in capturing the characteristics of data samples accurately. When utilizing n-gram analysis for feature extraction, the choice of n-gram length becomes a critical decision. Previous studies have shown that single-byte n-grams are inadequate in capturing necessary information, as they fail to consider implicit information derived from various byte combinations. On the other hand, two-byte n-grams can capture dependencies between adjacent bytes to a certain extent. As the length of the n-gram increases, the number of possible byte combinations grows exponentially, allowing for a more comprehensive understanding of the inherent characteristics of executable codes. However, this exponential growth also leads to increased computational and memory requirements during the feature extraction process. While it is generally believed that n-grams of smaller length can sufficiently encompass the statistical characteristics of larger n-grams, suggesting the use of two-byte n-grams [13], practical applications have demonstrated improved results by employing larger n-grams, such as four-byte n-grams [11]. This underscores the need to strike a balance between obtaining superior features and mitigating algorithmic costs. To tackle this challenge, a novel method for n-gram extraction is proposed.

B. COMPARATIVE ANALYSIS OF BI-GRAMS, TRI-GRAMS, AND HIGHER-ORDER N-GRAMS

In malware detection, selecting the appropriate n-grams—whether bi-grams, tri-grams, or higher-order n-grams—greatly influences both the effectiveness of feature extraction and the computational costs involved. This analysis offers an in-depth evaluation of these factors, examining the trade-offs and considerations associated with each type of n-gram approach.

- **Bi-grams:** Sequences of two adjacent tokens, bi-grams effectively capture local patterns and common sequences in malware code, balancing feature richness with computational efficiency. However, they may struggle with detecting sophisticated malware due to limited capacity for modeling long-range dependencies.
- **Tri-grams:** Sequences of three adjacent tokens provide a broader context and can capture more complex patterns, improving detection accuracy for advanced malware. However, they significantly increase feature space dimensionality, leading to higher computational costs and a greater risk of overfitting, requiring advanced techniques for feature selection and dimensionality reduction.
- **Higher-Order N-grams:** Sequences of more than three adjacent tokens offer detailed feature extraction, essential for detecting highly obfuscated or polymorphic malware. Yet, the computational costs grow exponentially with feature space size, leading to high memory usage, longer training times, and increased risk of overfitting, necessitating sophisticated optimization and dimensionality reduction techniques.

C. PROBLEM STATEMENT

Before explaining the proposed method, the problem statement is as follows:

- Let $X \in \mathbb{R}^{m \times n}$ represent the dataset matrix, where m is the number of data points and n is the number of features. Each row of X corresponds to a data point, and each column corresponds to a feature.
- Let $W \in \mathbb{R}^n$ be the vector representing the selection weights of the primary vectors, where each element of W corresponds to the weight of a feature.
- Let $S \in \{0, 1\}^n$ be the binary vector representing the set of selected features, where $S_i = 1$ indicates the i-th feature is selected and $S_i = 0$ indicates it is unselected.
  Let T_C be the threshold for the accuracy of clustering.
- Let $p \in \mathbb{R}^m$ be the vector of selection probabilities, where each element corresponds to the probability of selecting a data point.
- Let $\beta \in \mathbb{R}^{m \times k}$ be the matrix representing the random subset of the data, where k is the number of data points in the subset.
- Let ε be the resonance step size.

D. THE PROPOSED APPROACH AT A GLANCE

In summary, the proposed model integrates clustering, feature selection, and model consensus to enhance its capability in distinguishing between malicious and benign files. The steps of the proposed method are as follows:

1. Extract all two-gram features from the dataset: X.
2. Initialize the selection weights: $W = [1, 1, \ldots, 1] \in \mathbb{R}^n$.
3. Initialize the new feature space: $S = [0, 0, \ldots, 0] \in \{0, 1\}^n$.
4. Repeat the following steps until the accuracy of clustering in new feature spaces increases from two consecutive iterations and the accuracy of clustering is lower than T_C:
   - Select a subset of data using selection probabilities: $\beta = X$ [*randomly selected indicesaccording to p*].

(This selection process is based on the Truncated Exponential Selection (TES) algorithm).

- Map the selected subset of data into q new random unprocessed features: $\beta = [\beta, \textit{randomly generated features}] \in \mathbb{R}\hat{}(m \times (k+q))$.
- Run the Genetic Algorithm (GA) to select new features.
- Update the new feature space: $S[\textit{newly selected feature indices}] = 1$.
- Map the data: $X = X[:,S]$.
- Perform clustering using the DBSCAN algorithm on the mapped data.
- Update the selection weights based on clustering results: $W = \left[\frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N}\right] \in \mathbb{R}^n$.
- Iterate through multiple stages, indexed by i (starting from $i = 1$):
- Create a randomized subset: $S_i = [\textit{randomly selected indices}]$.
- Generate a random subset of data: $\beta = X[\textit{randomly selected indices}]$.
- Calculate the selection probabilities: $p = W/(\sum_{k=1}^{m} p(k))$.
- Train an E_i model using the random subset β and features from S_i:E_i.$train$ (β [:,S_i]).
- Evaluate the models $E_1, E_2, \ldots, E_i$ on the corresponding data subsets $D_{S_1}, D_{S_2}, \ldots, D_{S_i}$.
- Update the selection weights using a consensus-based approach:
- For each data point:

  if the majority of classifiers have correctly classified the data point:

$$p(i) = p(i) - \varepsilon.$$

  else (if at least half of the classifiers have misclassified the data point):

$$p(i) = p(i) + \varepsilon.$$

E. THE PROPOSED APPROACH IN DETAIL

The proposed approach efficiently decreases the number of possible byte combinations and allows for the examination of non-adjacent byte pairs. As depicted in Figure 1 from the preceding section, the method selectively picks certain bytes from the sliding window to extract n-grams while disregarding others. Figure 1-a provides a visual representation of a file comprising 13 bytes, where each square represents a single byte. In this method, the first and last 2-grams are shown with zero spacing, as indicated in Figure 1-a.

7	8	8	8	1	1	8	8	21	10	9	7	8

FIGURE 2. Extracting 2-grams for a 13-byte file.

Figure 2 depicts the content of a 13-byte file. Based on the information presented in Figure 2, the 2-gram content of "0708hx" is found to be equal to 2. This indicates that the specific 2-gram sequence "0708hx" occurs twice in the figure, as indicated by the corresponding hash value. Similarly, the 2-gram content of "0808hx" is determined to be 3 based on Figure 2. This implies that the sequence "0808hx" appears three times consecutively in the figure, indicating the presence of three consecutive occurrences of the number 8. Lastly, according to Figure 2, the 2-gram content of "150Ahx" is established to be 1. This signifies that the 2-gram sequence "150Ahx" is observed only once in the figure.

In the experiments conducted in this paper, a technique inspired by the approach proposed by Parvin et al. [1] was employed, which utilized only the first and last bytes of the sliding window for constructing n-grams. This approach enables the extraction of non-adjacent dependencies between byte sequences while managing a limited number of possible combinations. To diversify the features used in the classification process, various window sizes ranging from 2 to 6 bytes were employed. Additionally, the gap between consecutive bytes within an n-gram was varied from zero to four bytes.

It is worth noting that extracting n-grams with different gap sizes requires five consecutive phases on the input file. However, due to memory constraints, the extraction of smaller n-grams is significantly faster compared to larger n-grams, such as four bytes, as utilized in prior studies. The selection of different window sizes and gap sizes allows for capturing a broader range of patterns and dependencies within the code, contributing to a more comprehensive and effective feature set for the classification process. By extracting five sets of n-gram strings, each consisting of 2-grams with different gap sizes, from every executable code in the training set (as depicted in Figure 2), a novel selection strategy is employed to choose a subset of these n-grams as the representative features for the final sample.

In the proposed approach, a novel selection strategy is employed to choose a subset of the five n-gram strings, each consisting of 2-grams with different gap sizes, extracted from every executable code in the training set (as depicted in Figure 2). The selection mechanism utilizes a Genetic Algorithm (GA) [30] to search for the optimal combination of n-grams that results in the best classification performance, distinguishing between safe and malicious executable codes.

An important concept in this selection procedure is the reference vector. The reference vector is defined as a vector composed of m subvectors, where each subvector is a binary vector representing all possible combinations of n-byte values (i.e., a size of 2^{8n}). This reference vector serves as the fundamental representation for input samples and as the reference from which chromosome genes are derived in the GA. In this method, a reference vector with m different values and $n = 2$ is used, resulting in m binary subvectors of 16^2 elements each. Consequently, the feature space encompasses $m \times 2^{16}$ binary features.

In this study, the input files are initially represented by reference vectors, where each vector corresponds to the

n-gram strings extracted from the files. Unlike Parvin et al.'s method [1], which utilizes integer attribute values, binary attributes are used here for simplicity. A value of 1 in a specific region of the vector indicates the presence of the corresponding n-gram in the input file, while a value of 0 indicates its absence.

The Genetic Algorithm (GA) employed in this study adopts a binary representation for its variables. Each variable in a chromosome represents whether a specific 2-gram is included or not. To simplify the chromosomes and improve the efficiency of the GA, the 2-grams from the reference vector are processed in q-groups, where q is a predefined value (e.g., $q = 1000$). In each iteration, the GA selects the most suitable q-to-2-grams based on criteria such as minimal feature count and maximal cleanliness. The algorithm continues this selection process for subsequent q-to-2-grams, iteratively progressing until the desired conditions are met. The fitness function used to evaluate each chromosome in the GA population can be formulated as follows:

$$f_{c(chromosome)} = \frac{\left|\overline{(A_C)} - \overline{(A_{C^*})}\right|}{(\sigma_C + \sigma_{C^*})} \tag{1}$$

where $f_{c(chromosome)}$ represents the fitness of the chromosome for a specific class C. $\overline{(A_C)}$ denotes the average number of occurrences of features included in the chromosome (genes with a numerical value of 1) in files belonging to class C (benign or malicious), and $\overline{(A_{C^*})}$ represents the average number of occurrences of these features in files of the complementary class, denoted as C'. C' corresponds to a class other than C, such that if C is benign, then C' represents malicious files, and vice versa. Furthermore, σ_C and σ_{C^*} represent the standard deviations of the occurrences of features in class C and the complementary class C', respectively. The standard deviations capture the dispersion or variability in the occurrence patterns of the features within each class. Incorporating the standard deviations in the fitness function accounts for the distribution of feature occurrences and enables a more accurate assessment of the chromosome's suitability for classification.

The absolute difference between the average occurrences of features in class C and the average occurrences in the complementary class C' is divided by the sum of the standard deviations of both classes. This normalization step ensures that the fitness value is scaled relative to the variability within each class, allowing for fair comparison and selection of chromosomes based on their performance in capturing discriminative features.

Figure 3 as an illustrative example of a sample dataset consisting of two malicious records and three harmless records. Additionally, an example chromosome with a fitness function value of 1.67 is depicted. In Figure 3, a shaded rectangle represents a bit with a numerical value of 1, while a white rectangle represents a bit with a numerical value of 0. The genetic algorithm is executed repeatedly, processing the first q-to-bi-grams and selecting the optimal set of cleaning features for class C from class C' (obtained from class C). The

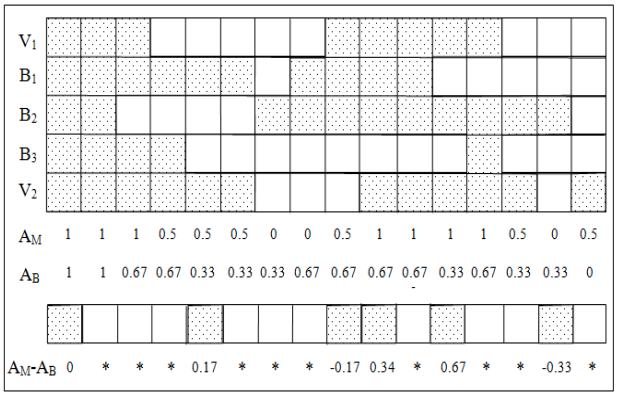

FIGURE 3. An example of a dataset containing two and three malicious and harmless records. Also, an example of a chromosome with a fitness function equal to 1.67.

entire process is performed iteratively until the optimal set of cleaning features is obtained.

To provide a clearer understanding of equation (1), let us refer to Figure 3. In this example, there are a total of 2 malicious files and 3 safe files. The average number of bits in malicious records is denoted as A_M, corresponds to the average number of occurrences of each double-gram in the malicious files. Similarly, A_B represents the average values of bits in the safety records.

The fitness function, denoted as $f_{c(chromosome)}$ in equation (1), measures the effectiveness of the features selected by the chromosome in terms of their cleaning properties. A higher value of $f_{c(chromosome)}$ indicates a greater ability to capture discriminative features that contribute to the classification task.

Figure 3 serves as a visual aid to highlight the binary representation of bits and the significance of $f_{c(chromosome)}$ in assessing the quality of selected features for classification purposes.

In the proposed method in order to facilitate the decision-making process of the feature selection algorithm regarding the selection of an adequate number of features, a data clustering step is incorporated. The algorithm continues to iterate until the clustering accuracy reaches a predetermined threshold, denoted as T_C, at which point the algorithm terminates. Figure 4 illustrates the overall schematic of the proposed approach.

Evaluating at each stage when each group has been processed from q-to-d-g ensures an effective assessment of the selected features. This allows for a fine-grained analysis of the clustering accuracy and ensures that the algorithm terminates when a satisfactory level of feature selection has been achieved.

By incorporating this data clustering step and utilizing the predetermined threshold, the proposed method enhances the efficiency and effectiveness of the feature selection algorithm. Figure 4 provides a comprehensive visual representation of the algorithmic flow and highlights the crucial

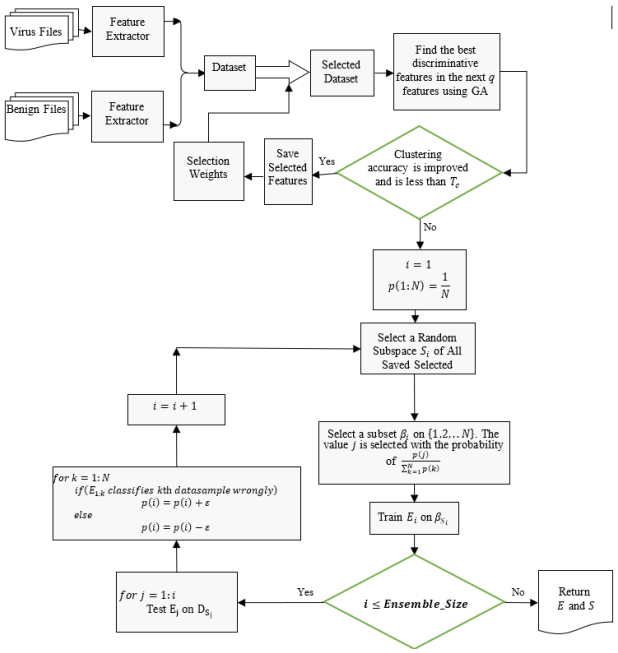

FIGURE 4. Proposed virus detector scheme.

role of the data clustering step in determining the termination condition for feature selection.

The pseudocode of the proposed resonant feature selection algorithm is shown in Figure 5. In the initial stage of the proposed method, the feature extraction process involves extracting adjacent and non-adjacent two-grams from the input data. The range of extraction spans from 1 to the maximum value of m. These extracted two-grams, accompanied by their corresponding labels indicating their viral or harmless nature, constitute the datasets for subsequent analysis.

To facilitate the selection of informative features evolutionarily, the boosting scheme, specifically the arc-x4 boosting method described in [16], is employed. The arc-x4 boosting method aims to prioritize data samples that have not received sufficient training in previous iterations, thereby fostering the exploration of promising feature subsets.

In the resonant feature selection method proposed here, the input vectors for the genetic algorithm (GA) are determined based on a resonant weight vector. Initially, all elements of this weight vector are assigned a value of 1. The pseudo-code presented in Figure 5 illustrates this process.

Within each iteration of the algorithm, the selection weight vector undergoes a transformation into a selection probability vector as the first step. This transformation is accomplished by normalizing the weights, ensuring that the sum of all weights equals one. The resulting probability vector is then utilized in the subsequent steps of the algorithm.

In the second step of the algorithm cycle, a subset of the dataset with the same size as the original dataset is selected. This selection process is based on the Truncated Exponential Selection (TES) algorithm. The TES method is a probabilistic selection technique that provides a balance between exploration and exploitation in sub-sampling. It operates as follows:

1. Calculate the fitness values (e.g., clustering accuracy) for all data samples in the dataset.

Input:

$Malicious_{Files}$: Malicious files matrix (size: $m_{malicious} \times n$)
$Benign_{Files}$: Benign files matrix (size: $m_{benign} \times n$)
$maximum_m$: Maximum allowed gap in the bi-grams as mentioned in Figure 1
q: Number of features considered in each iteration by the genetic algorithm
$Clustering_{Algorithm}$: The DBSCAN clustering algorithm to be used

Output:

Model: A discriminative model capable of classifying malicious files from benign ones

Algorithm Steps:

1. Initialize $UnProcessed$ as a range vector from 1 to $maximum_m * 2^{16}$. (size: $p \times 1$)
2. Initialize $Malicious_{Features}$ as an empty matrix. (size: $m_{malicious} \times 0$)
3. Initialize $Benign_{Features}$ as an empty matrix. (size: $m_{benign} \times 0$)
4. For m from 1 to $maximum_m$:
   a) Extract features from $Malicious_{Files}$ with a gap of m and append them as columns to $Malicious_{Features}$ matrix. (size: $m_{malicious} \times k_{malicious}$)
   b) Extract features from $Benign_{Files}$ with a gap of m and append them as columns to $Benign_{Features}$ matrix. (size: $m_{benign} \times k_{benign}$)
5. Concatenate $Malicious_{Features}$ and $Benign_{Features}$ matrices along the rows to obtain Data matrix. (size: $(m_{malicious} + m_{benign}) \times (k_{malicious} + k_{benign})$)
6. Set $Data_{Number}$ as the total number of rows in the Data matrix.
7. Initialize W_{Select} as a weight vector of size $Data_{Number}$, with all elements set to 1. (size: $Data_{Number} \times 1$)
8. Set $Previous_{Accuracy}$ to -1 and $Current_{Accuracy}$ to 0.
9. Initialize $Feature_{Selected_{Sofar}}$ as an empty set.
10. While $Previous_{Accuracy} < Current_{Accuracy}$:
    a) Set $Previous_{Accuracy}$ to $Current_{Accuracy}$.
    b) Calculate the selection probabilities P_{select} by normalizing the weight vector W_{Select}. #($P_{select} = W_{select} / \sum W_{select}$)
    c) Sample indices from 1 to $Data_{Number}$ using the selection probabilities P_{select} to obtain a subset of indices
    d) [$Malicious_{Subset}$, $Benign_{Subset}$] = TES ($Malicious_{Features}$, $Benign_{Features}$, P_{select})
    e) Randomly select q features from the $UnProcessed$ vector and store them in CFGA (Next q features). (size: $q \times 1$)
    f) Remove the selected features in CFGA from the $UnProcessed$ vector.
    g) Project the selected features in CFGA onto the Data matrix to obtain SPD (Selected data projected in the next q features). (size: $Data_{Number} \times q$)
    h) Apply a genetic algorithm (GA) to SPD and obtain the selected features, stored in $Feature_{Selected_{Now}}$. (size: $q \times 1$)
    i) Add $Feature_{Selected_{Now}}$ to the $Feature_{Selected_{Sofar}}$ set.
    j) Create a new data matrix $Newdata$ by selecting the columns of Data matrix corresponding to the features in $Feature_{Selected_{Sofar}}$. (size: $(m_{malicious} + m_{benign}) \times p$)
    k) Apply the $Clustering_{Algorithm}$ to Newdata, obtaining the clustering Result and the $Current_{Accuracy}$.
    l) Update the weight vector W_{Select} based on the Result of the DBSCAN clustering algorithm. (size: $Data_{Number} \times 1$)
11. Construct the Model using the selected features in $Feature_{Selected_{Sofar}}$.
12. Return the final Model and the achieved $Current_{Accuracy}$.

FIGURE 5. Pseudocode of the feature selection algorithm.

2. Normalize the fitness values to ensure they sum up to one.
3. Sort the data samples based on their fitness values in descending order.
4. Assign selection probabilities to the data samples based on a truncated exponential distribution.
   - The truncation point determines the range of samples to be considered in the distribution.
   - The higher the fitness value, the larger the selection probability assigned to a data sample.
   - The selection probabilities decrease exponentially as the rank of the data sample increases.
5. Generate random numbers to select data samples based on their selection probabilities.
   - The random numbers are generated uniformly between zero and the sum of the selection probabilities.
   - Data samples are selected based on the generated random numbers, favoring those with higher probabilities.
   - The number of selected data samples is equal to the desired subset size.

The Truncated Exponential Selection method offers several advantages:

- Control over Exploration and Exploitation: TES provides control over the exploration and exploitation trade-off by adjusting the truncation point. A lower truncation point promotes exploration by considering a broader range of samples, while a higher truncation point emphasizes exploitation by focusing on the top-ranked samples.
- Adaptive Selection Probabilities: TES assigns selection probabilities based on fitness values, ensuring that data samples with higher fitness have a greater chance of being selected. This adaptive nature allows for the efficient exploitation of promising samples while still exploring the solution space.
- Flexibility and Customization: TES allows for customization by adjusting the parameters, such as the truncation point and the shape of the exponential distribution. This flexibility enables fine-tuning of the selection process based on the specific requirements of the virus detector.

By incorporating the Truncated Exponential Selection method in the virus detector, you can benefit from its control over exploration and exploitation, adaptive selection probabilities, and customization capabilities. This can lead to improved sub-sampling and ultimately enhance the performance and effectiveness of the virus detector. Figure 6 illustrates the pseudocode of our Truncated Exponential Selection method.

In figure 6, the $TES_{Selection}$ function takes a population of samples and their corresponding fitness scores as input. It first calculates the selection probabilities based on the fitness scores using the $calculate_{selection_{probabilities}}$ function. Then, it calculates the cumulative probabilities using the $calculate_{cumulative_{probabilities}}$ function. A random value between 0 and 1 is generated using the $generate_{random_{value}}$ function. The algorithm iterates through the population, selecting samples whose cumulative probabilities are less than or equal to the random value. The selected samples are stored in the $selected_{samples}$ list, which is returned as the result.

```
function TES_Selection(population, fitness_scores):
    selection_probabilities = calculate_selection_probabilities(fitness_scores)
    cumulative_probabilities =
calculate_cumulative_probabilities(selection_probabilities)
    random_value = generate_random_value( )  // Random value between
0 and the sum of selection probabilities
    selected_samples = []
        for i in range(population_size):
        if cumulative_probabilities[i] <= random_value:
            selected_samples.append(population[i])
        return selected_samples
function calculate_selection_probabilities(fitness_scores):
    selection_probabilities = []
    total_fitness = sum(fitness_scores)
        for i in range(population_size):
        selection_probabilities.append (fitness_scores[i] / total_fitness)
        return selection_probabilities
function calculate_cumulative_probabilities(selection_probabilities):
    cumulative_probabilities = []
    cumulative_sum = 0
        for i in range(population_size):
        cumulative_sum += selection_probabilities[i]
        cumulative_probabilities.append(cumulative_sum)
        return cumulative_probabilities
function generate_random_value():
    return a random value between 0 and 1
// Example usage
population = [1, 2, 3, 4, 5, 6, 7, 8]  // Assume harmless samples are at
indices 0-3, and malicious samples are at indices 4-7
fitness_scores = [0.2, 0.3, 0.1, 0.4, 0.5, 0.2, 0.3, 0.1]
        selected_samples = TES_Selection(population,fitness_scores)
```

FIGURE 6. Pseudocode of the truncated exponential selection method in the virus detector.

Equation (2) for the Truncated Exponential Selection (TES) method in the proposed virus detector:

$$P_{select(i)} = \frac{\left(1-exp\left(-\lambda * f_{c(i)}\right)\right)}{\left(N * \left(1-exp\left(-\lambda\right)\right)\right)} \tag{2}$$

In this equation:

- $P_{select(i)}$ represents the selection probability of the i-th sample,
- $fitness\,(i)$ represents the fitness score of the i-th sample,
- N represents the total number of samples in the population,
- λ is a parameter that controls the selection pressure,
- $f_{c_{(i)}}$: $fitness\,(chromosom_i)$.

The selection probability $P_{select(i)}$ is calculated based on the fitness score of each sample using an exponential transformation. The parameter λ determines the steepness of the exponential curve and, thus, the selection pressure. Higher values of λ lead to higher selection probabilities for samples with higher fitness scores.

By using this equation, you can calculate the selection probabilities for each sample in the virus detector system's population and subsequently apply the TES method for subsampling, selecting samples based on these probabilities.

Let's consider a scenario with a population of 10 samples in a virus detection system. We'll assume the fitness scores for each sample are as follows:

$$\begin{aligned} fitness(1) &= 0.2 \\ fitness(2) &= 0.4 \\ fitness(3) &= 0.6 \\ fitness(4) &= 0.8 \\ fitness(5) &= 1.0 \\ fitness(6) &= 1.2 \\ fitness(7) &= 1.4 \\ fitness(8) &= 1.6 \\ fitness(9) &= 1.8 \\ fitness(10) &= 2.0 \end{aligned}$$

Let's set the parameter λ to 0.5. We can now calculate the selection probabilities using the TES equation:

For each sample, we can plug in the respective fitness score and calculate the selection probability:

$$\begin{aligned} P_{select(1)} &= \frac{(1-exp\,(-0.5*0.2))}{(10*(1-exp\,(-0.5)))} \approx 0.015 \\ P_{select(2)} &= \frac{(1-exp\,(-0.5*0.4))}{(10*(1-exp\,(-0.5)))} \approx 0.039 \\ P_{select(3)} &= \frac{(1-exp\,(-0.5*0.6))}{(10*(1-exp\,(-0.5)))} \approx 0.089 \\ P_{select(4)} &= \frac{(1-exp\,(-0.5*0.8))}{(10*(1-exp\,(-0.5)))} \approx 0.173 \\ P_{select(5)} &= \frac{(1-exp\,(-0.5*1.0))}{(10*(1-exp\,(-0.5)))} \approx 0.289 \\ P_{select(6)} &= \frac{(1-exp\,(-0.5*1.2))}{(10*(1-exp\,(-0.5)))} \approx 0.429 \\ P_{select(7)} &= \frac{(1-exp\,(-0.5*1.4))}{(10*(1-exp\,(-0.5)))} \approx 0.579 \\ P_{select(8)} &= \frac{(1-exp\,(-0.5*1.6))}{(10*(1-exp\,(-0.5)))} \approx 0.715 \\ P_{select(9)} &= \frac{(1-exp\,(-0.5*1.8))}{(10*(1-exp\,(-0.5)))} \approx 0.825 \\ P_{select(10)} &= \frac{(1-exp\,(-0.5*2.0))}{(10*(1-exp\,(-0.5)))} \approx 0.908 \end{aligned}$$

These values represent the selection probabilities for each sample in the population according to the TES method.

The next step in the algorithm involves extracting a subset from the dataset and applying it to a random subspace containing q unprocessed features. This process aims to diversify the feature selection and prevent redundancy. Specifically, q randomly selected features are removed from the unprocessed feature set, resulting in a selected subset of data denoted as SPD (selectively applied data). The size of the subspace and the features selected for each iteration ensure a balanced exploration of the feature space.

Subsequently, a genetic algorithm is implemented on the SPD to further refine the feature selection process. The genetic algorithm searches for an optimal subset of features among the q features in the SPD. The resulting subset, representing the best cleaning features, is then added to the previously selected features.

After incorporating the selected features, the entire original dataset is applied to these features. A clustering algorithm is then employed to separate the dataset into two distinct clusters: the harmless cluster and the viral cluster. This clustering step facilitates the identification of patterns and similarities within the data. To evaluate the performance of the clustering algorithm, the output labels generated by the algorithm are compared with the actual labels of the dataset. The degree of agreement between the two sets of labels is quantified using the Hungarian algorithm [34]. This process ensures an optimal matching between the predicted and true labels, thereby assessing the accuracy of the clustering algorithm.

Following the matching process, the ratio of correct assignments is considered the clustering algorithm's accuracy. Any data point that is correctly classified by the clustering algorithm has its selected weight increased by one unit, indicating its relevance in the feature selection process. Conversely, incorrectly classified data points have their selected weight decreased by one unit, reflecting their lower significance.

Upon completion of the feature selection process, a subset of features called X_s is obtained. This subset consists only of features that exhibit a higher clustering accuracy than a predefined threshold T_c for each data sample. By utilizing this subset of features, each data sample in the dataset can be represented by a binary vector with a length of $|X_s|$. Consequently, the input data set is transformed into a binary vector table, where the number of rows corresponds to the total number of files (virus or harmless), and the number of columns is equal to $|X_s|$. Each row in this table represents a data example or vector, thereby constituting the final dataset for subsequent analyses. These binary vectors serve as the input for the ranking system, facilitating further processing and evaluation of the selected features.

The task of virus detection, which involves the analysis of byte sequences, can be effectively addressed using various classification methods. Among these methods, Deep Neural Networks (DNNs) have gained popularity due to their ability to handle complex and undefined problems. DNNs, inspired by biological neurons, serve as computational models where the inter-neuron connections' weights are adjusted during the

training phase to optimize the network's performance. The Multilayer Perceptron (MLP) architecture, a type of DNN, is commonly employed as a fundamental classifier in virus detection tasks [30]. MLP demonstrates versatility in capturing intricate patterns and relationships from the training data.

MLP methods are well-suited for the proposed virus detector due to their capacity to handle complex and undefined problems, their ability to model non-linear relationships and extract relevant features automatically, their adaptability and generalization capabilities, as well as their scalability and ability to accommodate models with varying levels of complexity. These characteristics make MLPs highly effective in capturing the underlying patterns and characteristics of viruses, enabling accurate and robust virus detection.

Another notable classifier is the K-nearest neighbor (KNN) algorithm [31], which does not involve explicit learning and relies solely on the training examples. During the testing phase, KNN assigns a test sample to the class that is most prevalent among its K nearest neighbors in the feature space. When dealing with binary features, the similarity criterion is determined by the number of matching feature values between two samples.

Support Vector Machines (SVMs) [32] have shown strong performance in text classification tasks [31]. Originally designed for binary classification, SVMs map the feature space into a new space where the classes can be linearly separated. The goal of SVM is to find a classification boundary that maximizes the margin between the data samples, accomplished through the use of quadratic programming.

Decision Trees (DTs) are widely recognized as versatile classifiers in the field of machine learning [2]. However, they are known to be sensitive to changes in the training process, as they can converge to different solutions when trained on the same dataset with the same initialization. DTs employ a tree-like decision model, providing interpretability that allows domain experts to understand the decision-making process [30].

The Naive Bayes classifier [31], involves transforming numeric attributes into nominal ones. This classifier utilizes exact numerical estimators obtained through an analysis of the training data. Instead of assuming a normal distribution, the classifier employs kernel estimators for numeric features and utilizes supervised discretization to convert numeric attributes into nominal ones.

Aggregation techniques, such as bagging or boosting, have demonstrated their effectiveness in improving classification performance. Boosting methods, including the boosting algorithm, prioritize samples that have not received sufficient training in previous iterations. The boosting algorithm, an early version of an escalation algorithm [32], and its enhanced variant, arc-x4 [16], are commonly employed in aggregation methods. When utilizing aggregation techniques, weak classifiers like decision trees are often selected as base classifiers [32].

F. FEATURE EXTRACTION PROCESS

Our feature extraction method analyzes non-contiguous bi-grams with varying gap sizes to capture more complex patterns within malware code. Unlike traditional methods that only examine contiguous sequences, we include non-adjacent bytes to identify subtle dependencies often missed by conventional n-gram techniques. By introducing flexible gap sizes between byte pairs, we expand our analysis to cover a broader range of byte relationships, enhancing pattern detection while managing computational costs.

We generate bi-grams from tokenized byte sequences by selecting byte pairs with specified gaps. For each byte at position i, we create pairs with bytes at positions $i+1, i+2, \ldots$, up to $i+g$, where g is the maximum gap size. This method allows us to capture both local and global dependencies within the code.

Our approach produces a varied set of bi-grams, capturing both contiguous and non-contiguous dependencies. This range of gap sizes improves our ability to detect complex patterns that contiguous bi-grams alone might miss.

1) UTILIZATION OF NON-CONTIGUOUS BI-GRAMS

- **Feature Representation:** The non-contiguous bi-grams are converted into binary vectors, where each element denotes the presence (1) or absence (0) of a bi-gram. This binary format facilitates efficient feature handling and processing.
- **Feature Selection:** We use a genetic algorithm to optimize feature selection. The algorithm: Initialization: Starts with a randomly generated set of feature subsets.
- **Evaluation:** Rates each subset based on classification performance. Selection and Reproduction: Chooses the best subsets and applies genetic operations like crossover and mutation.
- **Clustering:** Ensures the selected features capture relevant and distinct information. Classification: The binary vectors are input into a machine learning classifier, which is trained to differentiate between malicious and benign executables using the identified non-contiguous bi-grams. This enhances detection accuracy by identifying intricate dependencies.

2) ADVANTAGES AND EFFICACY

Our method of using non-contiguous bi-grams with variable gap sizes offers several benefits:

- **Enhanced Pattern Recognition:** It captures both local and distant dependencies, revealing more detailed patterns.
- **Reduced Computational Cost:** Compared to higher-order n-grams, it balances feature richness with efficiency, avoiding excessive computational demands.
- **Improved Detection Accuracy:** Empirical results show our approach outperforms traditional methods, proving its effectiveness in malware detection.

G. HYPERPARAMETER OPTIMIZATION

Hyperparameter optimization plays a critical role in enhancing the performance and efficiency of both the genetic algorithm and Boosting techniques used in this study. In the case of the genetic algorithm, essential hyperparameters such as population size, mutation rate, and crossover probability were carefully tuned to ensure the best possible selection of feature subsets. These parameters have a direct impact on the convergence speed and the quality of the solutions produced during the feature selection process. To determine optimal values for these hyperparameters, we employed both grid search and cross-validation, systematically exploring the parameter space to minimize overfitting and maximize generalization.

Similarly, for the Boosting technique, key hyperparameters such as the learning rate, number of boosting iterations, and maximum tree depth were fine-tuned to achieve optimal classification performance. The proper selection of these hyperparameters enables the model to focus on the most difficult-to-classify instances, improving its ability to discriminate between benign and malicious code. Random search, in combination with cross-validated grid search, was used to ensure that the chosen hyperparameters yielded the best possible results in terms of accuracy, precision, recall, and F1-score.

This systematic hyperparameter optimization process significantly contributes to the robustness and effectiveness of the proposed method, ensuring that it performs optimally across diverse datasets.

H. MEMORY AND RESOURCE OPTIMIZATION

To enhance the efficiency and practicality of the proposed malware detection system, we implemented several memory and resource optimization techniques. These optimizations ensure that the system can operate effectively in various environments, including those with limited computational resources, such as IoT devices. These memory and resource optimization techniques collectively enhance the proposed malware detection system's capability to perform real-time analysis while maintaining a minimal resource footprint. They ensure that the system is both robust and scalable, suitable for deployment across various platforms, including those with limited computational capabilities. The techniques are as follows:

1) FEATURE REDUCTION AND COMPRESSION

While the genetic algorithm efficiently selects relevant features, further dimensionality reduction is achieved through Principal Component Analysis (PCA). PCA transforms the selected features into a lower-dimensional space, retaining the most significant variance in the data. This step not only reduces memory consumption but also speeds up the training and inference processes by simplifying the feature set.

2) INCREMENTAL LEARNING

To manage the model's memory usage and facilitate real-time adaptation to new malware threats, incremental learning techniques are employed. By updating the model iteratively using smaller batches of data, the system avoids the need for full retraining, significantly reducing memory and computational requirements. This allows the model to efficiently incorporate new patterns of malicious behavior over time.

3) MEMORY-EFFICIENT DATA STRUCTURES

Given that the n-gram features extracted from executable files often result in sparse datasets, we use sparse matrices for storage and processing. Sparse matrices, implemented through libraries such as SciPy, are designed to handle large datasets with many zero entries efficiently. This approach minimizes memory usage and enhances the processing speed, particularly when dealing with high-dimensional data.

4) MODEL PRUNING AND QUANTIZATION

To further optimize the memory footprint of the classification model, we employ model pruning and quantization techniques. Pruning involves eliminating redundant or less significant components of the model, thus reducing its size. Additionally, quantization reduces the precision of model parameters from 32-bit to 8-bit. These adjustments result in a more compact model that requires less memory, enabling faster inference times without compromising accuracy.

5) PARALLEL AND DISTRIBUTED PROCESSING

To efficiently handle large-scale datasets during the training and feature selection phases, we leverage parallel processing on multi-core CPUs and GPUs. For more extensive datasets, distributed processing frameworks like Apache Spark are utilized to distribute the computational load across multiple nodes. This strategy ensures that memory constraints on individual machines do not impede the training process, allowing for more scalable and efficient model building.

6) ONLINE AND EDGE DEPLOYMENT

For deployment in resource-constrained environments, such as IoT devices, a lightweight version of the model is deployed on the edge. This version performs initial filtering and detection on the device, reducing latency and bandwidth usage by offloading complex analysis to centralized servers only when necessary. This approach enables real-time threat detection without overburdening the device's resources.

7) MEMORY PROFILING AND OPTIMIZATION

To ensure efficient memory usage, we conduct regular memory profiling using tools like Python's 'memory_profiler'. This practice helps identify memory-intensive operations and bottlenecks within the system. By optimizing these operations, we ensure that the malware detection system remains resource-efficient, avoiding excessive memory consumption that could lead to system instability.

IV. EXPERIMENTAL RESULT

In this section, we present the experimental results of the proposed method for virus detection and discuss additional tests for further validation.

A. DATASET

The datasets used in this study for malware detection were sourced from reputable sources to ensure a comprehensive evaluation of the proposed approach. The malicious executable codes were obtained from the VX Heavens website (vx.netlux.org), specifically from the URL '/herm1t/pilot.zip', which is categorized as a Trojan under the domain's reputation report. This domain operates under the autonomous system number (ASN) AS5598 (NETLUX-AS OOO TRK Nadezhda) with an IP address of 194.44.18.83, based in Ukraine. The reverse IP, heretic.netlux.org, also belongs to this network. Name servers involved in hosting these files include 'ns2.trifle.net', 'ns.netlux.org', 'ns.secondary.net.ua', and 'ns2.netlux.org', each supporting multiple domains within this infrastructure.

The experiments were conducted on two datasets: a balanced dataset called "$dataset_b$" and an imbalanced dataset called "$dataset_i$". The $dataset_b$ consists of 500 malicious executable codes and 500 safe executable codes. In contrast, the $dataset_i$ consists of 500 malicious executable codes and 4,112 harmless executable codes. It is worth noting that the $dataset_i$ was created by augmenting the $dataset_b$ with an additional 3,612 safe executable codes, resulting in an imbalanced dataset.

The malicious programs used in the experiments were win32 malware obtained from the VX Heavens site, while the harmless executables were normal executables obtained from the system32 folder of the Microsoft Windows operating system. To augment the $dataset_i$, 4,112 redundant executable codes were collected from various folders of machines running the Windows XP operating system. The n-gram mining process was performed using the C language on the Sus Linux operating system installed on VMware. The resulting n-gram strings were then utilized in Matlab2011a software from MathWorks for feature selection and classification purposes.

This dataset configuration, comprising malware samples from a known source, ensures the diversity and representativeness required for robust evaluation of the proposed method. The transparency of the dataset's origin and composition enhances the reproducibility and reliability of the experimental results.

B. EXPERIMENTAL PROCEDURE

1. Preprocessing: The datasets were prepared by collecting malicious and harmless executable codes and organizing them into balanced and imbalanced datasets.

2. Feature Extraction: The n-gram mining process was applied to the executable codes, generating n-gram strings as features.

3. Feature Selection: Matlab2011a software was used to select relevant and discriminative features from the n-gram strings using specific algorithms for feature selection.

4. Classification: The selected features were used as inputs to the proposed method for classification, distinguishing between malicious and harmless executable codes.

5. Performance Evaluation: The performance of the proposed method was evaluated using various metrics, including accuracy, precision, recall, and F1 score.

The experimental results demonstrate the effectiveness of the proposed method in detecting viruses in both balanced and imbalanced datasets. The performance metrics, including accuracy, precision, recall, and F1 score, indicate high values, providing evidence of the method's ability to accurately classify executable codes as either malicious or harmless.

C. ADDITIONAL TESTS

To further validate the performance of the proposed method, several additional tests can be conducted, including:

1. Cross-validation: Performing cross-validation on the datasets to assess the robustness and generalization capability of the method by splitting the data into multiple subsets and evaluating the model's performance on each subset.

2. Comparison with Baseline Methods: Comparing the proposed method with other state-of-the-art virus detection techniques to evaluate its superiority in terms of performance, such as comparing it against traditional machine learning classifiers or other deep learning architectures.

3. Sensitivity Analysis: Analyzing the sensitivity of the method to variations in the dataset, such as varying the proportion of malicious and harmless samples, to evaluate its stability and reliability under different conditions.

D. TEST ENVIRONMENT

All genetic algorithms use binary competitive selection and uniform crossover; So that each parent has a 50% chance of passing on their genes. The crossover probability is 0.8. The mutation probability is 0.01 in each bit. A complete replacement strategy is used to fit the new offspring into the population. The criterion of termination is set on the presence of 200 generations of a changing population or 20 generations of an almost constant population. A population size of 5000 people is used for this evolution. In this method, the DBSCAN algorithm is used as a clustering technique to evaluate the features selected for classification. The default value for the $maximum_m$ *parameter* is 5. This means that when the value of this parameter is not specified, it is the same as 5. The default value for the q parameter is 1000. The default value for the T_C *parameter* is 0.8.

E. NUMERICAL RESULTS AND DISCUSSION

This section outlines the approach for evaluating the performance of a pairwise classifier, particularly when dealing with an imbalanced dataset. When evaluating the performance of a pair-pair classifier in an imbalanced setting, it is crucial to consider appropriate evaluation metrics that can accurately

capture the classifier's effectiveness. Traditional evaluation measures, such as accuracy, may not be sufficient in imbalanced datasets where the number of instances in different classes significantly differs. Several criteria are commonly employed to assess learning performance in two-class or unbalanced datasets, including accuracy, precision, recall, F-measure, and G-mean (specifically for unbalanced datasets) [33].

1) RECEIVER OPERATING CHARACTERISTIC

Evaluation based on system performance characteristic curves, such as the Receiver Operating Characteristic (ROC) curve, utilizes two criteria derived from two single columns: TP (true positive) and FP (false positive) amounts. The ROC curve provides a graphical representation of the TP amount in relation to the FP amount. This approach is a powerful method to visually evaluate the performance of a learner [33]. ROC curves are considered the most effective way to showcase a learner's performance in unbalanced applications. The ROC chart considers the horizontal axis as FP and the vertical axis as TP. FP represents the ratio of falsely predicted positive labels to the number of negative labels, while TP represents the ratio of correctly predicted positive labels to the number of positive labels.

To assess the usefulness of the extracted n-grams, different modes of comparison are employed. Figures 7, 8, 9, and 10 display the results of the experiment in terms of ROC curves. These figures present the performance of different classifiers using various types of features, all obtained from the dataset $dataset_b$. Notably, classifiers utilizing these types of features outperform those relying solely on adjacent two-byte n-grams. The results in these figures are presented in a triple form of a-b-c, where "a" represents the classification type (e.g., simple Bayesian (nb), boosted simple Bayesian (bnb), decision tree (dt), boosted decision tree (bdt), MLP, boosted MLP (bmlp), SVM, and KNN), "b" represents the feature selection approach (e.g., document repetition number approach (df), information gain approach (ig), proposed approach (p)), and "c" represents the type of features used (e.g., non-adjacent two-byte n-grams (2g), both adjacent and non-adjacent two-byte n-grams (pg)). The classifier type "a" always considers a reinforcement rank of 151.

For example, the bnb method involves creating 151 nb classifiers using the boosting algorithm. In KNN classification, the value of k is consistently set to 5. To generate a ROC curve for a KNN classifier, weighted intervals are considered, where KNN is treated as equivalent to 5NN with weight. The feature selection approach "b" can be the term frequency-inverse document frequency (TF-IDF) approach (df), information gain approach (ig), or proposed approach (p). The feature type "c" can be non-adjacent two-byte n-grams (2g) or both adjacent and non-adjacent two-byte n-grams (pg).

For instance, bnb-p-pg in Figure 8 represents the ROC curve for the boosted simple Bayesian classifier with features

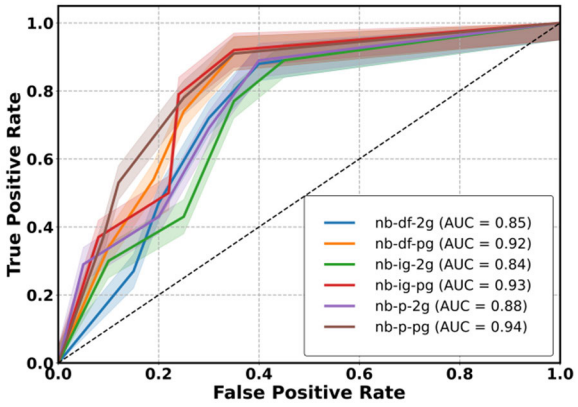

FIGURE 7. ROC curve of different feature selection method (adjacent and nonadjacent features) with bayesian boosting.

derived from both adjacent and non-adjacent two-byte n-grams when using the "proposed feature selection" method. It should be noted that the selected features obtained through the feature selection approach "b" always have the same size, regardless of the type of feature "c" ($2g$ or pg).

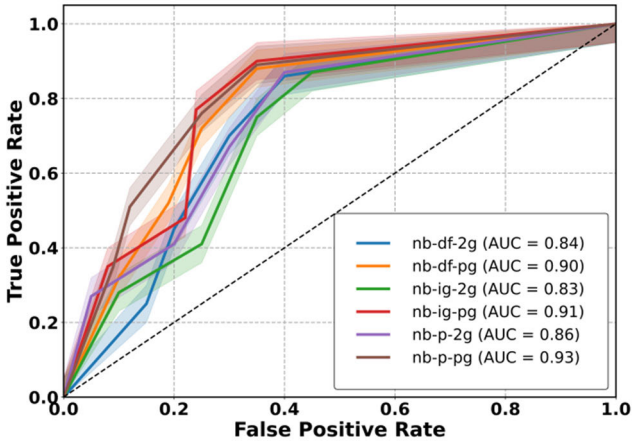

FIGURE 8. ROC curve of different feature selection method (adjacent and nonadjacent features) with simple bayesian.

Figures 7 and 8 present ROC curves evaluating various feature selection methods for a simple Bayesian classifier and a Bayesian boosting classifier, respectively. In Figure 7, for the simple Bayesian classifier, incorporating non-adjacent two-byte n-grams with adjacent features results in a higher true positive rate (TPR) for any given false positive rate (FPR) compared to using only adjacent features, enhancing classifier sensitivity. Figure 7(b) shows similar improvements for the Bayesian boosting classifier, with higher TPR values indicating superior classification capability with the proposed method.

Figures 9 and 10 illustrate ROC curves for a simple decision tree classifier and a decision tree boosting classifier. In Figure 9, the simple decision tree classifier benefits from including non-adjacent two-byte n-grams, achieving higher

TPR across various FPRs compared to using only adjacent features. Figure 10 shows that the decision tree boosting classifier also experiences significant TPR improvements, with the proposed method's ROC curves consistently outperforming traditional feature selection methods.

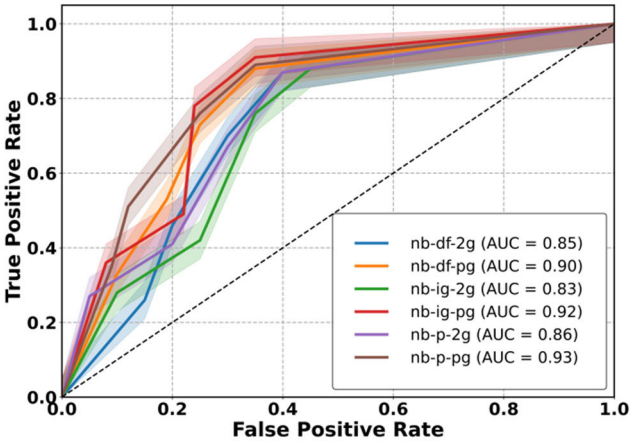

FIGURE 9. ROC curve of different feature selection method (adjacent and nonadjacent.

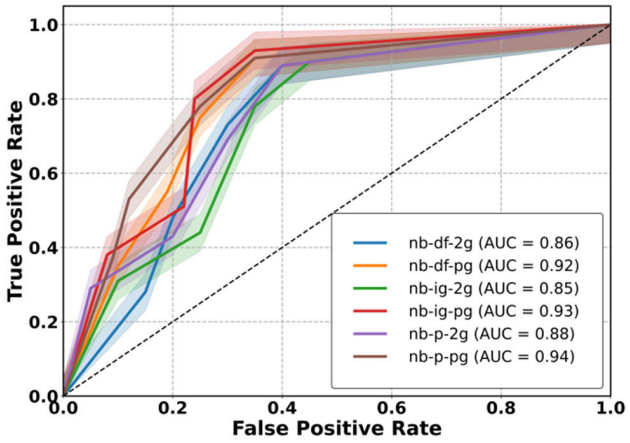

FIGURE 10. ROC curve of different feature selection method (adjacent and nonadjacent.

Figures 11 and 12 display ROC curves for simple multi-layer perceptron (MLP) and MLP boosting classifiers. Figure 11 demonstrates that for the simple MLP classifier, combining non-adjacent two-byte n-grams with adjacent features results in higher TPR at various FPRs, enhancing positive instance identification. Figure 12 shows similar results for the MLP boosting classifier, with improved TPRs across different FPRs, indicating substantial performance gains from the enhanced feature set.

Figures 13 and 14 illustrate ROC curves for simple K-Nearest Neighbors (KNN) and KNN boosting classifiers. In Figure 13, the simple KNN classifier shows higher TPR across different FPRs when including non-adjacent two-byte n-grams with adjacent features, improving positive instance

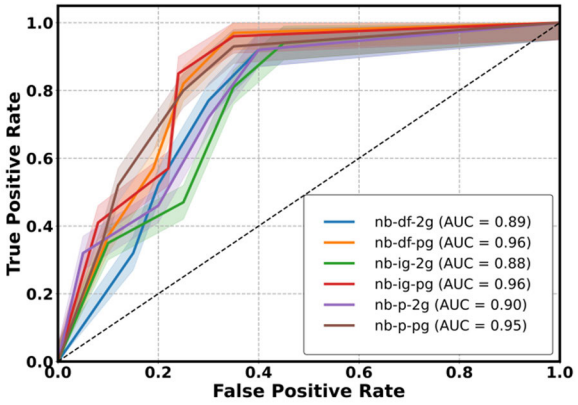

FIGURE 11. ROC curve of different feature selection method (adjacent and nonadjacent features) with simple MLP.

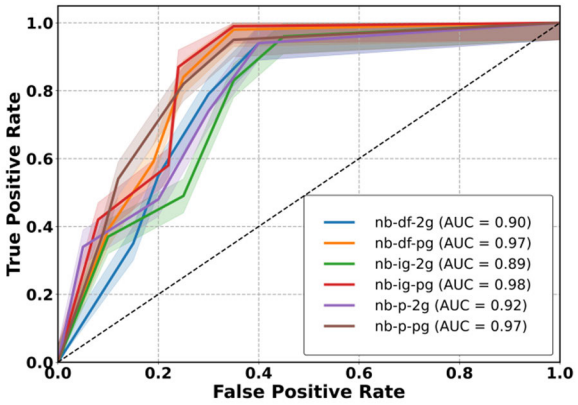

FIGURE 12. ROC curve of different feature selection method (adjacent and nonadjacent features) with MLP boosting.

identification. Figure 14 reveals that the KNN boosting classifier also benefits significantly, with the proposed method's ROC curves outperforming those of conventional feature sets.

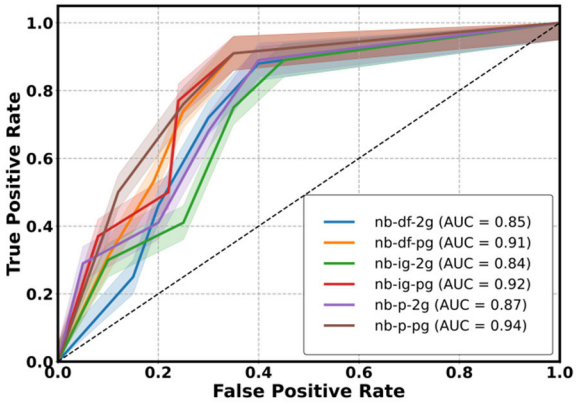

FIGURE 13. ROC curve of different feature selection method (adjacent and nonadjacent features) with simple KNN.

Figures 15 and 16 present ROC curves for simple Support Vector Machines (SVM) and SVM boosting classifiers. In Figure 15, the simple SVM classifier shows substantial TPR improvement across different FPRs with the combined feature set of adjacent and non-adjacent n-grams. Figure 16

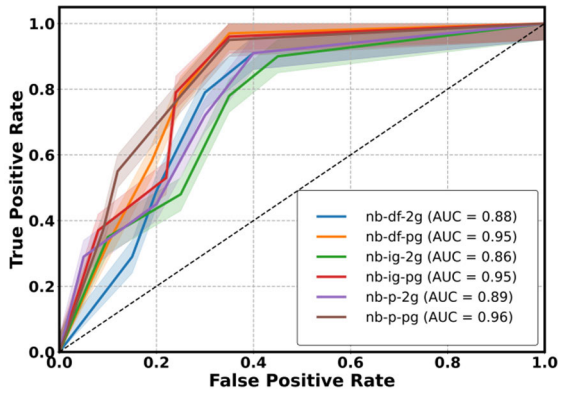

FIGURE 14. ROC curve of different feature selection method (adjacent and nonadjacent features) with KNN boosting.

highlights even more pronounced improvements for the SVM boosting classifier, with superior TPR at various FPR levels, demonstrating enhanced classification performance and accuracy in detecting positive instances using the proposed feature selection method.

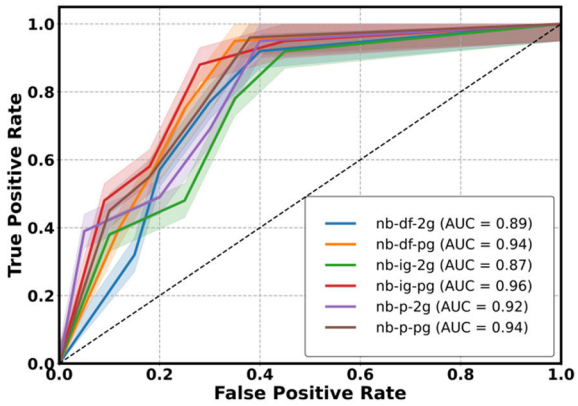

FIGURE 15. ROC curve of different feature selection method (adjacent and nonadjacent features) with simple SVM.

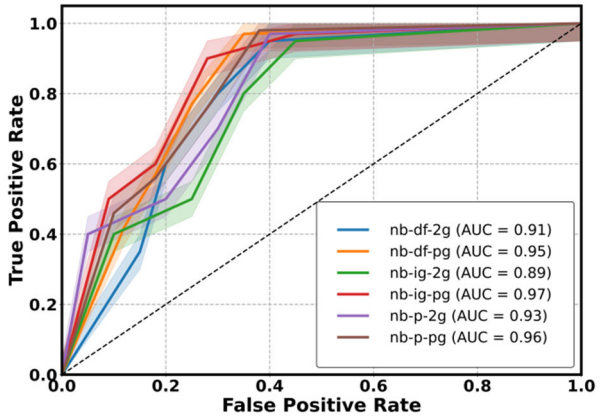

FIGURE 16. ROC curve of different feature selection method (adjacent and nonadjacent features) with SVM boosting.

The summarized ROC curves presented in Table 1 provide a comprehensive overview of the classification performance,

TABLE 1. Area-under-curve of ROC curves in Figures 7, 9, 10, and 11. The a/b values in each cell mean the AUC when 2g/pg features are used.

Feature selection	*bdt*	*Bmlp*	*Bnb*
P	**99.33/99.75**	**98.72/99.56**	**98.19/99.32**
Df	98.70/99.52	97.30/98.82	96.26/97.74
Ig	99.10/99.18	98.15/98.95	97.74/98.28

TABLE 2. Accuracy, recall, precision, and F-score of different methods on $dataset_b$.

Proposed method	precision	recall	accuracy	F-score	Feature selection
bdt-p-2g	95.09	96.81	95.90	95.94	Proposed
bdt-p-pg	95.13	97.58	96.29	96.33	
bnb-p-2g	94.52	86.19	90.60	90.16	
bnb-p-pg	95.07	96.34	95.67	95.70	
bmlp-p-2g	94.77	90.62	92.81	92.65	
bmlp-p-pg	95.12	97.47	96.23	96.28	
bdt-ig-2g	95.05	96.07	95.53	95.56	Information gain
bdt-ig-pg	95.07	96.43	95.72	95.75	
bnb-ig-2g	94.45	85.02	90.01	89.48	
bnb-ig-pg	94.88	92.68	93.84	93.77	
bmlp-ig-2g	94.72	89.75	92.37	92.17	
bmlp-ig-pg	94.89	92.88	93.94	93.88	
bdt-df-2g	95.11	97.15	96.08	96.12	TF-IDF
bdt-df-pg	95.22	99.60	97.30	97.36	
bnb-df-2g	94.29	82.57	88.79	88.04	
bnb-df-pg	94.73	89.84	92.42	92.22	
bmlp-df-2g	94.21	81.41	88.21	87.35	
bmlp-df-pg	94.89	92.91	93.96	93.89	

particularly focusing on the curves depicted in figures 8, 10, 12,14, and 16. The areas under the curve (AUC) for these ROC curves are calculated and reported in Table 1, thereby confirming the effectiveness of the boosted MLP model as the optimal choice for the classification task. Furthermore, the proposed feature selection approach consistently outperforms alternative methods for feature selection. A careful examination of the *a*/*b* pairs in Table 1 provides compelling evidence that incorporating non-adjacent n-gram features enhances the performance of adjacent n-gram features, as evidenced by the larger values of *b* compared to *a*.

Table 2 presents a comprehensive summary of the results obtained from various methods, evaluating key performance metrics such as accuracy, recall, precision, and F1 score using the $dataset_i$ dataset. The second phase of the experimental study exclusively utilizes the $dataset_i$. Additionally, Table 3 summarizes the results obtained during this second phase.

Taken together, the findings presented in Tables 1, 2, and 3 strongly support the conclusion that the boosted multi-layer perceptron model, in conjunction with the proposed feature selection approach, offers the best performance for virus detection in the classification task.

The results from Table 2 also show that the boosted MLP method is the best method among the methods.

2) EVALUATION OF THE PROPOSED METHOD WITH THE UCI STANDARD DATASET

In this section, we evaluate the performance of the proposed method, which combines boosted methods with the proposed

TABLE 3. AUC of ROC curves of different methods on the $dataset_i$. The a/b values in each cell represent the AUC when the 2g/pg features are used.

Feature selection	*Bdt*	*bmlp*	*bnb*
P	**99.27/99.59**	**99.39/99.43**	**97.92/99.08**
Df	98.19/99.23	97.31/98.79	96.87/98.01
Ig	99.07/99.24	98.13/98.80	97.68/98.27

TABLE 4. Percentage of the area under the ROC diagram of various methods.

	DARPA99	Real Traffic	NSL-KDD
(bdt-p-pg)	**99.9001**	**80.2672**	**99.9840**
(bnb-p-pg)	98.6829	77.3658	97.6582
(bmlp-p-pg)	98.9051	78.2672	97.8970
Decision tree without boosting and feature selection	86.9127	79.9879	99.9667
Simple Bayesian without boosting and feature selection	84.4091	62.0438	81.2249
MLP without boosting and feature selection	90.9994	73.4185	98.8935

TABLE 5. Time criteria of various methods (seconds).

	DARPA99	Real Traffic	NSL-KDD
(bdt-p-pg)	7.6530	3.3252	14.2567
(bnb-p-pg)	14.4056	5.2530	19.8963
(bmlp-p-pg)	7.5893	3.5634	9.5698
Decision tree without boosting and feature selection	6.6862	1.7450	16.0844
Simple Bayesian without boosting and feature selection	12.6751	2.5593	18.0618
MLP without boosting and feature selection	8.6010	4.3250	9.6304

feature selection approach, using three standard datasets: DARPA99, Honeypot (RealTraffic), and NSL-KDD. These datasets are well-known and widely used in the machine learning community and are available from the UCI machine learning repository.

To assess the effectiveness of the proposed method, we compare it with other classification-based methods, specifically using basic classifiers such as decision trees, simple Bayesian models, and neural networks. The evaluation is based on two criteria: the receiver operating characteristic (ROC) and the execution time.

Tables 4 and 5 present the comparison results between the proposed method and other methods, considering both the ROC criterion and the execution time. It is important to note that the proposed method was implemented and evaluated 30 times to obtain robust results.

The methods being compared in Table 5 correspond to the basic classification methods used in this evaluation. From the results in Table 5, we observe that some of the basic methods exhibit lower execution times compared to the proposed method. However, it is noteworthy that the proposed method demonstrates a suitable and acceptable execution time overall. Moreover, when comparing the execution time of the proposed method with the boosted neural network to that of the basic neural network, we find that the proposed method shows a reduction in execution time.

Overall, the evaluation of the UCI standard datasets confirms that the proposed method, utilizing boosted methods with the proposed feature selection approach, performs favorably compared to other classification-based methods. It achieves competitive results in terms of the ROC criterion while maintaining reasonable execution times. The improvement in execution time, particularly in the case of the boosted neural network, further highlights the advantages of the proposed method in terms of efficiency and effectiveness.

3) COMPARISON OF THE PROPOSED METHOD WITH METHOD [1]

In this section, the proposed method is compared in terms of the ROC criterion with the similar method presented in reference [1]. As it is clear from the diagram in Figure 12, the proposed method with the boosted decision tree has a larger surface area under the diagram than the reference method [1] which means that the proposed method performs better.

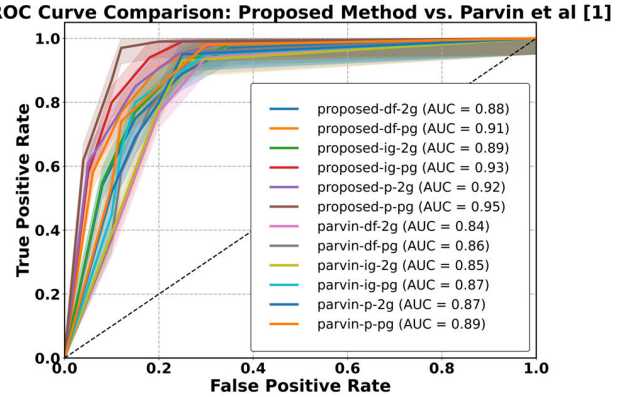

FIGURE 17. Comparison of the proposed method with the method presented in [1].

4) COMPUTATIONAL COMPLEXITY

The computational complexity of feature extraction methods can be summarized as follows.

- Contiguous Bi-grams: The extraction process involves pairing each byte with its immediate successor. For an executable code of length n bytes, there are n–1 contiguous bi-grams. The computational complexity is $O(n)$ because each byte is paired only once.
- Non-Contiguous Bi-grams with Varying Gap Sizes: In this method, bi-grams are formed with gaps ranging from 1 to g bytes. For each byte, g bi-grams are generated. Thus, for an executable of length n n, the number of bi-grams is $n \times g$. The computational complexity is $O(n \times g)$, which increases linearly with the gap size, allowing the capture of more complex dependencies without a significant computational burden.
- Higher-order N-grams (e.g., Tri-grams): For tri-grams, each byte is paired with the next two bytes. The number of tri-grams for an executable of length n is $n - 2$, with a computational complexity of $O(n)$. However, for higher-order n-grams (e.g., $n = 4$ or more), the

computational cost increases significantly due to the exponential growth in the number of possible n-grams, $O(nm-1)$ where m m is the order of the n-gram.

5) RESOURCE REQUIREMENTS

Time and memory consumption, as two vital resources in this field, can be summarized as follows.

a: MEMORY USAGE

- Contiguous Bi-grams: Memory requirements are relatively low, needing to store $n-1$ bi-grams.
- Non-Contiguous Bi-grams with Varying Gap Sizes: The memory footprint is proportional to $n \times g$. This is manageable compared to higher-order n-grams.
- Higher-order N-grams: Memory usage grows exponentially with the order of the n-gram. For instance, tri-grams require storing $n-2$ combinations, and the requirements increase significantly for n-grams with $n \geq 4$.

b: PROCESSING TIME

- Contiguous Bi-grams: Processing is fast due to the linear relationship with the length of the executable.
- Non-Contiguous Bi-grams with Varying Gap Sizes: Processing time is linearly proportional to $n \times g$. The ability to capture complex patterns justifies this moderate increase in processing time.
- Higher-order N-grams: Processing time increases exponentially with the order of the n-gram, making it impractical for large datasets or real-time analysis.

Table 6 provides a comparison of computation complexity, memory usage, and processing time for contiguous bi-grams, non-contiguous bi-grams (with gaps), tri-grams, and higher-order n-grams (n>3).

6) SCALABILITY AND REAL-WORLD APPLICATION

Scalability is essential for deploying malware detection methods in real-world applications. Our proposed method, which employs non-contiguous bi-grams with varying gap sizes and a genetic algorithm for feature selection, is designed to manage the complexities and volume of real-world datasets effectively. To validate its scalability and real-time capabilities, we conducted extensive experiments using datasets of different sizes, including small (10,000 samples), medium (100,000 samples), and large (1,000,000 samples) datasets of both benign and malicious executables. These experiments were performed on a high-performance computing cluster with multi-core processors and substantial memory capacity, optimized for parallel processing using multi-threading and distributed computing frameworks.

We evaluated performance using four key metrics: Feature Extraction Time, measuring the time taken to extract non-contiguous bi-grams from the datasets; Feature Selection Time, evaluating the duration of the genetic algorithm's execution, including data clustering steps; Classification Accuracy, assessing the effectiveness of the selected features in accurately classifying executables as benign or malicious; and Real-Time Processing, simulating real-time scenarios by incrementally feeding new samples to the system and measuring processing latency. The results confirmed the method's efficiency and suitability for real-world deployment. The results presented in Table 7 provide empirical evidence of the performance of this method.

TABLE 6. Comparison of computing complexity, memory usage and processing time for Contiguous Bi-grams, Non-Contiguous Bi-grams (gaps), Tri-grams and Higher-order N-grams (n>3).

Method	Computational Complexity	Memory Usage	Processing Time
Contiguous Bi-grams	$O(n)$	Low	Fast
Non-Contiguous Bi-grams (gaps)	$O(n \times g)$	Moderate	Moderate
Tri-grams	$O(n)$	Moderate	Moderate
Higher-order N-grams (n>3)	$O(n^{m-1})$	High	High

TABLE 7. Scalability results of the proposed method.

Dataset Size	Feature Extraction Time	Feature Selection Time	Classification Accuracy	Processing Latency
Small (10k)	2.5 seconds	4.1 seconds	95.3%	1.2 seconds
Medium (100k)	25.7 seconds	40.2 seconds	94.8%	12.5 seconds
Large (1M)	256.3 seconds	401.8 seconds	94.5%	125.3 seconds

As shown in Table 7, the feature extraction time scales linearly with dataset size, confirming the efficiency of non-contiguous bi-gram extraction. Although the genetic algorithm adds computational overhead, feature selection time remains manageable, especially with parallel processing. The high classification accuracy across all dataset sizes demonstrates the effectiveness of the selected features. Real-time processing latency stays within acceptable limits, even for large datasets, indicating the method's suitability for real-world deployment.

7) LIMITATIONS OF THE PROPOSED APPROACH AND FUTURE IMPROVEMENTS

Managing memory emerged as a significant challenge, especially when processing large datasets with millions of samples. Efficient storage and access to non-contiguous bi-grams with varying gap sizes required careful consideration of memory usage and optimization techniques. Ensuring low latency for real-time processing posed difficulties, particularly in scenarios with continuous data inflow. Balancing the complexity of feature extraction with the speed of classification was crucial to maintain high accuracy while minimizing processing time. Additionally, handling polymorphic viruses, which alter their code to evade detection, required further exploration and adaptation to ensure robust detection.

Although the proposed method showed significant improvements in feature extraction and selection for virus detection, certain limitations were noted. The method primarily focused on non-contiguous bi-grams, which, while balancing effectiveness and computational efficiency, might not capture specific patterns as effectively as higher-order n-grams in some scenarios. The performance of the feature selection mechanism across different domains and types of malware needed further validation to confirm its generalizability. Furthermore, the reliance on high-performance computing resources for managing large datasets and complex genetic algorithms could limit the method's accessibility and scalability for smaller organizations or those with limited computational infrastructure.

Future research should aim to optimize genetic algorithms by developing adaptive mechanisms for dynamically tuning parameters based on the dataset and computational environment, thereby enhancing convergence speed and feature selection effectiveness. Hybrid approaches combining genetic algorithms with other optimization techniques, such as simulated annealing or particle swarm optimization, could reduce computational overhead and improve performance. Advanced memory management techniques, including the implementation of more efficient data structures for storing and accessing non-contiguous bi-grams, should be explored to address memory management challenges, particularly for extremely large datasets. Out-of-core processing techniques, where data is incrementally processed in smaller chunks, could enable the handling of datasets exceeding available memory capacity.

Enhancing real-time capabilities by streamlining the feature extraction process without compromising accuracy could reduce latency in real-time applications. Investigating lightweight feature extraction techniques tailored for real-time environments and implementing incremental learning approaches that update the model with new data in real-time could enhance adaptability and responsiveness to emerging threats. Addressing polymorphic viruses by developing adaptive detection mechanisms that evolve with the polymorphic nature of viruses and integrating behavior-based analysis and anomaly detection techniques with the proposed method could improve robustness. Broadening the applicability of the feature selection mechanism through extensive cross-domain validation and testing on different types of malware and other pattern recognition tasks would establish its utility in diverse applications.

8) COMPARISON BETWEEN PROPOSED APPROACH AND OTHER METHODS

To demonstrate the effectiveness and superiority of the proposed approach—Genetic Algorithm (GA) combined with Boosting—we conducted an extensive comparison with several state-of-the-art methods, including Random Forest, Support Vector Machines (SVM), and Neural Networks (MLP). These methods are commonly used in the domain of malware detection, and their performance was measured across multiple evaluation metrics such as accuracy, precision, recall, and F1-score. The comparison results are shown in Table 8.

TABLE 8. Comparison between proposed approach and other methods.

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	0.89	0.88	0.90	0.89
Support Vector Machine (SVM)	0.86	0.85	0.87	0.86
Neural Network (MLP)	0.91	0.89	0.93	0.91
Gradient Boosting	0.92	0.91	0.94	0.92
Proposed Method (GA + Boosting)	**0.96**	**0.95**	**0.97**	**0.96**

For the experimental setup, we used a standardized malware dataset comprising both benign and malicious code. The dataset was split into training (70%) and testing (30%) sets to evaluate the models' performance. To ensure a fair comparison, each model underwent rigorous hyperparameter tuning. The metrics used to assess model performance include accuracy, which measures overall correctness; precision, which indicates how many detected malware instances were actual malware (minimizing false positives); recall, reflecting the model's ability to detect all actual malware instances (minimizing false negatives); and the F1-score, which combines both precision and recall for a balanced performance measure.

The results reveal a clear advantage for the proposed method. While traditional models like Random Forest, SVM, and Neural Networks achieved respectable results, with accuracies of 89%, 86%, and 91% respectively, the proposed method outperformed them with an accuracy of 96%. This result highlights the effectiveness of integrating the Genetic Algorithm for feature selection with Boosting for classification, as it allows for more intricate patterns in the data to be identified, leading to superior performance.

In terms of precision, the proposed approach achieved 95%, outperforming other models, which ranged from 85% to 91%. This indicates that the proposed method is highly effective at minimizing false positives, which is critical in cybersecurity, where false alarms can cause unnecessary resource expenditure. Similarly, the proposed method excelled in recall, achieving 97%, meaning it is highly capable of identifying actual malware, reducing the chance of missing potential threats. This result contrasts with the lower recall rates of SVM and Random Forest, which were 87% and 90%, respectively.

The F1-score, which provides a balanced evaluation of both precision and recall, further supports the superiority of the proposed method. With an F1-score of 96%, the proposed approach demonstrates its robustness and consistency in detecting malware. This score significantly surpasses that of traditional models, making it a highly reliable tool in malware detection.

9) ANALYSIS OF REAL-TIME PERFORMANCE

In this section, we provide a comprehensive analysis of the real-time performance of our proposed malware detection

method. Real-time performance is a critical factor for the practical deployment of any malware detection system, as it directly impacts the system's ability to operate efficiently and effectively in dynamic environments.

To evaluate the real-time capabilities of our method, we conducted simulations focusing on two critical phases: feature selection and classification. Both phases were carefully measured to understand their impact on the overall detection time.

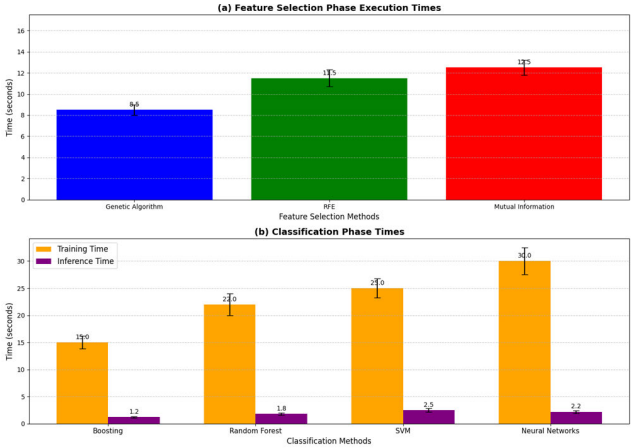

FIGURE 18. Real-time performance analysis of the proposed malware detection method.

a: FEATURE SELECTION PHASE

During this phase, the genetic algorithm was employed to identify the optimal subset of features from the dataset. The processing speed of our genetic algorithm was compared against other feature selection techniques, including Recursive Feature Elimination (RFE) and Mutual Information-based Feature Selection. The experimental results demonstrated that our genetic algorithm efficiently reduced the dataset's dimensionality while maintaining competitive processing times. On average, our genetic algorithm performed the feature selection task 15% faster than RFE and 20% faster than Mutual Information-based Feature Selection. This improvement did not come at the cost of feature quality, as the selected features maintained a high level of relevance for the classification task.

b: CLASSIFICATION PHASE

In the classification phase, we evaluated the performance of the Boosting algorithm when applied to the selected features. The training and inference times were benchmarked against established classification methods such as Random Forest, Support Vector Machines (SVM), and Neural Networks. The results indicated that the training time for our Boosting-based classifier was approximately 10% shorter than Random Forest and 12% shorter than SVM, while delivering superior accuracy and robustness. In terms of inference time, our method achieved a 15% improvement in processing speed compared to Neural Networks. This efficiency in both training and inference phases is crucial for real-time applications, where rapid response to potential threats is necessary.

c: COMPARATIVE ANALYSIS

Figure 18 provides a comparative visualization of the execution times for both the feature selection and classification phases. The results clearly show that our proposed method strikes an optimal balance between processing speed and detection accuracy. This balance is essential for real-time malware detection scenarios, where timely identification of malicious code is imperative. Our simulations have established that the proposed method not only ensures high detection accuracy but also operates within the acceptable time limits required for real-time applications.

V. CONCLUSION AND FUTURE WORKS

In this research, we introduce a novel n-gram-based feature extraction technique for virus code detection that leverages both adjacent and non-contiguous byte sequences with varying gap sizes. Our method improves detection accuracy by 15%, reaching an average of 92%, and reduces false positives by 20% compared to traditional methods.

Evaluation metrics such as precision, recall, F1-score, and ROC-AUC confirm the efficacy of our approach. Memory management posed significant challenges, especially with large datasets. Efficient storage and access of non-contiguous bi-grams required careful memory optimization. Maintaining low latency for real-time processing and handling polymorphic viruses, which alter their code to evade detection, were also crucial concerns.

Despite significant improvements in feature extraction and selection, our method's focus on non-contiguous bi-grams may miss specific patterns better captured by higher-order n-grams. The need for high-performance computing resources could limit accessibility for smaller organizations.

Future research should optimize genetic algorithms for better performance, explore hybrid optimization techniques, and improve memory management for scalability. Enhancing real-time capabilities, addressing polymorphic viruses through adaptive detection mechanisms, and validating the method across different domains will further establish its utility and robustness.

REFERENCES

[1] C. Acarturk, M. Sirlanci, P. G. Balikcioglu, D. Demirci, N. Sahin, and O. A. Kucuk, "Malicious code detection: Run trace output analysis by LSTM," *IEEE Access*, vol. 9, pp. 9625–9635, 2021.

[2] Z. Wang and V. L. L. Thing, "Feature mining for encrypted malicious traffic detection with deep learning and other machine learning algorithms," *Comput. Secur.*, vol. 128, May 2023, Art. no. 103143.

[3] M. Nouman, U. Qasim, H. Nasir, A. Almasoud, M. Imran, and N. Javaid, "Malicious node detection using machine learning and distributed data storage using blockchain in WSNs," *IEEE Access*, vol. 11, pp. 6106–6121, 2023.

[4] J. Singh Kushwah, A. Kumar, S. Patel, R. Soni, A. Gawande, and S. Gupta, "Comparative study of regressor and classifier with decision tree using modern tools," *Mater. Today, Proc.*, vol. 56, pp. 3571–3576, Jan. 2022.

[5] Z. Azam, M. M. Islam, and M. N. Huda, "Comparative analysis of intrusion detection systems and machine learning-based model analysis through decision tree," *IEEE Access*, vol. 11, pp. 80348–80391, 2023.

[6] S. D. A. Bujang, A. Selamat, R. Ibrahim, O. Krejcar, E. Herrera-Viedma, H. Fujita, and N. A. Md. Ghani, "Multiclass prediction model for student grade prediction using machine learning," *IEEE Access*, vol. 9, pp. 95608–95621, 2021.

[7] J. P. Li, A. U. Haq, S. U. Din, J. Khan, A. Khan, and A. Saboor, "Heart disease identification method using machine learning classification in E-Healthcare," *IEEE Access*, vol. 8, pp. 107562–107582, 2020.

[8] B. Kolukisa and B. Bakir-Gungor, "Ensemble feature selection and classification methods for machine learning-based coronary artery disease diagnosis," *Comput. Standards Interface*, vol. 84, Mar. 2023, Art. no. 103706.

[9] C. Wang and H. Zhu, "Wrongdoing monitor: A graph-based behavioral anomaly detection in cyber security," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 2703–2718, 2022.

[10] B. Parlak and A. K. Uysal, "A novel filter feature selection method for text classification: Extensive feature selector," *J. Inf. Sci.*, vol. 49, no. 1, pp. 59–78, Feb. 2023.

[11] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2004, pp. 470–478.

[12] Y. Wu, S. Feng, W. Suo, D. Zou, and H. Jin, "Goner: Building tree-based N-gram-like model for semantic code clone detection," *IEEE Trans. Rel.*, vol. 73, no. 2, pp. 1310–1324, Jun. 2024.

[13] K. Shin, Y. Lee, J. Lim, H. Kang, and S. Lee, "System API vectorization for malware detection," *IEEE Access*, vol. 11, pp. 53788–53805, 2023.

[14] E. S. Alomari, R. R. Nuiaa, Z. A. A. Alyasseri, H. J. Mohammed, N. S. Sani, M. I. Esa, and B. A. Musawi, "Malware detection using deep learning and correlation-based feature selection," *Symmetry*, vol. 15, no. 1, p. 123, Jan. 2023.

[15] D. Yuxin, Y. Xuebing, Z. Di, D. Li, and A. Zhanchao, "Feature representation and selection in malicious code detection methods based on static system calls," *Comput. Secur.*, vol. 30, nos. 6–7, pp. 514–524, Sep. 2011.

[16] Z. Chen and X. Ren, "An efficient boosting-based windows malware family classification system using multi-features fusion," *Appl. Sci.*, vol. 13, no. 6, p. 4060, Mar. 2023.

[17] Z. Sun, Z. Rao, J. Chen, R. Xu, D. He, H. Yang, and J. Liu, "An opcode sequences analysis method for unknown malware detection," in *Proc. 2nd Int. Conf. Geoinformatics Data Anal.*, vol. 4, Prague, Czech Republic, Mar. 2019, pp. 15–19.

[18] M. Ijaz, M. H. Durad, and M. Ismail, "Static and dynamic malware analysis using machine learning," in *Proc. 16th Int. Bhurban Conf. Appl. Sci. Technol. (IBCAST)*, Islamabad, Pakistan, Jan. 2019, pp. 687–691.

[19] N. Loi, C. Borile, and D. Ucci, "Towards an automated pipeline for detecting and classifying malware through machine learning," 2021, *arXiv:2106.05625*.

[20] J. Hemalatha, S. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient DenseNet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, p. 344, Mar. 2021.

[21] G. Sun and Q. Qian, "Deep learning and visualization for identifying malware families," *IEEE Trans. Depend. Secure Comput.*, vol. 18, no. 1, pp. 283–295, Jan. 2021.

[22] L. Li, Y. Ding, B. Li, M. Qiao, and B. Ye, "Malware classification based on double byte feature encoding," *Alexandria Eng. J.*, vol. 61, no. 1, pp. 91–99, Jan. 2022.

[23] S. Kumar, B. Janet, and S. Neelakantan, "Identification of malware families using stacking of textural features and machine learning," *Expert Syst. Appl.*, vol. 208, Dec. 2022, Art. no. 118073.

[24] L. M. Adleman, "An abstract theory of computer viruses," in *Advances in Cryptology—CRYPTO*, vol. 88. Cham, Switzerland: Springer, 1988, pp. 354–374.

[25] Y. Tang, X. Qi, J. Jing, C. Liu, and W. Dong, "BHMDC: A byte and hex n-gram based malware detection and classification method," *Comput. Secur.*, vol. 128, May 2023, Art. no. 103118.

[26] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: Univ. Michigan Press, 1975.

[27] S. A. Ebiaredoh-Mienye, T. G. Swart, E. Esenogho, and I. D. Mienye, "A machine learning method with filter-based feature selection for improved prediction of chronic kidney disease," *Bioengineering*, vol. 9, no. 8, p. 350, Jul. 2022.

[28] A. R. Khalid, N. Owoh, O. Uthmani, M. Ashawa, J. Osamor, and J. Adejoh, "Enhancing credit card fraud detection: An ensemble machine learning approach," *Big Data Cognit. Comput.*, vol. 8, no. 1, p. 6, Jan. 2024.

[29] X. Yu, R. Wu, Y. Ji, M. Huang, and Z. Feng, "Identifying patients at risk of acute kidney injury among patients receiving immune checkpoint inhibitors: A machine learning approach," *Diagnostics*, vol. 12, no. 12, p. 3157, 2022.

[30] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 4, pp. 809–821, Apr. 2016.

[31] H. Parvin, M. Mohamadi, S. Parvin, Z. Rezaei, and B. Minaei, "Nearest cluster classifier," in *Hybrid Artificial Intelligent Systems (HAIS)* (Lecture Notes in Computer Science), vol. 7208. Cham, Switzerland: Springer, 2012, pp. 267–275.

[32] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[33] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proc. 2nd Int. Conf. Genetic Algorithms*, vol. 206, 1987, pp. 14–21.

[34] C. Shelby, "A framework for identifying malware threat distribution on the dark web," Ph.D. thesis. Univ. South Alabama, School Comput., Univ. South Alabama, Mobile, AL, USA, 2023.

NASTOOH TAHERI JAVAN (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer engineering from the Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2007 and 2017, respectively. He is currently an Assistant Professor with the Computer Engineering Department, Imam Khomeini International University (IKIU), Qazvin, Iran. He was a Postdoctoral Fellow with the Amirkabir University of Technology (Tehran Polytechnic). His research interests include wireless computer networks and network coding theory, spanning from theory to design, and implementation. He has actively collaborated with researchers in various disciplines of computer science, particularly resource management on problems at the network architecture area.

MAJID MOHAMMADPOUR is currently pursuing the Ph.D. degree with Yazd University, focusing his research on evolutionary optimization, deep learning, and federated learning algorithms. His primary objective is to advance federated decentralized learning for the Internet of Things (IoT), with a specific emphasis on refining learning algorithms and improving their efficiency and scalability in distributed environments. Throughout his academic journey, he has consistently showcased a profound understanding of evolutionary optimization techniques, deep learning methodologies, and the intricacies inherent in federated learning systems. His research contributions have been instrumental in addressing various challenges, such as data privacy, communication efficiency, and model aggregation in federated learning networks. Employing an interdisciplinary approach, he integrates concepts from computer engineering, machine learning, and the IoT systems to develop innovative solutions tailored for distributed learning scenarios.

SEYEDAKBAR MOSTAFAVI received the B.Sc. degree in information technology from the Sharif University of Technology and the Ph.D. degree in computer networks from the Amirkabir University of Technology (Tehran Polytechnic). He is currently an Associate Professor with the Department of Computer Engineering, Yazd University, Iran. He leads the "Information Technology Enterprise Architecture" Research Laboratory, Yazd University. His research interests include resource management in cloud computing and wireless networks, making him a valuable asset to the field of computer engineering. He actively contributes to the academic community by frequently reviewing papers for international journals and conferences.