Article

Enhancing Smart IoT Malware Detection: A GhostNet-based Hybrid Approach

Abdulwahab Ali Almazroi [1,*] and Nasir Ayub [2]

[1] Department of Information Technology, College of Computing and Information Technology at Khulais, University of Jeddah, Jeddah 21959, Saudi Arabia
[2] Department of Creative Technologies, Air University Islamabad, Islamabad 44000, Pakistan; nasir.ayubse@gmail.com
* Correspondence: aalmazroi@uj.edu.sa

Abstract: The Internet of Things (IoT) constitutes the foundation of a deeply interconnected society in which objects communicate through the Internet. This innovation, coupled with 5G and artificial intelligence (AI), finds application in diverse sectors like smart cities and advanced manufacturing. With increasing IoT adoption comes heightened vulnerabilities, prompting research into identifying IoT malware. While existing models excel at spotting known malicious code, detecting new and modified malware presents challenges. This paper presents a novel six-step framework. It begins with eight malware attack datasets as input, followed by insights from Exploratory Data Analysis (EDA). Feature engineering includes scaling, One-Hot Encoding, target variable analysis, feature importance using MDI and XGBoost, and clustering with K-Means and PCA. Our GhostNet ensemble, combined with the Gated Recurrent Unit Ensembler (GNGRUE), is trained on these datasets and fine-tuned using the Jaya Algorithm (JA) to identify and categorize malware. The tuned GNGRUE-JA is tested on malware datasets. A comprehensive comparison with existing models encompasses performance, evaluation criteria, time complexity, and statistical analysis. Our proposed model demonstrates superior performance through extensive simulations, outperforming existing methods by around 15% across metrics like AUC, accuracy, recall, and hamming loss, with a 10% reduction in time complexity. These results emphasize the significance of our study's outcomes, particularly in achieving cost-effective solutions for detecting eight malware strains.

Keywords: Internet of Things; deep learning; malware detection; optimization methods; classification; GhostNet; Jaya Algorithm

check for updates

Citation: Almazroi, A.A.; Ayub, N. Enhancing Smart IoT Malware Detection: A GhostNet-based Hybrid Approach. *Systems* **2023**, *11*, 547. https://doi.org/10.3390/systems11110547

Academic Editors: Tetiana Hovorushchenko, Ivan Izonin and Hakan Kutucu

Received: 18 September 2023
Revised: 3 November 2023
Accepted: 8 November 2023
Published: 11 November 2023

1. Introduction

Presently, advanced technologies like massive data analytics, Artificial Intelligence (AI), Immersive Virtual Environments (IVE), and the Internet of Things (IoT) have evolved into integral elements of the Fourth Industrial Revolution. These technologies are now integrated into various fields, and they have had a significant impact [1]. As the IoT market expands, the devices, systems, and applications within it influence industries and reshape our daily lives. In the Internet of Things (IoT) realm, devices connect and share information, enhancing convenience in our lives. Nonetheless, this high degree of interconnectedness also increases cybersecurity risks, such as Distributed Denial of Service (DDoS) assaults that flood networks and malevolent crypto-mining that seizes resources for digital currency. These threats have been increasing rapidly [1–4].

Adding to the challenge, some manufacturers quickly produce and distribute IoT devices without strong security measures. This can lead to vulnerable devices entering the market, catching the interest of malware creators. When these devices are compromised, they can expose personal data and spread malware to more extensive networks [5,6].

Kaspersky Lab, a cybersecurity company, reported that in 2018, they found 121,588 different types of IoT malware, more than three times the 32,614 they found in 2021. They discovered over 120,000 variations of intelligent malware [7].

The convenience of IoT comes with security concerns. Malware, a type of harmful software, can infiltrate computer systems, networks, and devices, causing significant damage. As these attacks become more advanced, they become harder to stop using traditional security methods. IoT devices, in particular, are at risk due to their limited ability to defend against attacks [8]. Such attacks can compromise device security, leading to data breaches, financial losses, and other serious consequences. Given these challenges, researchers are focusing on finding ways to detect and classify malware in IoT systems. One approach is using signatures, where known malicious code patterns are compared to what is happening in the system. Another way is by observing the behavior of programs and identifying anything unusual. Finding the optimal solution to safeguard IoT devices is essential since both approaches have advantages and disadvantages.

The expansion of the Internet of Things (IoT) has enabled the connectivity of devices, but it has also sparked an increase in DDoS assaults and cryptocurrency mining [9]. Some IoT devices lack proper security, making them easy targets for malware [10,11]. Kaspersky Lab found over 121,000 types of IoT malware in 2018, a sharp increase from 2017 [12]. Researchers are using machine learning to combat this. By training models on malware and safe programs, machine learning can identify both known and unknown malware, adapting to evolving threats. However, challenges include the need for diverse malware samples and susceptibility to manipulation. Deep learning techniques, like convolutional and recurrent neural networks, show promise in recognizing patterns in malware but require substantial resources [9,12].

To detect and categorize IoT malware, researchers conduct studies using feature learning and classification [3,13]. Analysis phases are divided into static, dynamic, hybrid, and memory analyses [10,14]. Detection methods include specification-based, signature-based, cloud-based techniques and heuristic-based [15,16]. These techniques can prevent malware from spreading to other IoT devices. Nevertheless, IoT devices often feature constrained hardware resources, making promptly identifying emerging malware threats challenging. Researchers are now developing novel models by applying machine learning techniques and enhancing their capabilities. A complete approach to malware detection is provided by hybrid models that utilize signature-driven and behavior-based techniques [17]. These models integrate machine learning with other techniques and employ feature selection to enhance efficiency. IoT devices face evolving malware threats; researchers continue to explore methods to protect these devices.

This research paper introduces an ensembler model designed to detect and classify malware on Internet of Things (IoT) devices. The proposed model combines the capabilities of GhostNet Gated Recurrent Unit (GRU) and an optimization technique, Jaya Algorithm (JA), resulting in a more efficient and accurate malware detection and classification process. The model's methodology involves extracting crucial features from malware samples using the Mean Decrease Impurity (MDI) sequences. These derived features are input into the machine learning models to enable additional feature extraction and examination. The JA is employed to optimize the model's performance when dealing with a huge size of data. This algorithm aids in pinpointing the optimal parameter settings, thereby elevating the model's effectiveness by reducing computational intricacy.

The primary achievements of this study include the following:

1. We investigate eight big datasets encompassing malware and benign instances, essential insights to train our model effectively: Gagfyt, Hide and Seek, Kenjiro, Linux Hajime, Mirai, Muhstik, Okiru, and Tori.
2. By adeptly utilizing feature engineering techniques like precise feature scaling, extensive one-hot encoding, and insightful feature importance analysis employing MDI and XGBoost, we empower our model to outperform in identifying intricate malware behaviors.

3. We proposed a pioneering deep learning ensemble named GNGRUE, designed to effectively process and classify large datasets, thereby enhancing the efficiency of our approach.
4. The proposed approach, GNGRUE, enhances classification performance accuracy by 15%, accompanied by a 10% reduction in time complexity compared to existing approaches.
5. By tuning the parameters of GNGRUE through JA, our model can easily handle large amounts of data with the same accuracy and execution time.
6. Real-world Applicability: By successfully identifying and countering eight distinct malware strains, our research provides a valuable contribution to securing IoT systems. This impact resonates particularly in sectors like smart cities and advanced manufacturing.

The structure of this study is as follows: Section 2 provides a concise overview of previous studies in the malware identification and categorization field. Section 3 elaborates on the intended refined hybrid model, providing further insight into its essential elements and operational mechanisms. Moreover, Section 4 delineates the outcomes obtained from experiments and assesses the efficacy of the proposed ensembler. Finally, Section 5 presents the conclusion, summarizing the central findings and suggesting potential enhancements for future research.

2. Literature Review

Many researchers have devoted their efforts to categorizing and detecting malicious software, employing methods that cover dynamic, static, and approaches driven by artificial intelligence. This section offers an overview of diverse techniques utilized in classifying malicious software. Static evaluation methods are used to extract unchanging attributes like byte sequences, textual elements, and opcode sequences [16,17], as well as factors such as function length allocation [17], functional call graphs [18], and attributes of PE files [19]. A study by the investigator [20] involved the evaluation of various machine learning approaches utilizing static characteristics derived from binary files. Similarly, the author of [21] also implemented an indirect random forest technique based on stationary features to categorize malicious software. The identification of signature-based malware commonly entails procedures that involve analyzing malicious code, creating signatures, and storing them in databases. However, these strategies prove ineffective when dealing with zero-day malware frequently generated by malicious actors. Both automated processing and code obfuscation are susceptible to scrutiny through static analysis. Dynamic assessment methodologies capture behavioral components, such as system calls, network interactions, and sequences of instructions [22].

The author proposed a similarity-based approach in [23], which involves classifying malicious software. Moreover, Hidden Markov Models (HMMs) are employed to retrieve sequences of API calls, and similarity scores were computed to aid in categorizing malicious software. Their approach has prominent computational overhead and demonstrates heightened effectiveness when applied to a smaller dataset. Dynamic analysis is limited by the potential for changing malware behavior within virtual environments. Hybrid approaches have been developed to categorize malicious software, amalgamating attributes from unchanging and ever-changing assessment techniques and computational intelligence methods.

In [24], the author utilized a hybrid Support Vector Machine (SVM) to extract dynamic attributes associated with API calls and detect malicious software. Additionally, the study suggests that this hybrid approach outperforms the sole reliance on static or dynamic isolation techniques. More recently, visual-based techniques have gained considerable attention in research on analyzing malicious software [25]. For instance, researchers have represented sequences of opcodes and system calls as visual images [26]. Furthermore, [27] proposed an effective method for distinguishing packed and encrypted malicious software, generating graphical representations of malicious software binaries to categorize them.

Visual methodologies have proven instrumental in enhancing the precision of categorizing binary files and broadening the scope of extensively utilized techniques, as exemplified by the pioneering investigation in [28]. By harnessing visualizations grounded in bytes, researchers swiftly discern pivotal file components. The work detailed in [29] applied GIST texture attributes extracted from grayscale visual depictions, leveraging the K Nearest Neighbors (KNN) algorithm with Euclidean distance to classify malicious software. This methodology boasts lower computational overhead compared to the n-gram malware classification method. The author in [30] introduced an automated analytical approach that generates entropy graphs from grayscale images.

Nonetheless, their technique contends with elevated entropy measures and lacks pattern visualization, diminishing its efficacy in identifying packed malware. In the context of [31], the author extracted intensity, Gabor, and wavelet attributes from binary images through a technique impervious to code obfuscation. Furthermore, the author advocated using grayscale images and applying the local mean method to reduce image dimensions, enhancing ensembling capabilities.

A unique approach to studying malware activity was discovered during the investigation by [32]. This approach involves using API hooking to retrieve data from applications that use API calls and parameters. This technique decodes unique malware activities from the sequence of calls to the API and data that are collected. Incorporating machine learning approaches like SVM, Ensemble Methods (Random Forest) algorithms, and Tree-based Models to classify malware using inferred behaviors was one way to tackle the problem of precisely detecting malware. This challenge was attributed to the subjectivity between analysts and the behaviors they interpreted.

A novel method using treemaps and thread graphs to graphically summarize a large number of behavior reports from CWSandbox was presented by the author in [33]. Data visualization is made easier by treemaps, which provide insights into the frequency of executed processes and API requests. Conversely, the thread graph translates behavior data into sequential images, enumerating individual thread operations within processes over time.

Examining alterations in malware quality relative to legitimate software of specific categories, the investigation detailed in [34] applies conventional software quality measurements to extract diverse elements and uncover prevailing trends. Meanwhile, the author introduces a rapid technique for identifying variant malicious code, employing image processing technology to represent malware visually. In the initial stage, the binary malware file is transformed into a grayscale image, capitalizing on Convolutional Neural Networks (CNN) for autonomously deriving features from this visual representation. Furthermore, by employing the bat algorithm, a data equalization method is implemented on the malware image, effectively addressing overfitting concerns stemming from distinct malware families. This approach to detecting malware mainly boasts swift performance and a remarkable accuracy rate of 94.5%. Tackling the intricate task of identifying novel variations stands at the core of evolution detection, and this study harnesses the power of semi-supervised learning while embracing a wide variety of features. Table 1 shows the malware classification comparison study and the method used.

In response to the limits of signature-based techniques, researchers have moved their attention to behavior analysis and anomaly detection methods [9,35]. The necessity of utilizing machine learning in malware identification and intrusion detection is demonstrated by this alteration in strategy, which has resulted in the integration of machine learning algorithms into various network-level anomaly detection systems. These techniques include methodically evaluating network traffic data and collecting discriminative characteristics that discriminate between genuine and malicious traffic. After that, these traits are used to train classification algorithms to detect prospective assaults.

These classification models' outputs are frequently shown in binary form, classifying each instance of data as either normal or anomalous. Similar to this, a study [36] used feature selection sets derived from earlier studies in Android malware traffic identification to test five supervised machine learning techniques. RF, MLP, KNN, Naive Bayes (NB), and Decision Tree (J48) were the algorithms employed. Furthermore, utilizing feature selection sets from the stated approach, the experimental results showed that the Multi-Layer Perceptron (MLP) classifier outperformed all other classifiers, with an 83% accuracy rate, a 90% True Positive (TP) rate, and a 23% False Positive (FP) rate.

Table 1. Comparative Study of Malware Classification and Detection Methods.

Ref	Method	Key Characteristics	Differences	Limitations	Dataset
[14,15]	Malware Analysis and Detection	Static, Dynamic, Machine Learning Based Strategies	Varied approaches for malware categorization explored	Only specific malware is considered	muhstik, kenjiro
[16–19]	Static Assessment Techniques	Extracts byte, text, opcode sequences, function length distribution, functional call graph, PE file features	Focuses on static attributes for malware classification	Limited assessment of dynamic behaviors	MIRAI, okiru
[20,21]	Machine Learning Approaches	Utilizes static attributes for malware classification; Introduces oblique random forest technique	May face challenges in accurately detecting complex malware variants	Dependent on analysts' interpretation of behaviors	Binary file datasets
[22]	Dynamic Assessment Methods	Extracts behavioral elements (system calls, network interactions, instruction sequences)	Captures malware behavior in dynamic environments	Limited applicability to heavily obfuscated malware	Behavioral traces, system call datasets
[23]	Similarity-Based Classification	Uses Hidden Markov Models (HMMs) to retrieve API call sequences	This entails computational overhead, particularly with reduced data	Efficacy may decrease with highly polymorphic malware	API call sequence datasets
[24]	Hybrid Methodologies	Combines static and dynamic attributes with machine learning	Offers an integrated approach combining diverse analysis techniques	Resource-intensive due to combined methodologies	Various malware datasets
[25,26]	Support Vector Machine (SVM) Approach	Extracts dynamic API call attributes for identification; Utilizes hybrid approach for improved results	The hybrid approach may outperform standalone techniques	Requires substantial labeled data for training	Dynamic behavior traces, labeled datasets
[27–31]	Visual-Based Techniques	Represents opcodes and system calls as visual images	Utilizes visual representations for malware analysis	Limited capability for handling dynamic behaviors	Image datasets, malware binaries
[32]	Treemaps, Thread Graphs	Visualizes malware behaviors from behavior records; Condenses significant behavior record reports	Offers visual summarization of behavior data	Complexity may increase with a large volume of behavior records	Behavior
[33,34]	Variant Malicious Code Detection	Utilizes image processing for rapid detection; Applies grayscale image conversion CNN features extraction	Rapid detection approach with high accuracy	May struggle with highly encrypted or packed malware	Image datasets, malware binaries

Recurrent neural networks (RNNs) have been shown to be useful in identifying network traffic behavior by other researchers [37] who modeled the traffic as a series of evolving states over time. This method converted network traffic features into character sequences, allowing RNNs to learn their temporal characteristics. However, the study's findings suggested that RNN detection models found difficulties in dealing with traffic behaviors that required more clearly identifiable information and some occurrences of unbalanced network traffic.

Seven machine learning techniques were applied to the 24 attributes of the well-known Kyoto 2006+ dataset in a recent study [38]. Among these were NB, SVM, K-Means, Fuzzy C-Means (FCM), KNN, Radial Basis Function (RBF), and an ensemble methodology that included the six previously mentioned techniques. The results of the tests indicated that the majority of these learning algorithms generated very high accuracy rates, consistently above the 90% threshold.

In addition, a self-learning anomaly detection system based on Discriminative Restricted Boltzmann Machine (DRBM) neural networks was described in [39]. In its training model, this technique was unusual because it only used normal traffic data to dynamically generate the knowledge necessary for identifying anomalous traffic with amazing accuracy. Three experiments were conducted: one with the popular KDD'99 public dataset, one with compromised network host, and one utilizing real traffic traces from a healthy network host. The initial studies produced the most accurate findings, with 92% and 96% accuracy rates, respectively.

Another significant study [40] focused on anomaly identification in complicated network settings by decreasing noisy data associated with irrelevant attributes. A mix of machine learning techniques for feature selection was used. To discover feature clusters, the investigation began with the unsupervised k-means clustering approach. Subsequently, features were ranked using the Naive Bayes algorithm and the Kruskal-Wallis test, leading to the selection of the most pertinent qualities based on their rankings. In the last stage, the C4.5 selection tree algorithm looked at a few chosen attributes from the first phase. The findings of the experiment showed that a significant reduction in the number of characteristics improved the speed and accuracy of anomaly identification by lowering processing overhead.

After a comprehensive literature review, we propose an optimized solution for malware detection. The details of this proposed framework are expounded upon in Section 3.

3. Motivation and Problem Statement

Constant and dynamic assessment, machine learning, ensemble approaches, and the integration of massive data methodologies have historically been the focus of malware detection research [14–19]. However, with the increasing importance of artificial intelligence-driven malware detection, there is a need for fresh and innovative solutions. One promising approach is ensemble learning, where multiple classifiers are trained and selectively employed to enhance detection capabilities. Despite its widespread use, signature-based detection remains limited to known malware variations, while combining passive and active analysis methods requires significant time and effort. Addressing the class imbalance between benign and malicious instances, a well-established challenge in the literature [24–26], has proven to be an effective strategy.

This research introduces a pioneering methodology that combines static and dynamic analyses to evaluate malware's runtime behavior. Ensemble learning with the GhostNet classifier and the JA optimization technique substantially improves the efficiency and effectiveness of malware detection.

4. Proposed System Model

This study uses a range of eight malware datasets to provide a hybrid deep learning method for malware identification. The initial stage involves loading the dataset into DataFrames and consolidating all datasets with the target column (malicious or benign) to determine outcomes. Further data pattern investigations are conducted through Exploration Data Analysis (EDA).

Recognizing the potential overfitting issues associated with unbalanced datasets, we address them by considering the distribution of the target variable. Subsequently, the input data are transformed into a format suitable for deep learning processing, utilizing one-hot encoding feature engineering. Feature scaling is applied to normalize the data, aligning the range of data characteristics or independent variables. Crucial features are identified using Mean Decrease in Impurity (MDI) and XGBoost algorithms before incorporating them into the Zeek Analysis Tool (ZAT) DataFrame, facilitating the management of extensive data volumes. K-means and PCA algorithms are employed for data clustering, and the effectiveness of these clustering techniques is evaluated using the silhouette score metric. After clustering, the isolation forest method is utilized to identify anomalies or deviations in the dataset.

The dataset becomes suitable for machine learning analysis after completing these preprocessing steps. It is then divided into 25% training and 75% test data. Optimal GNGRU parameters, enhancing classification, are determined using the JA optimization technique. For a comprehensive overview of the proposed model's workflow, refer to Figure 1.

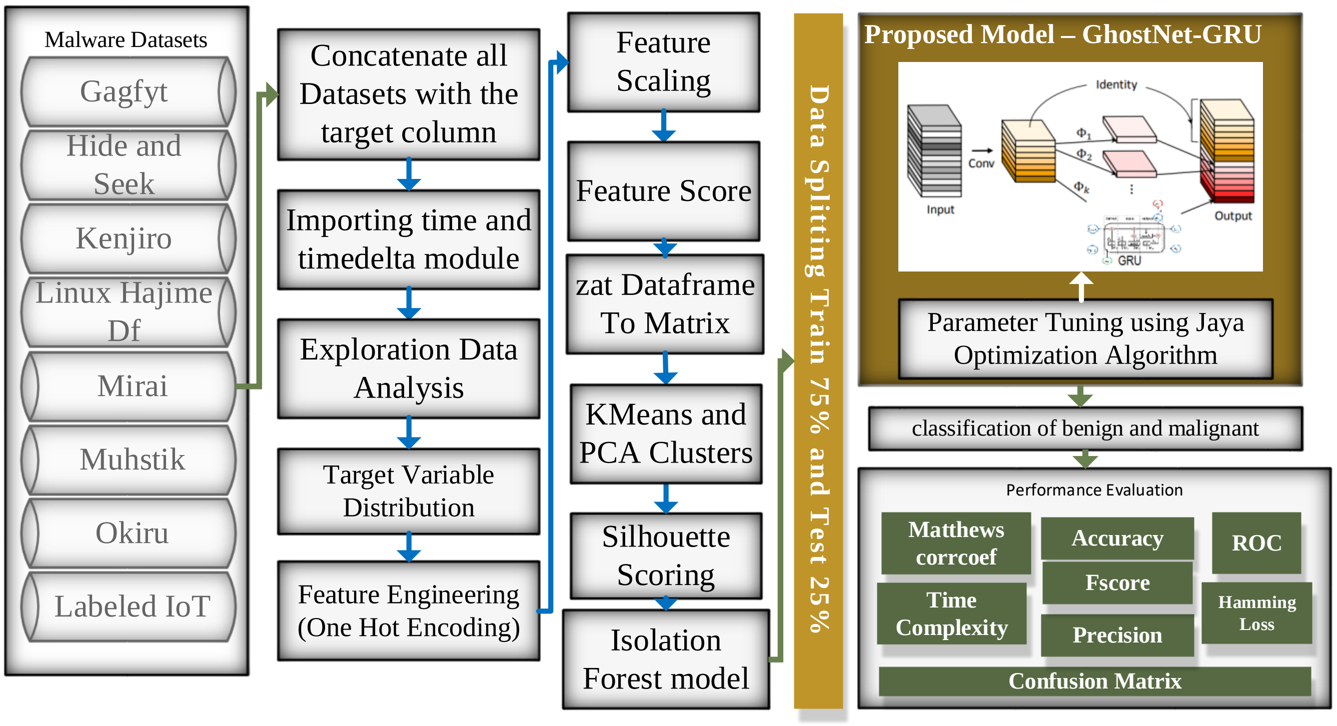


Figure 1. Detailed Flowchart of the Proposed System Model.

4.1. Dataset

The dataset utilized in this study is the IoT 23 dataset, which contains network traffic originating from IoT devices. This dataset comprises twenty distinct scenarios, each showcasing a variety of network traffic situations. These scenarios encompass both benign traffic from IoT devices and instances involving IoT devices infected with malware [41]. The dataset can be categorized into three subsets: authentic network traffic from typical IoT devices and network captures (in pcap files) depicting infected devices. Due to the dynamic nature of malware activities and the substantial traffic generated during infections, the pcap files are rotated every 24 h. However, in some cases, specific pcap files had to be extended beyond 24 h due to their extensive growth, resulting in variations in capture durations.

Table 2 shows additional situations in the IoT 23 dataset. It contains the scenario ID, dataset name, packet count, duration (in hours), Zeek ID flows, pcap file size, and malware sample names utilized to infect the devices. This knowledge is critical for comprehending the many circumstances inside the dataset and its benign and harmful properties. Due to the wide collection of network traffic events, the IoT 23 dataset is a great resource for academics, educators, and developers trying to improve machine learning algorithms and threat detection models.

Table 2. Summary of IoT 23 Dataset Scenarios and Associated Malware Samples.

SNo	Name	Packets	Duration (h)	Zeek Flows	Pcap Size	Name of Dataset
1	Gagfyt	217,000	12	35810	2. 8 MB	CTU-IOT 60-1
2	Hide and Seek	1,786,000	112	1008749	173 MB	CTU-IOT 1-1
3	Kenjiro	50,000	24	54659	433 MB	CTU-IOT 17-1
4	Linux Hajime	637,000	24	6378294	8.33 MB	CTU-IOT 9-1
5	Mirai	130,000	24	3394346	1.34 MB	CTU-IOT 48-1
6	Muhstik	496,000	36	156104	50 MB	CTU-IOT 3-1
7	Okiru	1,300,000	24	1364513	20 MB	CTU-IOT 3-1
8	Tori	50,000	24	3288	3.9 MB	CTU-IOT 3-2

4.2. PreProcessing

We used several data analysis and preparation techniques before applying classification methods. First, we combined the target column with the entire dataset in a single DataFrame. The preprocessing processes were completed in the following order, which we will review in more detail and are shown visually in Figure 2.

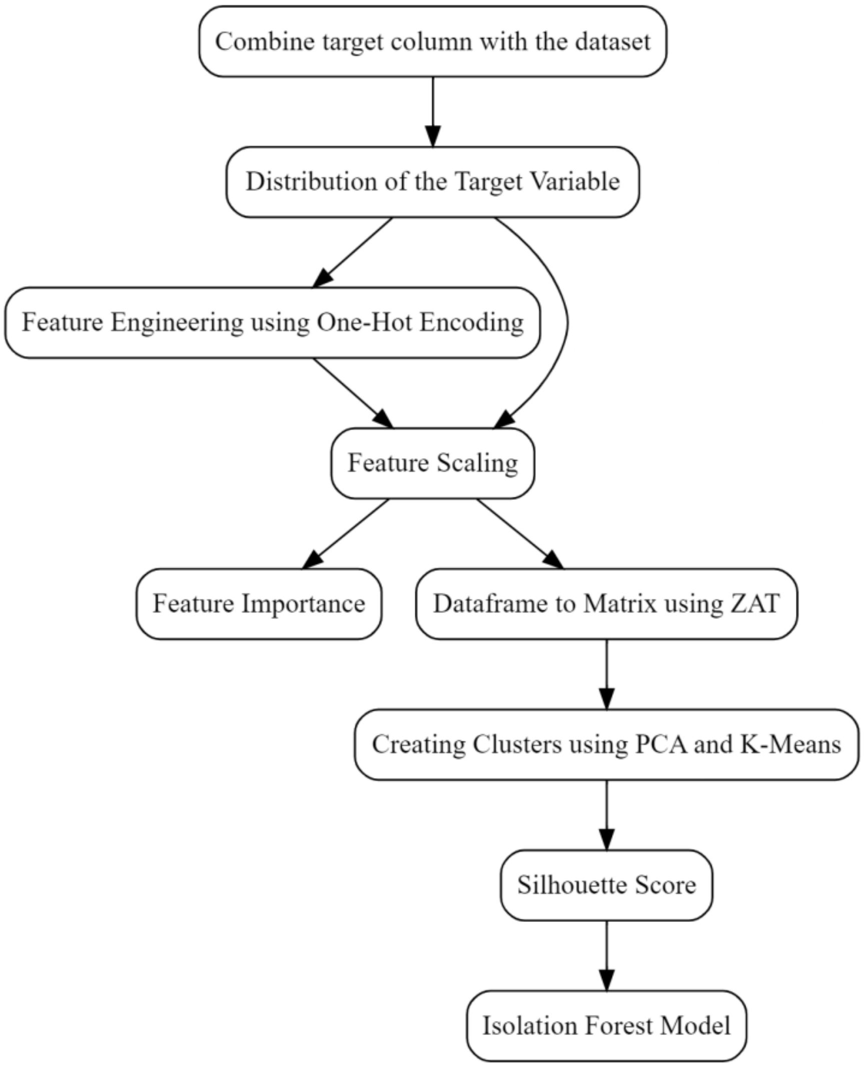

Combine target column with the dataset
Distribution of the Target Variable
Feature Engineering using One-Hot Encoding
Feature Scaling
Feature Importance
Dataframe to Matrix using ZAT
Creating Clusters using PCA and K-Means
Silhouette Score
Isolation Forest Model

Figure 2. Preprocessing workflow.

Distribution of the target variable: A binary variable that indicates whether a sample is malicious (1) or benign (0) is commonly used as the target variable in malware detection. The distribution of the target variable may have a major impact on how well the machine learning model used for detection performs [42].

A model trained on such data may be biased towards the majority class if the distribution of the target variable is noticeably unbalanced, with many benign instances and a few malicious ones (or vice versa). For instance, a simple model that consistently predicts benign would attain a 95% accuracy rate if 95% of the samples are benign and only 5% are malicious. However, it will not reveal any potentially harmful materials.

Several statistics, such as the proportion of malicious samples, the number of instances within every class, and the average and variance of the variable of interest, can be used to quantify the distribution of the target variable. The following mathematical formulation of this is found in Equation (1):

$$\mathrm{p} = \mathrm{n\_malicious}/(\mathrm{n\_malicious} + \mathrm{n\_benign}) \tag{1}$$

n malicious and n benign are the number of malicious and benign samples, respectively, if p is the fraction of harmful samples. As an alternative, the mean and variance of the target variable can be determined using Equations (2) and (3) as follows [42]:

$$\mathrm{mean} = (\mathrm{n\_malicious} \times 1 + \mathrm{n\_benign} \times 0)/(\mathrm{n\_malicious} + \mathrm{n\_benign}) \tag{2}$$

$$\mathrm{variance} = (\mathrm{n\_malicious} \times (1 - \mathrm{mean})^2 + \mathrm{n\_benign} \times (0 - \mathrm{mean})^2)/(\mathrm{n\_malicious} + \mathrm{n\_benign}) \tag{3}$$

Knowing the target variable's mean and variance is crucial to identifying malware. The mean, corresponding to the fraction of malicious samples, shows the target variable's average value. Conversely, the variance gauges how much the target variable is spread out from the mean value. These data may be used to assess how well machine learning models identify malware and to better understand the distribution of the target variable.

Feature Engineering using One-Hot Encoding: Each malware sample in our dataset is assigned to one of several malware classes. An 8-bit vector with one-hot encoding, where each location corresponds to a potential malware kind, is one approach to represent each form of malware [43]. The virus types Gagfyt and Hide and Seek, for instance, can be expressed as [1, 0, 0, 0, 0, 0] and [0, 1, 0, 0, 0, 0, 0], respectively.

Using this method, each sample in our dataset may be represented by a one-hot encoded vector that shows the type of malware corresponding to. A sample known as Muhstik, for example, can be described as [0, 0, 0, 0, 1, 0, 0].

We can represent the additional samples in our dataset using the same method. After one-hot encoding of the malware labels, these vectors are put into predictive algorithms for malware identification and classification. These algorithms may then analyze the one-hot encoded vectors to find patterns and connections across malware strains. These data allow new malware samples to be categorized according to their kind.

Feature Scaling: The malware dataset includes samples classified by Gagfyt and Hide and Seek and tagged with one of the above-mentioned malware kinds. These characteristics can fall into various ranges, making comparing and analyzing them challenging. Feature scaling makes it possible for machine learning techniques to obtain additional information from the data by standardizing the range of values for certain features [43]. We applied a normalization approach to scale the feature values to fall between 0 and 1 to standardize the feature values. By performing the following procedures, this approach can used to analyze each feature in the dataset: We start by calculating the lowest and maximum values for each attribute across all malware samples. Then, for each dataset sample, we apply the normalization equation to each feature value as [43] provides in Equation (4):

$$x' = \frac{x - x^{min}}{x^{max} - x^{min}} \tag{4}$$

Let x symbolize the initial value of an attribute, xmin represents the dataset's minimum value for that attribute, xmax denotes the dataset's maximum value for the attribute, and x′ signifies the standardized value of the attribute. We can use feature scaling to ensure all features are placed on a comparable scale, facilitating more precise comparisons and analyses. This, in turn, can result in enhanced performance and increased accuracy when employing machine learning algorithms on the dataset.

Consider an instance involving the malware type Gagfyt, where the properties are Gagfyt = 100 and Hide and Seek = 50. The normalization equation can be independently applied to each value if the dataset's minimum and maximum values for the Gagfyt feature are 50 and 500 and for the Hide and Seek feature are 20 and 200, respectively.

Normalized Gagfyt value: $Gagfyt' = \frac{100-50}{500-50}$, the output is 0.1875. This means that this sample's normalized value of the Gagfyt feature is 0.1875.

Normalized Hide and Seek score: 〖Hide and Seek〗^′ = (50-20)/(200-20), resulting in an output of 0.325. This signifies that this sample's normalized Hide and Seek feature value is 0.325.

Implementing feature normalization on our dataset of malicious samples can improve the effectiveness of machine learning models for identifying and categorizing malware. This technique ensures that diverse characteristics are harmonized onto a consistent scale. As a result, using feature scaling to the dataset aligns each feature's value, improving the effectiveness of algorithms that use machine learning for malware detection and classification.

Feature Importance: Comprehending feature significance is crucial to understanding how machine learning models make decisions. It lets us pinpoint the features or variables that substantially influence the model's predictions. In malware detection, grasping feature importance aids in recognizing and classifying crucial traits or indicators that contribute to identifying malware.

XGBoost, a prevalent machine learning algorithm renowned for its precision in constructing models for malware detection, employs the Mean Decrease Impurity (MDI) technique [44]. This technique assesses feature relevance by quantifying the overall reduction in impurities, such as entropy or the Gini index, achieved through data partitioning based on a specific feature. We explore the essential attributes for efficient malware identification and categorization by examining the MDI ratings assigned to each attribute.

The process of calculating feature significance through MDI in the development of an XGBoost model for malware detection involves the following steps:

- Train an XGBoost model using the training data.
- Access the feature_importance attribute of the trained model to obtain an array of feature importance scores.
- Rank the feature significance ratings in descending order to identify the most critical characteristics.

By analyzing the feature significance scores, we can pinpoint the paramount traits or indicators of malware that are pivotal for successful malware detection. With this information, we can refine our machine learning model to amplify its precision in recognizing malware. Prioritizing these critical malware indicators can significantly enhance the model's detection efficacy [44].

Furthermore, utilizing our model and enhancing efficiency becomes achievable by identifying and removing redundant or superfluous elements. This effective method improves the model's performance while lowering its complexity. Better outcomes are produced by a malware detection and classification framework that is more effective.

DataFrame-to-Matrix using ZAT: The Zero Access Tool (ZAT) is a Python package employing tools for analyzing and visualizing malware data. It supports a wide range of file formats and sources and is made to manage massive amounts of data. One of its characteristics is the DataFrame-to-Matrix methodology, which converts a DataFrame holding information about malware into a matrix representation [45]. Before transforming the data from a DataFrame to a matrix representation, data must first be transformed into a two-dimensional array of numerical values. Numerical characteristics can be scaled to ensure that their range and distribution are equivalent, and categorical data can be encoded using one-hot or label encoding techniques. Once the data are organized into a matrix, they may be fed into various machine learning algorithms for classification.

A DataFrame can be mathematically transformed into a matrix representation using Equation (4). Every column in Equation (4) denotes a feature, and every row is a sample. Let X be the initial DataFrame with n rows and m columns [45].

$$Y = [x^1, x^2, \ldots, x^n] \tag{5}$$

Let f^i be the i-th feature in x and let x′ be a matrix representation of y, with each row being a sample and each column representing a feature [28] in Equation (6).

$$y' = [f^1(e^1), f^2(e^1), \ldots, f^k(e\_1); f^1(e^2), f^2(e^2), \ldots, f^k(e^2); \ldots f^1(e^k), {}^{f2}(e^k), \ldots, f^m(e^k)] \quad (6)$$

In this case, the value given to the i-th feature for the jth sample is denoted by fi(yk). Depending on the characteristics of the features in the original DataFrame, several techniques may be used for encoding and scaling the data before they become a matrix representation.

Creating Clusters using PCA and K-Means: Malware detection techniques that employ clustering using KMeans and PCA may gather samples of malware with comparable traits. This method makes it easier to recognize patterns in the data. It can help identify several malware strains, including Gagfyt, Hide and Seek, Kenjiro, Linux Hajime, Mirai, Muhstik, Okiru, and Tori.

To employ KMeans clustering and PCA [46] for malware detection, a dataset of malware samples must first be preprocessed and feature-engineered to identify essential qualities that discriminate between various forms of malware. The dimensionality of the feature space is decreased by using PCA to isolate a smaller group of orthogonal axes that best capture the range of the data. The malware samples are then clustered based on their reduced feature representations using KMeans clustering.

With KMeans clustering, a dataset X containing n malware samples with m features is divided into k clusters C1, C2, …, Ck. Reducing the total squared distances between every malware sample and its designated centroid is how Equation (7) accomplishes this [46].

$$Minimize\sum_{i=1}^{k} = \sum_{x=0}^{k} x^j \in c^i \left|\left|X^j - \mu^i\right|\right|^2 \quad (7)$$

where ci is the collection of malware samples allocated to the cluster node's centroid, and PCA is demonstrated by the following:

A PCA, which aims to capture the most variability in the data, attempts to transform a dataset X of n malware samples with m characteristics into a new collection of k features. Equation (8) is what brings about the transition:

$$Y = XW \quad (8)$$

W is a matrix whose k most significant eigenvalues are identical to the k orthogonal eigenvectors of X's covariance matrix.

Gagfyt, Hide and Seek, Kenjiro, Linux Hajime, Mirai, Muhstik, Okiru, and Tori are a few examples of the various malware kinds that may be recognized once the malware samples have been clustered using KMeans and PCA. Threat intelligence, malware detection, and cybersecurity can all gain from it.

Silhouette Score: A clustering assessment method called Silhouette Scoring rates how well instances inside a cluster are categorized and how well clusters are generated. Using this method, the resulting clusters' quality may be evaluated. With a high Silhouette Score, linked malware samples have been effectively categorized, and the resulting clusters are distinctive [47]. The output quality of a clustering method is assessed in malware detection using Silhouette Scoring. Combining malware samples with similar characteristics or behavior simplifies spotting newly emerging malware strains. By grouping malware based on traits like Gagfyt, Hide and Seek, Kenjiro, Linux Hajime, Mirai, Muhstik, Okiru, and Tori, one may better understand the larger malware environment and identify upcoming risks.

In the context of malware detection, let X be a collection of feature vectors that act as samples of malware. Let c(i) represent the cluster to which instance i belongs, and let C represent the collection of clusters created as a consequence of applying a clustering procedure to the data in X. The silhouette coefficient, indicated as follows in Equation (9), can be determined for each instance i in X [47].

$$sc(k) = (v(k) - g(k))/maxg((k), v(k)) \quad (9)$$

The Silhouette Scoring tool measures how well instances are grouped inside a cluster and how well clusters are separated to assess the created clusters' quality. A high Silhouette Score denotes the effective grouping of relevant malware samples that produce distinctive clusters. Malware samples are classified by similarity using a clustering approach called Silhouette Scoring to detect new malware strains quickly.

Consider a set X of feature vectors representing malware samples to better understand the mathematical formula underlying the Silhouette Score in malware detection. Each instance i belongs to a particular cluster c(i) after the clustering method applied on X yields a set of clusters represented by C. Several distance metrics, including cosine similarity and Euclidean distance, may be used to calculate the Silhouette coefficient for each instance in X. The ratio between the average dissimilarity to all examples in other clusters, represented by b(i), and the average dissimilarity to all instances in the same cluster, represented by a(i), is used to calculate the Silhouette coefficient, for instance, i.

When an instance has a score of -1, it means that it is badly matched to its cluster and favorably matched to nearby clusters; a score of 0 shows that it is equally comparable to instances in both its own and surrounding clusters. The range of the silhouette coefficient is -1 to 1. As in Equation (10), the effectiveness of the clustering process is assessed using the average Silhouette coefficient of all instances in X [48].

$$S = (1/|X|) \times sum(s(i)) \quad (10)$$

where |X| represents the number of occurrences included in X.

When the silhouette score is high, the clusters are well separated, and the instances within each cluster are relatively similar. This demonstrates that the clustering method effectively combined similar malware samples into clusters and that the resulting clusters are distinctive for malware detection.

Isolation Forest Model: The isolation forest model is a machine learning technique that spots malware and anomalies across applications. Constructing random decision trees pinpoints abnormal data points, like virus samples. For malware detection, it is trained on feature sets like Gagfyt, Hide and Seek, Kenjiro, Linux Hajime, Mirai, Muhstik, Okiru, and Tori to unveil similar unusual behavior. To create the isolation forest model for malware detection [49], we define split points and randomly pick a feature subset from X. We select a feature and a random split point within its range for each split. Data are divided based on split points, repeating until each leaf node holds only one data point. The count of splits isolating a data point (h(x)) signifies its anomaly. Uncommon attributes in malware, needing fewer splits for isolation, result in lower h(x). The anomaly score can be computed using the equation [49] in Equation (11).

$$s(x) = 2^{(-E(h(x))/c(n))} \quad (11)$$

The isolation forest model employs key parameters: n for total data points, c(n) normalization, and E(h(x)) for average path length. Anomaly score, s(x), ranges from 0 to 1, with higher values indicating more irregularity. Once trained, the isolation forest identifies new, unusual behavior in malware samples. If a new sample significantly differs from known ones, it can be flagged as abnormal, indicating a potential new and severe threat.

4.3. Ensemble Classification Model

This article employs GNGRUE as the primary classifier and ensemble with the optimization method. Additionally, ML and DL state-of-the-art methods have been utilized to validate the ensemble method. Figure 3 describes the graphical representation of the optimized ensembler. A comprehensive description of the models is discussed below.

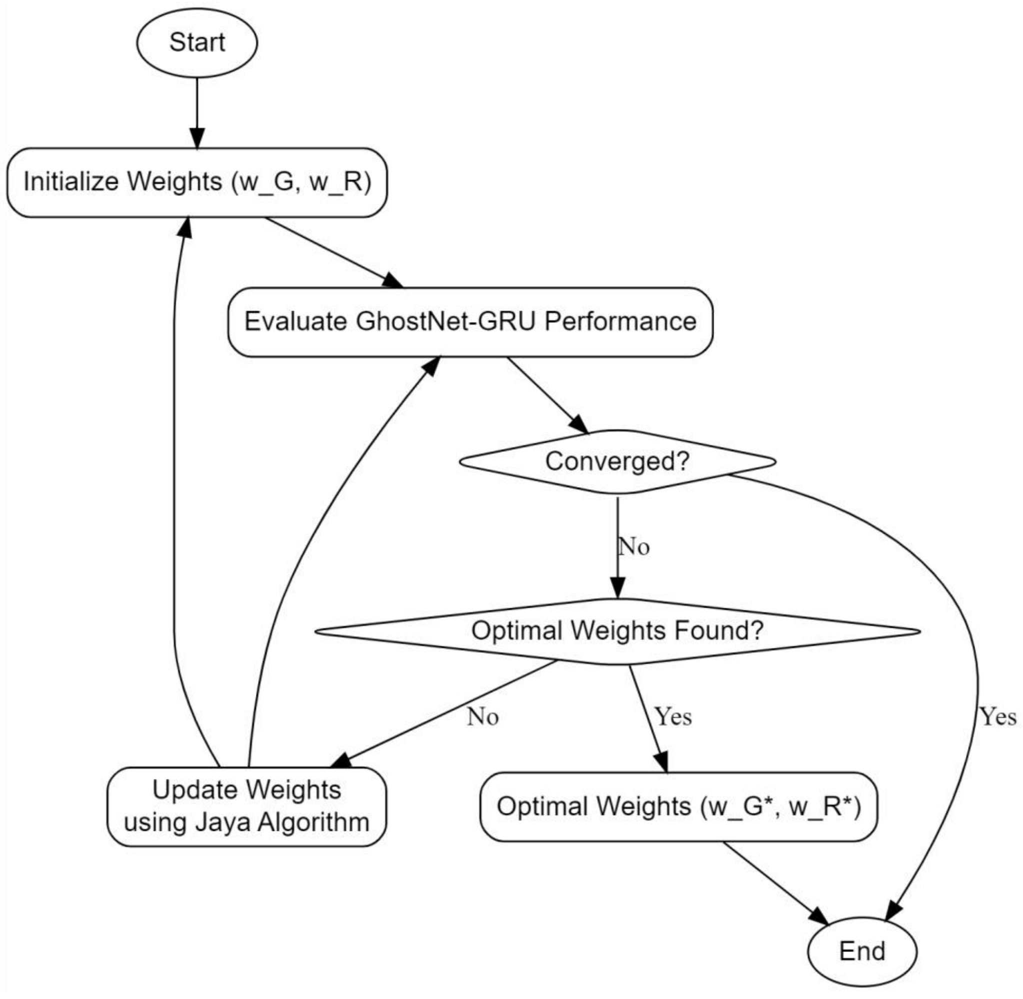

Start
Initialize Weights (w_G, w_R)
Evaluate GhostNet-GRU Performance
Converged?
No
Optimal Weights Found?
No
Yes
Yes
Update Weights using Jaya Algorithm
Optimal Weights (w_G*, w_R*)
End

Figure 3. Optimized ensembler graphical representation.

4.3.1. GNGRUE

GNGRUE contains two distinct yet complementary neural network components, each contributing unique capabilities to enhance the field of malware detection. GhostNet is a deep CNN framework that has been specifically designed to be efficient and accurate in classifying jobs. Designed with resource-constrained environments in mind [50], GhostNet achieves its compact size and excellent performance through an ingenious innovation known as the ghost module. This module introduces a parallel path alongside the primary convolutional layer, enabling more efficient feature extraction. By doing so, GhostNet optimizes computational complexity while preserving model accuracy, making it an ideal choice for scenarios with limited computing resources. However, GRUs belongs the recurrent neural network (RNN) family and are particularly good at processing sequential input, which is a critical need for malware detection. GRUs are highly effective at identifying complex patterns and temporal connections within sequences, which makes them indispensable for analyzing the subtle behaviors of software programs over time. In malware detection, this temporal understanding is pivotal in identifying malicious activities.

A combined approach combines the strengths of both GhostNet and GRUs to pursue robust malware detection. GhostNet is leveraged for its proficiency in malware feature extraction, efficiently processing the data of malware samples. The extracted features encode essential information about the malware characteristics that is then passed to the GRU component. The GRU component takes over, analyzing the sequential nature of malware behavior. It examines the temporal aspects, tracking how malware actions evolve. This is critical in identifying any abnormal or malicious patterns that might emerge during the execution of the software.

Our ensemble strategy involves harmonizing GhostNet's feature extraction with GRU's sequence analysis. To achieve this fusion of capabilities, we employ weighted averaging. We ensure a balanced contribution from both models by assigning weights wG and wR to the GhostNet and GRU predictions. The final combined prediction $C(x)$ for a given input x is determined using the Equation (12) [50]:

$$C(x) = wG \cdot G(x) + wR \cdot R(x) \tag{12}$$

This combined probability score, $C(x)$, offers a comprehensive perspective on the likelihood of x being malicious.

Optimization and Decision Making: To decide the nature of the input (malicious or benign), a decision threshold θ is set. If $C(x)$ is greater than or equal to θ, the input x is classified as malicious; otherwise, it is deemed benign. The weights wG and wR are fine-tuned during the training phase to optimize the ensemble's performance, ensuring a balance between GhostNet's feature extraction and GRU's sequence analysis capabilities.

4.3.2. Jaya Algorithm

The Jaya technique is a population-based optimization technique that was first presented by Dr. R. V. Rao [51]. Its primary purpose is to solve complicated optimization issues. The Jaya Algorithm is essential for adjusting the weights wG and wR in order to optimize the GNGRUE ensemble weights for malware detection and to maximize the ensemble's performance. We turn to the Jaya Algorithm for optimization to enhance the GNGRUE GhostNet-GRU ensemble's capabilities. The Jaya Algorithm operates on the principles of improvement and exploration.

Objective Function (Performance Metric): In optimizing the GNGRUE ensemble, we have an objective function, denoted as f(wG,wR), which quantifies the ensemble's performance. This function could represent metrics like accuracy, F1-score, or any other suitable measure.

Population Initialization: The algorithm starts with initializing a population of candidate solutions, each represented by weights (wG,wR).

Evaluation: Each candidate solution is evaluated using the objective function:

Objective function: f(wG,wR): This function measures how well the ensemble performs with the current weight combination.

Improvement and Exploration: The Jaya Algorithm classifies solutions into two categories. Improvement: A solution (wG′,wR′) is considered an improvement if it results in a lower objective function value, indicating better performance in Equation (13) [51]:

$$\mathrm{f(wG', wR') < f(wG, wR)} \tag{13}$$

Exploration: A solution (wG′,wR′) is categorized as an exploration if it does not lead to an improvement as in Equation (14):

$$\mathrm{f(wG', wR') \geq f(wG, wR)} \tag{14}$$

Population Update: The population is updated based on the principles of improvement and exploration. Specifically, solutions that lead to improvements are retained in Equation (15) [51]:

$$\mathrm{(wG, wR) = (wG', wR') if\ f(wG', wR') < f(wG, wR)} \tag{15}$$

Solutions resulting in explorations are replaced with new solutions in Equation (16):

$$\mathrm{(wG, wR) = GenerateNewSolution() if\ f(wG', wR') \geq f(wG, wR)} \tag{16}$$

The algorithm continues to iterate through these update steps to explore the solution space and refine the weight combinations.

Termination Criteria: These phases are repeated by the Jaya Algorithm until a termination criterion—such as a maximum number of iterations or an improvement threshold—is satisfied.

Optimal Weights: Once the algorithm terminates, the weights wG and wR associated with the best-performing solution in the population are considered optimal in Equation (17):

$$\text{Optimal Weights: } (wG{*}, wR{*}) \tag{17}$$

Ensemble Configuration: Finally, the GNGRUE is configured with the optimal weights (wG∗,wR∗), enabling it to accurately predict whether a given software sample is malicious or benign.

Figure 4 provides an overview of the iterative process by which the Jaya Algorithm fine-tunes the weights of the GNGRUE ensemble, seeking to maximize its performance in detecting malicious software. The algorithm combines principles of improvement and exploration to achieve optimal weight configurations for enhanced malware detection capabilities.

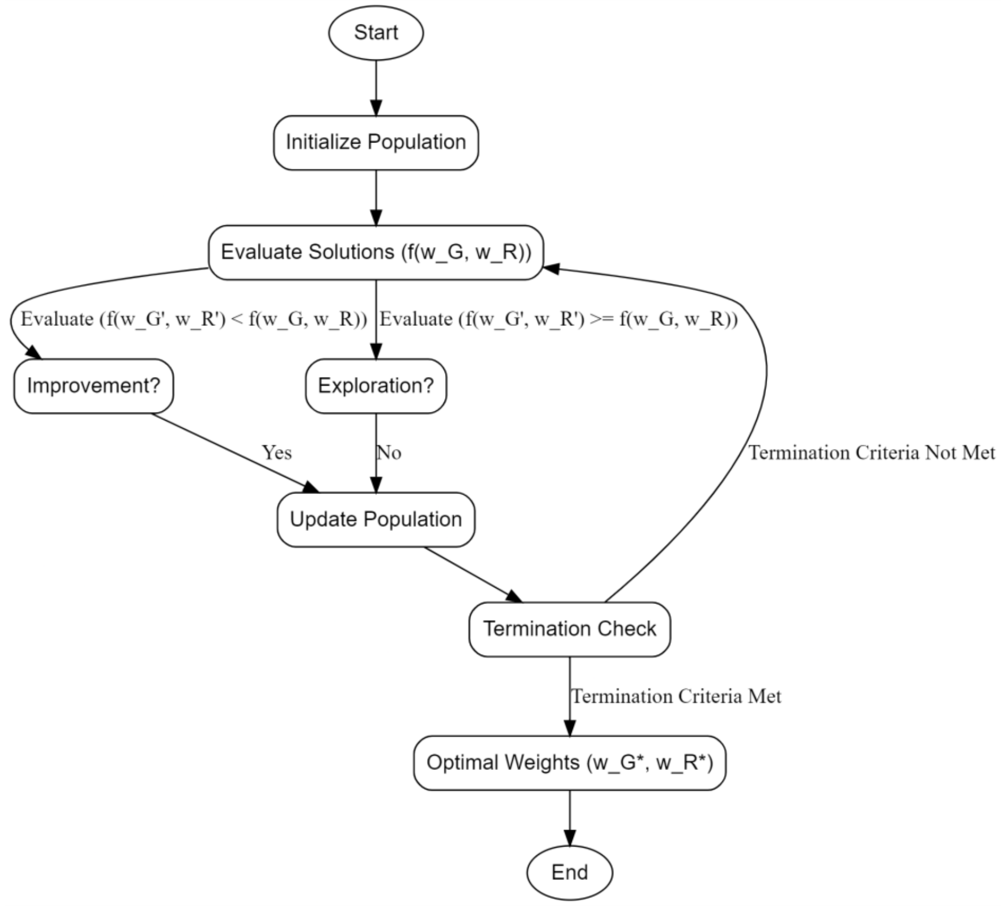

Start
Initialize Population
Evaluate Solutions (f(w_G, w_R))
Evaluate (f(w_G', w_R') < f(w_G, w_R))
Evaluate (f(w_G', w_R') >= f(w_G, w_R))
Improvement?
Exploration?
Yes
No
Termination Criteria Not Met
Update Population
Termination Check
Termination Criteria Met
Optimal Weights (w_G*, w_R*)
End

Figure 4. Jaya Algorithm in terms of optimizing the GNGRUE weights (flowchart).

5. Experimental Results

We used TensorFlow within the Google Colab environment in this part, utilizing its outstanding GPU resources for our malware detection system. Google Colab gives access to high-performance computing equipment: a multi-core CPU (Intel Xeon E5-2673 v4, 16 cores), 25 GB of system memory, a high-end GPU (NVIDIA Tesla T4 with 16 GB GDDR6 memory), and network connectivity of up to 50 Mbps. These criteria ensure we have the computational resources to conduct our study efficiently. Our process begins by collecting important details from incoming packet data and categorizing them based on established attack types. We used feature selection approaches to improve system performance and decrease computational complexity. These relevant characteristics were retrieved from incoming traffic patterns during the preprocessing stage. The availability of this complex computing equipment and GPU capabilities within Google Colab greatly helped the efficiency and efficacy of our feature selection and analysis operations.

Our framework's core consists of attack detection sub-engines, the backbone of our experiment. These sub-engines possess the versatility to identify various attack types, their number aligning with the variety in our training database.

To prepare for our experiments, we loaded the dataset into DataFrames. Subsequently, we labeled entries in the Traffic column as malicious for attack-related traffic and benign for non-malicious traffic, as shown in Table 3. This preprocessing step served as the foundation for our subsequent experiments. Moreover, we eliminate redundant columns and trim leading and trailing spaces from strings in preprocessing. We next filter out packets related to port scans, okiru malware, and other harmful packets, such as those connected to command-and-control (C&C) and distributed denial-of-service (DDOS) assaults, as Table 3 illustrates.

Table 3. Combining, Loading, and Filtering Malware Datasets (Datasets 1–3).

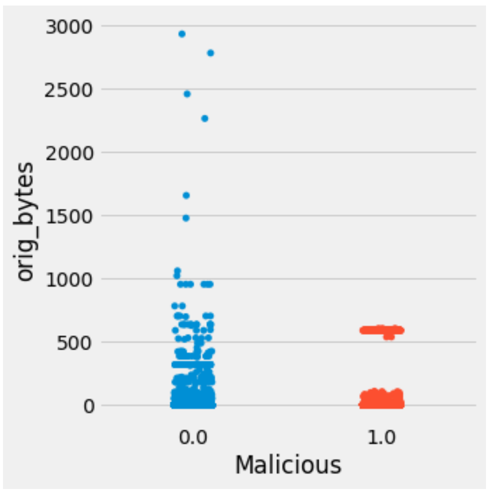
#	Column	Non-Null Count	Dtype
0	ts	8008 non-null	object
1	uid	8008 non-null	object
2	id.orig_h	8008 non-null	object
3	id.orig_p	8008 non-null	int64
4	id.resp_h	8008 non-null	object
5	id.resp_p	8008 non-null	int64
6	proto	8008 non-null	object
7	service	196 non-null	object
8	duration	3213 non-null	object
9	orig_bytes	3213 non-null	float64
10	resp_bytes	3213 non-null	float64
11	conn_state	8008 non-null	object
12	local_orig	0 non-null	float64
13	local_resp	0 non-null	float64
14	missed_bytes	8008 non-null	int64
15	history	7481 non-null	object
16	orig_pkts	8008 non-null	int64
17	orig_ip_bytes	8008 non-null	int64
18	resp_pkts	8008 non-null	int64
19	resp_ip_bytes	8008 non-null	int64
20	TrafficLabeled	8008 non-null	object

Subsequently, we implement the distribution of the target variable to specific columns to visually analyze the distribution of malicious and benign data, as illustrated in Table 4.

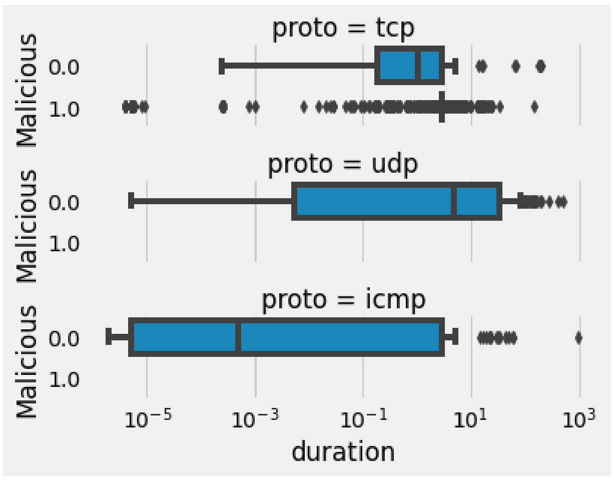
Figure 5 provides insights into examining the malicious attribute in conjunction with various services, presented through a box plot. This graphical representation showcases the distribution of malicious and benign data and highlights the presence of outliers for further examination.

Table 4. Combining, Loading, and Filtering Malware Datasets (Datasets 4–6).

#	Column	Non-Null Count	Dtype
0	ts	8008 non-null	object
1	id.orig_h	8008 non-null	object
2	id.orig_p	8008 non-null	object
3	id. resp_h	8008 non-null	object
4	id.resp_p	8008 non-null	object
5	proto	8008 non-null	object
6	service	8008 non-null	object
7	duration	8008 non-null	float 64
8	orig_bytes	8008 non-null	float64
9	resp_bytes	8008 non-null	float64
10	conn_state	8008 non-null	object
11	missed bytes	8008 non-null	int64
12	history	8008 non-null	object
13	orig_pkts	8008 non-null	int64
14	orig_ip_bytes	8008 non-null	int64
15	resp_pkts	8008 non-null	int64
16	resp_ip_bytes	8008 non-null	int64
17	Traffic_Labeled	8008 non-null	object
18	Maliciоūs	8008 non-null	loat6



(**a**)



(**b**)

Figure 5. Exploring the Distribution of Malicious Features with Other Dependent Features: (**a**) Feature orig_byes distribution. (**b**) Feature duration.

Figure 6 provides an insightful representation of the distribution of malicious data across individual variables. It highlights data points that deviate significantly from the main distribution, identifying them as outliers within the dataset. This figure further facilitates a comprehensive analysis of these outlier data points, shedding light on their impact on the dataset's overall correlation. While most data records exhibit strong relevance, it is essential to recognize and investigate records that deviate substantially from the norm, thereby classifying them as outliers.

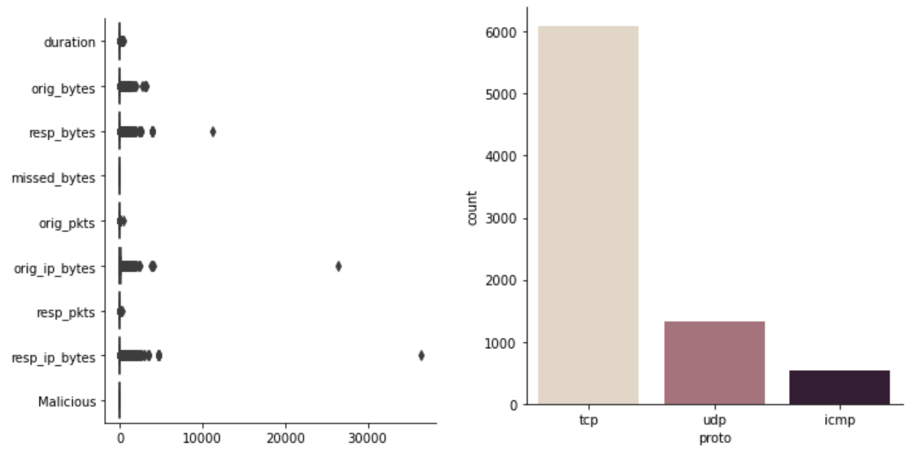


Figure 6. Analyzing Outliers and Packet Counts Across Various Protocols.

Additionally, when contrasting the packet reception counts originating from diverse protocols, we note a higher influx of packets for TCP than UDP and ICMP. This discovery underscores the necessity for heightened scrutiny before classification.

Figure 7 presents a custom diverging color map pattern showcasing feature relevancy through a heatmap visualization. Feature selection is crucial, and we utilize the MDI method in conjunction with XGBoost, a widely adopted technique. This method gauges feature importance by assessing the reduction in impurity when the feature is integrated into the decision tree-building process. The MDI approach quantifies how much a feature influences the decision-making within the tree. The importance of each feature is typically expressed as a percentage, with the most significant feature assigned a value of 100%. Subsequent features are ranked in order of importance, represented by the percentage decrease in impurity upon removal. Figure 7 includes additional details such as feature names, corresponding importance values, and graphical representations, such as bar charts or heatmaps, for a comprehensive view of feature importance.

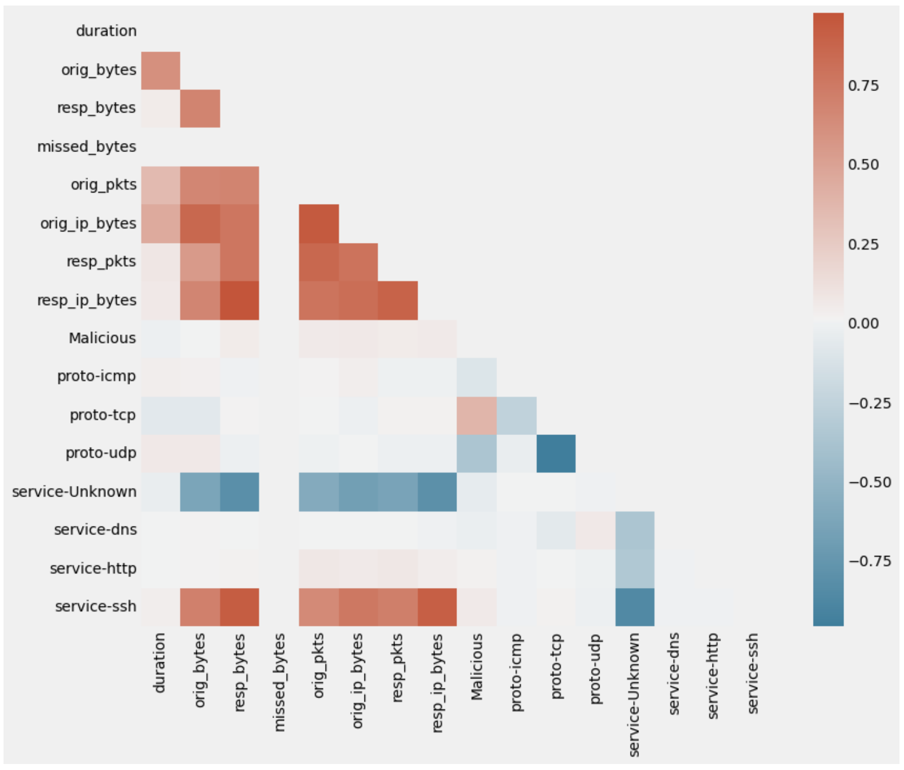


Figure 7. Analyzing Feature Dependency with a Heat Map Correlation.

Figure 8 effectively recognizes that a higher importance score signifies the feature's greater significance in influencing the model's predictions. Therefore, features with elevated importance scores should be deemed more pertinent and valuable in elucidating the target variable's behavior. Furthermore, further investigation is warranted in cases where multiple features share identical relevance ratings. This may include conducting correlation analysis or employing feature selection algorithms to identify the most informative subset of data for the model.

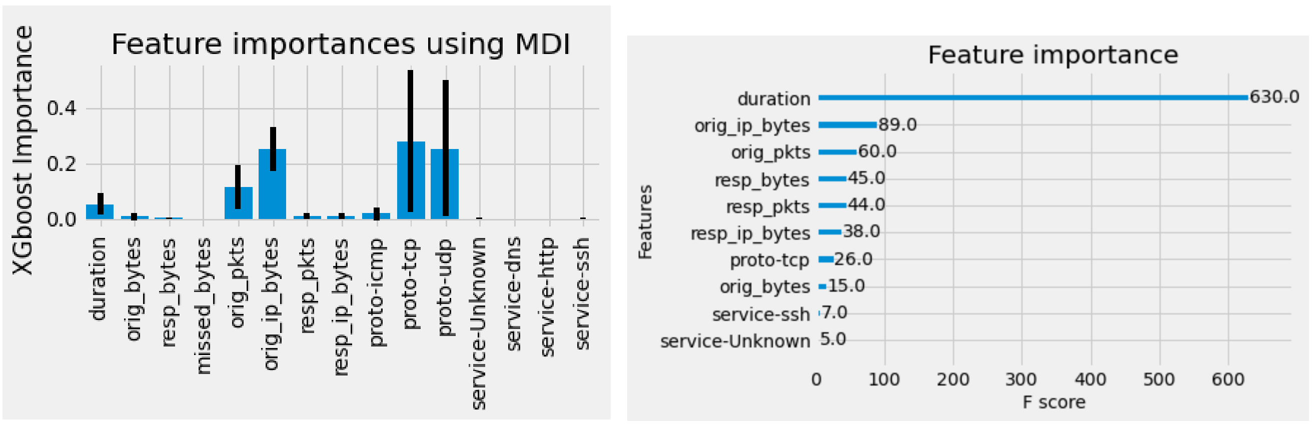


Figure 8. Relative Importance of Each Feature towards Target using MDI and XGBoost.

We employed the ZAT DataFrame-To-Matrix class to handle categorical data effectively. This class uses a heuristic approach to detect categorical data and ensures explicit conversion before data transformation. In our analysis and clustering of malware samples, we applied two well-established techniques: K-means and PCA clustering.

The clustering of malware using K-means and PCA is shown in Table 5. This process involves reducing the data to two or three dimensions using PCA and applying K-means clustering to group similar samples. In the resulting plot, each data point is represented as a colored dot, indicating its cluster membership. Additionally, centroids may be displayed, signifying the average values within each cluster.

Clustering is mostly used to find unique subgroups within the dataset and group malware samples that are similar to each other. Table 5 serves as a valuable tool for uncovering patterns and trends within the data, complementing the insights gained during exploratory data analysis (EDA). Consequently, Table 5 aids in understanding the dataset's behavior, highlighting potential relationships and similarities among the samples.

Based on our observations, it is clear that the clusters exhibit distinct separation, indicating significant dissimilarities in the behavior or characteristics of malware samples within each cluster. To assess the quality of our clustering, we employed the Silhouette Scoring technique, a standard method for evaluating clustering efficiency. By calculating the average distance between a data point and other points in its cluster, cohesiveness is represented. It also calculates, by average their distances, the spacing between the particular point and the points in the closest nearby cluster.

Silhouette Scores, which range from -1 to 1, serve as indicators of clustering quality. Higher values on this scale suggest more favorable clustering results. A Silhouette Score of 1 signifies that a data point is correctly assigned to its cluster and distinctly separated from surrounding clusters, while a score of -1 implies the opposite. Figure 9 presents a clustering representation using Silhouette Scores to evaluate the precision of our clustering outcomes. Each data point is depicted as a dot in this visual representation, and its vertical position corresponds to its Silhouette Score. Dot colors indicate the cluster to which each point is assigned, while the width of each bar illustrates the number of points allocated to each cluster. Clustering figures that exhibit well-defined and separate clusters and high average Silhouette Scores indicate effective clustering.

Table 5. Identifying Distinct Groups Through Clustering of Malware Attacks.

	ts	1d.orig p	1d.resp_p	duration	orig_bytes	resp_bytes	orig_ip_bytes	resp_ip_bytes	proto-icnp	proto-tcp	proto-udp	…
	00:29.6	123	123	0.102094	48	48	76	76	e	theta	1	…
	01:15.4	123	123	0.051969	48.6	48	76	76	8	8	1	…
	01:43.0	123	123	0.008233	96.6	96	152	152	0	theta	1	…
	02:22.8	123	123	0.008248	96	96	152	152	e	e	1	…
	03:48.0	29930	47659	0.044972	103	289	131	317	e	0	1	…
Cluster 0	…	…	…	…	…	…	…	…	…	…	…	…
	55:55.4	123	123	0.037777	48.6	48	76	76	e	B	1	…
	56:10.4	123	123	72.025952	384.8	384	698	698	8	theta	1	…
	56:48.0	49094	53	0.010745	39.6	55	67	83	8	8	1	…
	57:08.0	123	123	0.017637	48.6	48	76	76	0	theta	1	…
	57:08.0	123	123	0.017637	48.6	48	76	76	8	B	1	…
	ts	1d.orig p	1d.resp_p	duration	orig_bytes	resp_bytes	orig_ip_bytes	resp_1p_bytes	proto-icnp	proto-tcp	proto-udp	…
	00:10.4	123	123	0.00809	0	0	76	0	8	theta	1	…
	00:10.4	123	123	0.00009	0	B. 0	76	theta	theta	B	1	…
	08:32.2	29930	6889	0	0	0	95	theta	0	e	1	…
	08:36.0	29930	1844	31.982511	318	0	492	theta	a	e	1	…
	09:40.4	123	123	0.809090	0	0	76	theta	e	0	1	…
Cluster 1	…	…	…	…	…	…	…	…	…	…	…	…
	59:30.6	43763	39392	0.00009	0	0	40	theta	8	B	1	…
	59:39.8	43763	20473	0	0	0	40	theta	theta	theta	1	…
	59:39.0	43763	20473	0	0	0	40	theta	8	B	1	…
	59:44.8	43763	9376	0	0	0	40	theta	0	theta	1	…
	59:59.6	43763	50352	0.00809	0	0	40		a	e	1	…
	ts	id.orig_p	1d.resp_p	duration	or1g_bytes	resp_bytes	orig_ip_bytes	resp_1p_bytes	proto-icnp	proto-tcp	proto-udp	…
	00:38.5	24159	8981	0.000254	0	0	168	theta	0	1	0	…
	09:46.6	35874	2323	2.998558	0	0	189	0	theta	1	theta	…
	00:46.0	35874	2323	2.998558	0	0	180	theta	8	1	theta	…
	69:51,0	49894	23	2.998788	0		189	theta	theta	1	8	…
	01:00,6	52469	22	2.996594	0	0	189	6	a	1	8	…
Cluster 2	…	…	…	…	…	…	…	…	…	…	…	…
	58:29.6	49989	23	2.998792	0	e.0	189	6	a	1	theta	…
	58:29.6	49989	23	2.998792	0	e.e	189	theta	0	1	0	…
	59:34.6	24159	8881	0.000255	0	e.e	160	theta	theta	1	theta	…
	59:39.0	60379	8880	2.998789	0	0	189	0	8	1	0	…
	59:39.0	69379	8880	2.998789	0	0	189	e	theta	1	a	…
…	…	…	…	…	…	…	…	…	…	…	…	…

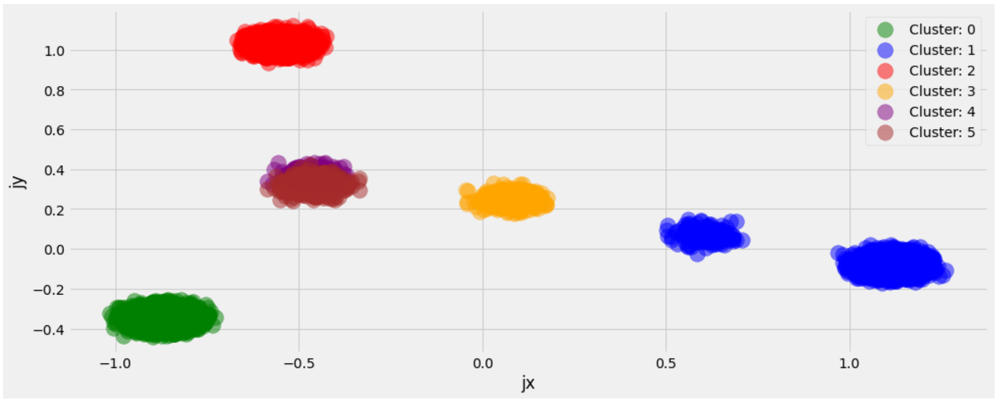


Figure 9. Utilizing Silhouette Scores for Distinct Group Identification through Clustering.

Subsequently, we proceeded with the classification phase. Our approach commenced with fine-tuning hyperparameters and weights within the GNGRUE, achieved through applying the Jaya Algorithm (JA). The optimized weights were passed through the ensembler, and the model underwent training and evaluation for each weight combination.

Optimization methods played a crucial role in effectively handling extensive malware data. Our comprehensive experimentation identified the most advantageous parameter combination, as determined by JA, resulting in optimal weights: w_G = 0.5742 and w_R = 0.3121. These weights represent the solution that best enhances our malware detection capabilities.

To evaluate our model, we assessed the performance of GNGRUE by passing the hyperparameters through the optimization method JA. Additionally, we evaluated our model by computing their True Positive and True Negative values, plotted for the proposed and existing methodologies in Figure 10.

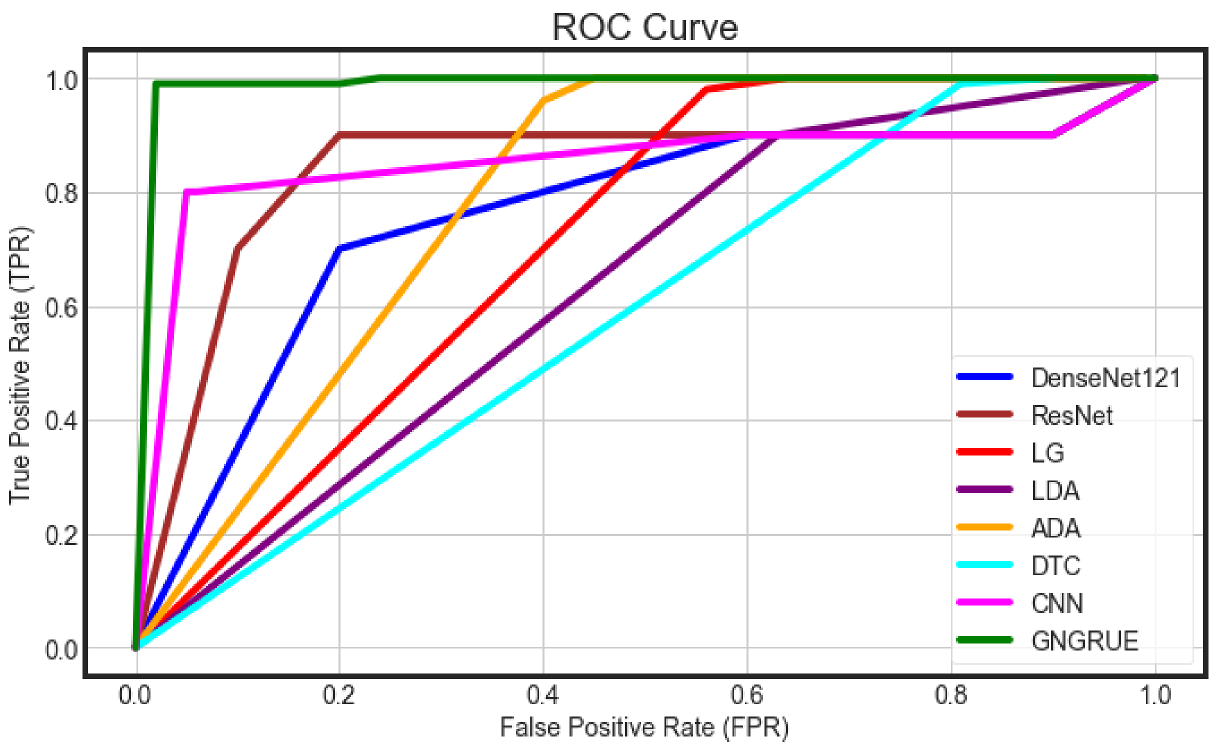


Figure 10. ROC Curve and Mapping True Positive as well as True Negative Values.

The main contribution of this article is GNGRUE, which demonstrated superior performance in handling large volumes of data while achieving high accuracy scores and ROC curve values. The accuracy values of the proposed model are depicted in Figure 11, ranging from 0 to 1. Higher accuracy is indicated by numbers that are closer to one, and lesser accuracy is shown by values that are closer to zero.

Using the assessment metrics given in Table 6, we evaluated the performance of our suggested ensemble strategy and contrasted it with the existing techniques. Notably, GNGRUE outperformed the existing methods. Our proposed methods displayed the ability to adjust their parameter values optimally, even with an expanded input dataset, resulting in enhanced performance.

The accuracy of binary classifications was assessed using the Matthews correlation coefficient (MCC) measure in Figure 12. The MCC value reflects the degree of correlation between the predicted and actual labels for malware samples. A higher MCC value indicates superior performance in correctly identifying malware samples. Figure 12 shows that the model correctly predicts all malware samples as positive and all benign samples as negative if a technique displays an MCC value close to one. Conversely, an MCC value closer to 0 or −1 suggests that the model is making incorrect predictions and performing the opposite of its intended purpose.

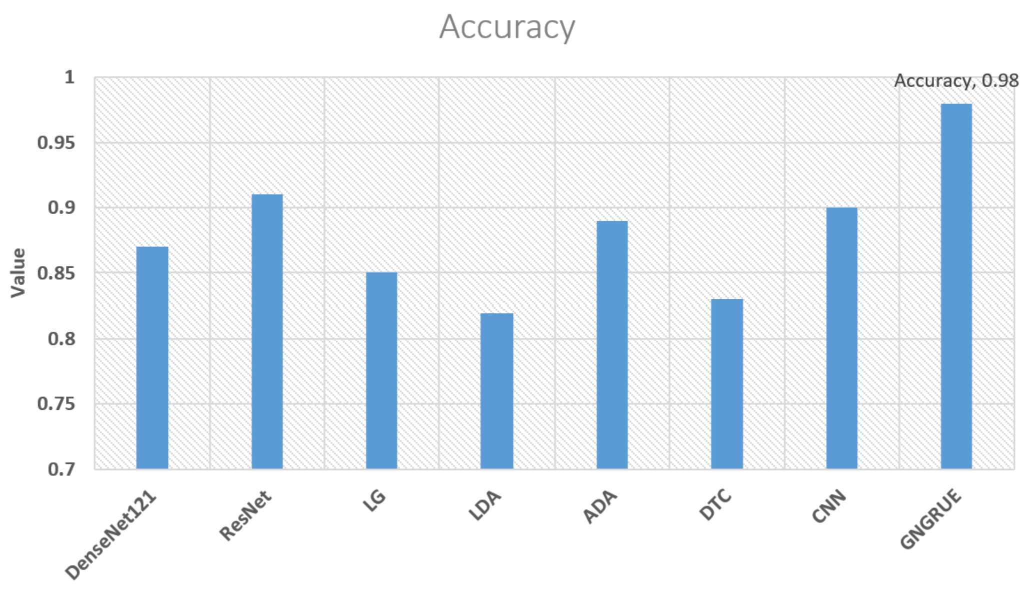


Figure 11. Accuracy Values between the Existing and Proposed Methods.

Table 6. Comparison of Performance Evaluation Metrics Values of Proposed vs. Existing.

Techniques	F1-Score	Accuracy	Precision	Recall	ROC-AUC	MCC	Fowlkes–Mallows Index	Cohen's Kappa
ADA [20]	0.78	0.89	0.81	0.76	0.88	0.67	0.79	0.65
DTC [21]	0.72	0.83	0.75	0.69	0.84	0.59	0.73	0.57
LDA [22]	0.68	0.82	0.72	0.65	0.8	0.55	0.7	0.52
CNN [27]	0.85	0.9	0.88	0.82	0.92	0.71	0.86	0.69
DenseNet121 [28]	0.82	0.87	0.85	0.79	0.92	0.74	0.82	0.71
LG [29]	0.73	0.85	0.76	0.7	0.87	0.63	0.75	0.61
ResNet [35]	0.89	0.91	0.92	0.87	0.94	0.79	0.88	0.78
GNGRUE	0.97	0.98	0.96	0.98	0.99	0.94	0.95	0.93

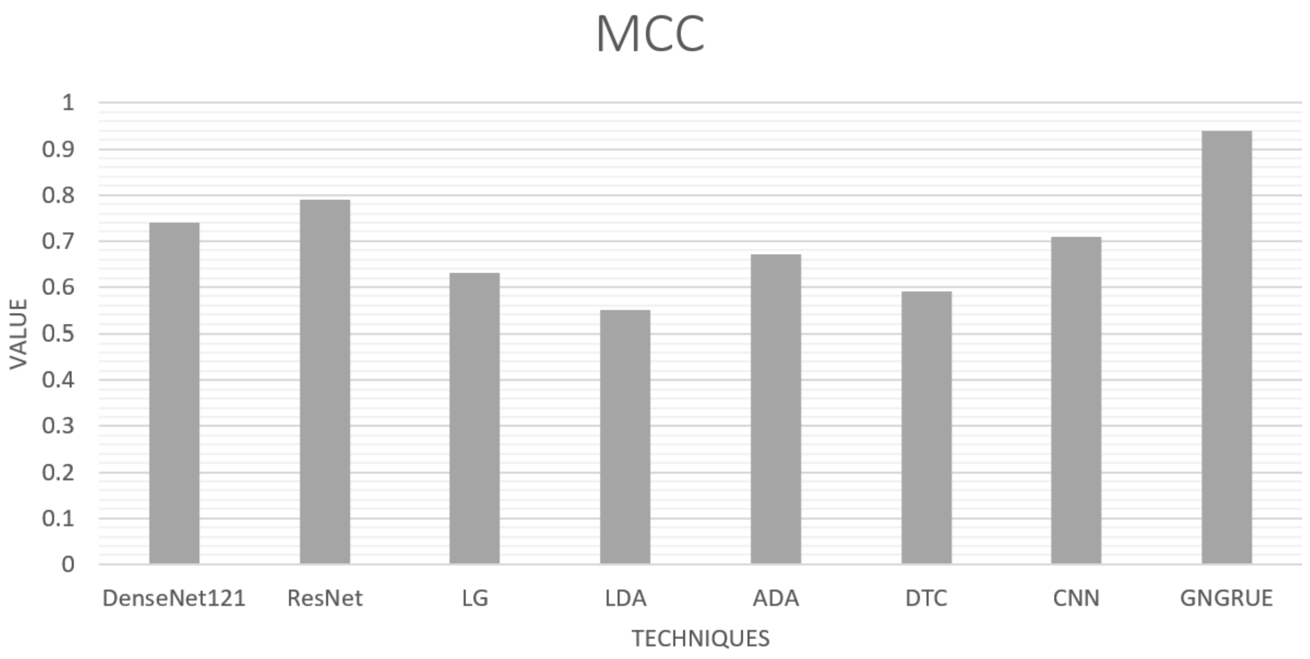


Figure 12. Comparison of MCC Values Between Existing and Proposed GNGRUE.

The suggested system is composed of a model selector module that chooses classifiers based on processing speeds and accuracy for specific sub-engines, and a feature selection module that minimizes the original feature set. Compared to conventional detection architectures, our novel hybrid categorization technique produces a system that is more accurate and efficient. Figure 13 displays the temporal complexity of both our suggested and cutting-edge techniques.

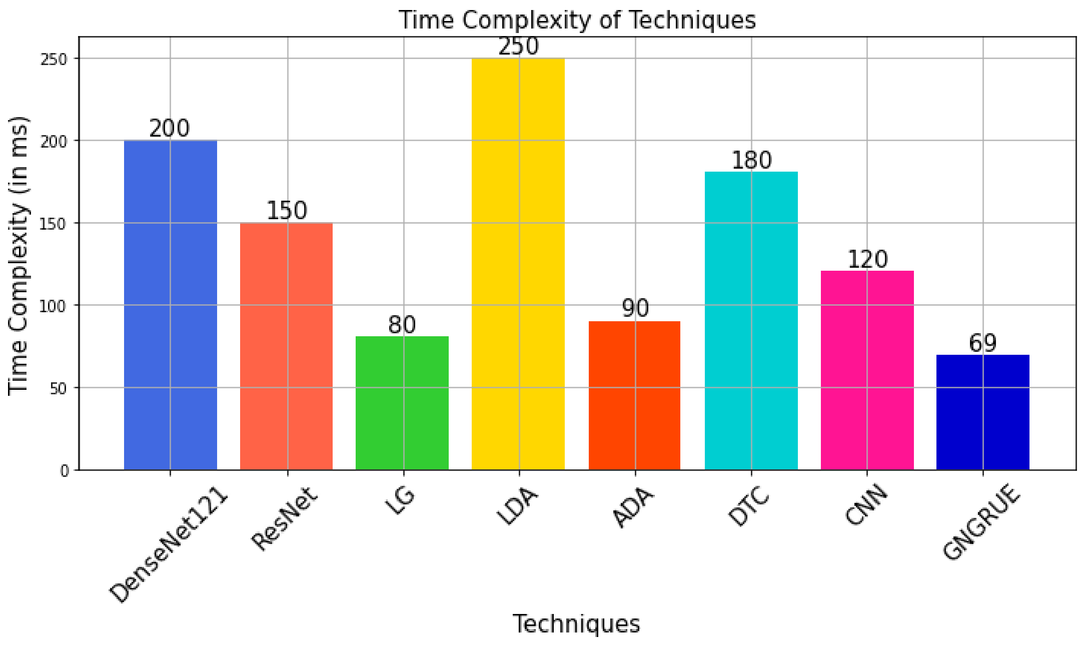


Figure 13. Time Complexity/Execution time of proposed and existing methods.

Furthermore, we conducted concurrent tests on the attack detector and model trainer modules. It was observed that the model trainer module's processing speed improved significantly, being three times faster during parallel training compared to serial training. The attack detection module also exhibited enhanced processing speed compared to serial classification, although it may require a multithreading architecture or a more powerful GPU for optimal performance, as shown in Figure 14. The experimental results highlight that our sequential design improves the accuracy of IoT malware attack detection and maintains a lightweight.

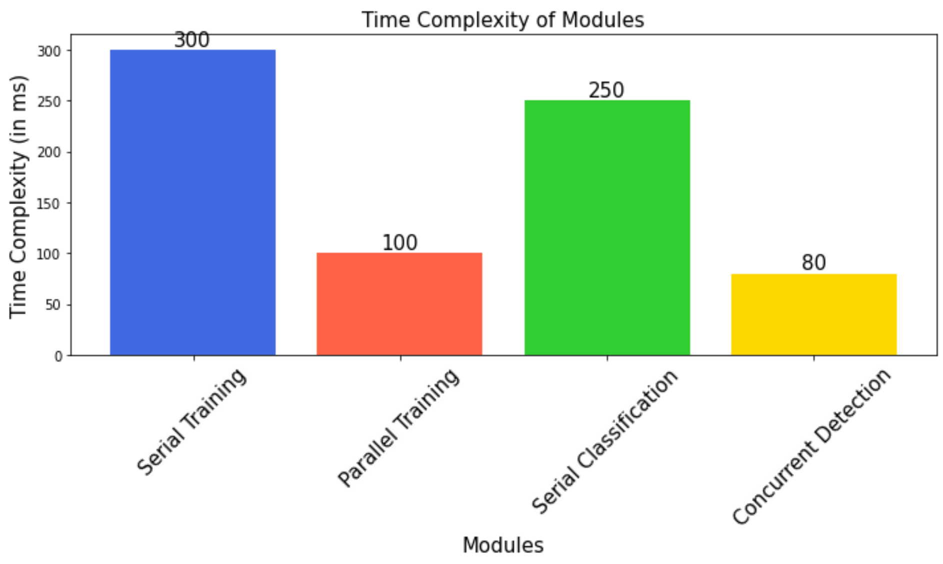


Figure 14. Time Complexity of different modules.

6. Discussion on Adaptability to New Types of Malware

In the context of malware detection, it is critical to evaluate the proposed method's adaptability to new types of malware that may not be included in the previously determined datasets utilized in our studies. While our strategy performed well on the eight particular datasets, the ever-changing malware landscape must thoroughly examine the capabilities to deal with upcoming threats.

Our approach's architecture is designed to tolerate new varieties of malware, even if they were not included in the training data. The feature selection approaches, and machine learning models used in our method, contribute to the system's versatility. These components are not limited to the eight datasets but are intended to generalize and extract useful information from various data sources. This versatility allows us to extend our strategy to previously unknown malware varieties.

As we highlight the significance of practicality and real-world applicability, it is acknowledged that the cybersecurity profession needs solutions that can effectively address new threats. As a result, our method's flexibility to new malware types matches the dynamic nature of the cybersecurity field. Our strategy is intended to serve as a basis for malware detection, with the understanding that continual research and development are required to improve its flexibility.

While our study offers findings using predefined datasets, we encourage further research into expanding and improving our technique to deal with new and diverse forms of malware. The capacity of any malware detection solution to adopt in conjunction with the ever-changing threat environment is critical to its real-world success. This dedication to flexibility is a critical component of our study and a focus area for future efforts. Our proposed technique provides a solid foundation for malware identification, displaying excellent performance on existing datasets. However, adaptability to new forms of malware remains a vital component that needs more research and development. We are committed to updating our technique as cybersecurity evolves to guarantee it stays successful in recognizing and mitigating future threats.

7. Conclusions

With the continuous evolution of IoT technologies, the incidence of cyberattacks on these devices, particularly malware attacks, is increasing. These threats present a substantial challenge in IoT environments. Attackers often exploit vulnerabilities through command and control (C&C) servers, seizing control of compromised devices to target unsuspecting hosts. Our proposed detection architecture addresses these issues by identifying known attacks and their variations while remaining adaptable to emerging threats. Precision and efficiency are achieved through feature selection, distribution analysis, clustering, and the isolation forest model. The incorporation of a hybrid classification method enhances accuracy and accelerates detection.

In our future research, we will investigate normal traffic patterns on diverse IoT devices to bolster our anomaly detection engine's ability to recognize unknown attacks. Furthermore, we plan to validate our system's performance through testbed implementation and propose strategies for effectively isolating compromised devices.

Author Contributions: Conceptualization, N.A.; Formal analysis, A.A.A.; Funding acquisition, A.A.A.; Investigation, A.A.A.; Methodology, A.A.A.; Resources, N.A.; Supervision, A.A.A.; Writing—original draft, N.A.; Writing—review and editing, A.A.A. All authors have read and agreed to the published version of the manuscript.

Funding: The authors extend their appreciation to the Deputyship for Research & Innovation, Ministry of Education in Saudi Arabia for funding this research work through project number MoE-IF-UJ-22-4100409-8.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The dataset is publicly available online on https://www.stratosphereips.org/datasets-iot23 (accessed on 1 September 2023).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mishra, N.; Pandya, S. Internet of Things Applications, Security Challenges, Attacks, Intrusion Detection, and Future Visions: A Systematic Review. *IEEE Access* **2021**, *9*, 59353–59377. [CrossRef]
2. Gaurav, A.; Gupta, B.B.; Panigrahi, P.K. A Comprehensive Survey on Machine Learning Approaches for Malware Detection in IoT-Based Enterprise Information System. *Enterp. Inf. Syst.* **2023**, *17*, 2023764. [CrossRef]
3. Macas, M.; Wu, C.; Fuertes, W. A Survey on Deep Learning for Cybersecurity: Progress, Challenges, and Opportunities. *Comput. Netw.* **2022**, *212*, 109032. [CrossRef]
4. Zhou, J.X.; Shen, G.Q.; Yoon, S.H.; Jin, X. Customization of On-Site Assembly Services by Integrating the Internet of Things and BIM Technologies in Modular Integrated Construction. *Autom. Constr.* **2021**, *126*, 103663. [CrossRef]
5. Shaukat, K.; Luo, S.; Varadharajan, V. A Novel Deep Learning-Based Approach for Malware Detection. *Eng. Appl. Artif. Intell.* **2023**, *122*, 106030. [CrossRef]
6. Palša, J.; Ádám, N.; Hurtuk, J.; Chovancová, E.; Madoš, B.; Chovanec, M.; Kocan, S. MLMD—A Malware-Detecting Antivirus Tool Based on the XGBoost Machine Learning Algorithm. *Appl. Sci.* **2022**, *12*, 6672. [CrossRef]
7. Maniriho, P.; Mahmood, A.N.; Chowdhury, M.J.M. A Study on Malicious Software Behaviour Analysis and Detection Techniques: Taxonomy, Current Trends and Challenges. *Future Gener. Comput. Syst.* **2022**, *130*, 1–18. [CrossRef]
8. Udousoro, I.C. Machine Learning: A Review. *Semicond. Sci. Inf. Devices* **2020**, *2*, 5–14.
9. Shaukat, K.; Luo, S.; Varadharajan, V. A Novel Method for Improving the Robustness of Deep Learning-Based Malware Detectors against Adversarial Attacks. *Eng. Appl. Artif. Intell.* **2022**, *116*, 105461. [CrossRef]
10. Aslan, Ö.A.; Samet, R. A Comprehensive Review on Malware Detection Approaches. *IEEE Access* **2020**, *8*, 6249–6271. [CrossRef]
11. Mishra, A.; Almomani, A. Malware Detection Techniques: A Comprehensive Study. *Insights Int. Interdiscip. J.* **2023**, *1*, 1–5.
12. Kwon, H.Y.; Kim, T.; Lee, M.K. Advanced Intrusion Detection Combining Signature-Based and Behavior-Based Detection Methods. *Electronics* **2022**, *11*, 867. [CrossRef]
13. Singh, J.; Singh, J. A Survey on Machine Learning-Based Malware Detection in Executable Files. *J. Syst. Archit.* **2021**, *112*, 101861. [CrossRef]
14. Tayyab, U.E.H.; Khan, F.B.; Durad, M.H.; Khan, A.; Lee, Y.S. A Survey of the Recent Trends in Deep Learning Based Malware Detection. *J. Cybersecur. Priv.* **2022**, *2*, 800–829. [CrossRef]
15. Aboaoja, F.A.; Zainal, A.; Ghaleb, F.A.; Al-rimy, B.A.S.; Eisa, T.A.E.; Elnour, A.A.H. Malware Detection Issues, Challenges, and Future Directions: A Survey. *Appl. Sci.* **2022**, *12*, 8482. [CrossRef]
16. Alomari, E.S.; Nuiaa, R.R.; Alyasseri, Z.A.A.; Mohammed, H.J.; Sani, N.S.; Esa, M.I.; Musawi, B.A. Malware Detection Using Deep Learning and Correlation-Based Feature Selection. *Symmetry* **2023**, *15*, 123. [CrossRef]
17. Hemalatha, J.; Roseline, S.A.; Geetha, S.; Kadry, S.; Damaševičius, R. An Efficient DenseNet-Based Deep Learning Model for Malware Detection. *Entropy* **2021**, *23*, 344. [CrossRef]
18. Ngo, Q.D.; Nguyen, H.T.; Le, V.H.; Nguyen, D.H. A Survey of IoT Malware and Detection Methods Based on Static Features. *ICT Express* **2020**, *6*, 280–286. [CrossRef]
19. Khalid, A.; Badshah, G.; Ayub, N.; Shiraz, M.; Ghouse, M. Software Defect Prediction Analysis Using Machine Learning Techniques. *Sustainability* **2023**, *15*, 5517. [CrossRef]
20. Ravi, V.; Alazab, M.; Selvaganapathy, S.; Chaganti, R. A Multi-View Attention-Based Deep Learning Framework for Malware Detection in Smart Healthcare Systems. *Comput. Commun.* **2022**, *195*, 73–81. [CrossRef]
21. Bhat, P.; Behal, S.; Dutta, K. A System Call-Based Android Malware Detection Approach with Homogeneous & Heterogeneous Ensemble Machine Learning. *Comput. Secur.* **2023**, *130*, 103277.
22. Dewanje, A.; Kumar, K.A. A New Malware Detection Model Using Emerging Machine Learning Algorithms. *Int. J. Electron. Inf. Eng.* **2021**, *13*, 24–32.
23. Patil, S.; Varadarajan, V.; Walimbe, D.; Gulechha, S.; Shenoy, S.; Raina, A.; Kotecha, K. Improving the Robustness of AI-Based Malware Detection Using Adversarial Machine Learning. *Algorithms* **2021**, *14*, 297. [CrossRef]
24. Taheri, R.; Ghahramani, M.; Javidan, R.; Shojafar, M.; Pooranian, Z.; Conti, M. Similarity-Based Android Malware Detection Using Hamming Distance of Static Binary Features. *Future Gener. Comput. Syst.* **2020**, *105*, 230–247. [CrossRef]
25. Sayadi, H.; Patel, N.; Sasan, A.; Rafatirad, S.; Homayoun, H. Ensemble Learning for Effective Run-Time Hardware-Based Malware Detection: A Comprehensive Analysis and Classification. In Proceedings of the 55th Annual Design Automation Conference, San Francisco, CA, USA, 24–29 June 2018; pp. 1–6.
26. Sihwail, R.; Omar, K.; Zainol Ariffin, K.A.; Al Afghani, S. Malware Detection Approach Based on Artifacts in Memory Image and Dynamic Analysis. *Appl. Sci.* **2019**, *9*, 3680. [CrossRef]
27. Huang, X.; Ma, L.; Yang, W.; Zhong, Y. A Method for Windows Malware Detection Based on Deep Learning. *J. Signal Process. Syst.* **2021**, *93*, 265–273. [CrossRef]

28. Atitallah, S.B.; Driss, M.; Almomani, I. A Novel Detection and Multi-Classification Approach for IoT-Malware Using Random Forest Voting of Fine-Tuning Convolutional Neural Networks. *Sensors* **2022**, *22*, 4302. [CrossRef]
29. Malvacio, E.M.; Duarte, J.C. An assessment of the effectiveness of pretrained neural networks for malware detection. *IEEE Latin America Transactions* **2023**, *21*, 47–53. [CrossRef]
30. Roseline, S.A.; Geetha, S.; Kadry, S.; Nam, Y. Intelligent Vision-Based Malware Detection and Classification Using Deep Random Forest Paradigm. *IEEE Access* **2020**, *8*, 206303–206324. [CrossRef]
31. Andreopoulos, W.B. Malware Detection with Sequence-Based Machine Learning and Deep Learning. In *Malware Analysis Using Artificial Intelligence and Deep Learning*; Springer: Cham, Switzerland, 2021; pp. 53–70.
32. Zhang, N.; Xue, J.; Ma, Y.; Zhang, R.; Liang, T.; Tan, Y.A. Hybrid Sequence-Based Android Malware Detection Using Natural Language Processing. *Int. J. Intell. Syst.* **2021**, *36*, 5770–5784. [CrossRef]
33. Gržinić, T.; González, E.B. Methods for Automatic Malware Analysis and Classification: A Survey. *Int. J. Inf. Comput. Secur.* **2022**, *17*, 179–203. [CrossRef]
34. Roh, Y.; Heo, G.; Whang, S.E. A Survey on Data Collection for Machine Learning: A Big Data-AI Integration Perspective. *IEEE Trans. Knowl. Data Eng.* **2019**, *33*, 1328–1347. [CrossRef]
35. Demirezen, M.U. Image Based Malware Classification with Multimodal Deep Learning. *Int. J. Inf. Secur. Sci.* **2021**, *10*, 42–59.
36. Manzil, H.H.R.; Naik, S.M. Detection approaches for android malware: Taxonomy and review analysis. *Expert Syst. Appl.* **2023**, *2023*, 122255.
37. Manzano, C.; Meneses, C.; Leger, P.; Fukuda, H. An empirical evaluation of supervised learning methods for network malware identification based on feature selection. *Complexity* **2022**, *2022*, 6760920. [CrossRef]
38. Kimmel, J.C.; Mcdole, A.D.; Abdelsalam, M.; Gupta, M.; Sandhu, R. Recurrent neural networks-based online behavioral malware detection techniques for cloud infrastructure. *IEEE Access* **2021**, *9*, 68066–68080. [CrossRef]
39. Ghurab, M.; Gaphari, G.; Alshami, F.; Alshamy, R.; Othman, S. A detailed analysis of benchmark datasets for a network intrusion detection system. *Asian J. Res. Comput. Sci.* **2021**, *7*, 14–33. [CrossRef]
40. Alavizadeh, H.; Jang-Jaccard, J.; Enoch, S.Y.; Al-Sahaf, H.; Welch, I.; Camtepe, S.A.; Kim, D.S. A Survey on Threat Situation Awareness Systems: Framework, Techniques, and Insights. *arXiv* **2021**, arXiv:2110.15747.
41. Saeed, M.M. A real-time adaptive network intrusion detection for streaming data: A hybrid approach. *Neural Comput. Appl.* **2022**, *34*, 6227–6240. [CrossRef]
42. Malware Dataset, IoT23. Available online: https://www.stratosphereips.org/datasets-iot23 (accessed on 12 September 2023).
43. Chicco, D.; Jurman, G. The Advantages of the Matthews Correlation Coefficient (MCC) over F1 Score and Accuracy in Binary Classification Evaluation. *BMC Genom.* **2020**, *21*, 6. [CrossRef]
44. Geetha, K.; Brahmananda, S.H. Network Traffic Analysis Through Deep Learning for Detection of an Army of Bots in Health IoT Network. *Int. J. Pervasive Comput. Commun.* **2022**. *ahead-of-print*.
45. Manju, N.; Harish, B.S.; Prajwal, V. Ensemble Feature Selection and Classification of Internet Traffic Using XGBoost Classifier. *Int. J. Comput. Netw. Inf. Secur.* **2019**, *11*, 37. [CrossRef]
46. AbdulRaheem, M.; Oladipo, I.D.; Imoize, A.L.; Awotunde, J.B.; Lee, C.C.; Balogun, G.B.; Adeoti, J.O. Machine Learning Assisted Snort and Zeek in Detecting DDoS Attacks in Software-Defined Networking. *Int. J. Inf. Technol.* **2023**, 1–17. [CrossRef]
47. Anaraki, S.A.M.; Haeri, A.; Moslehi, F. A Hybrid Reciprocal Model of PCA and K-means with an Innovative Approach of Considering Sub-datasets for the Improvement of K-means Initialization and Step-by-Step Labeling to Create Clusters with High Interpretability. *Pattern Anal. Appl.* **2021**, *24*, 1387–1402. [CrossRef]
48. Shutaywi, M.; Kachouie, N.N. Silhouette analysis for performance evaluation in machine learning with applications to clustering. *Entropy* **2021**, *23*, 759. [CrossRef] [PubMed]
49. Lovmar, L.; Ahlford, A.; Jonsson, M.; Syvänen, A.C. Silhouette Scores for Assessment of SNP Genotype Clusters. *BMC Genom.* **2005**, *6*, 35. [CrossRef]
50. Hariri, S.; Kind, M.C.; Brunner, R.J. Extended Isolation Forest. *IEEE Trans. Knowl. Data Eng.* **2019**, *33*, 1479–1489. [CrossRef]
51. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. GhostNet: More Features from Cheap Operations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; IEEE/CVF, Seattle, WA, USA, 13–19 June 2020; pp. 1580–1589.