Babylonian Journal of Machine Learning
Vol.2026, **pp**. 1–22
DOI: https://doi.org/10.58496/BJML/2026/001; ISSN: 3006-5429
https://mesopotamian.press/journals/index.php/BJML

Research Article

Effective Android Malware Detection Using CNN and LSTM Model with GWO-Based Feature Selection

Hawraa.O. Musa[1], *, , Muhanad.T. Younis [2],

[1] *Information Institute for Postgraduate Studies ,Iraqi Commission for Computers and Informatics ,Baghdad, Iraq.*

[2] *Department of Computer Science, College of Science, Mustansiriya University, Baghdad, Iraq.*

ARTICLEINFO

Article History

Received 17 Nov 2025
Revised 15 Dec 2026
Accepted 28 Jan 2026
Published 19 Feb 2026

Keywords

Malware detection

Deep Learning

CICMalDroid 2020

Grey Wolf Optimizer (GWO)

Cybersecurity

ABSTRACT

Malware or malicious software represents one of the most significant cybersecurity threats, compromising the integrity, confidentiality, and availability of computer systems and networks. Traditional malware detection methods seek to locate specific malware samples and families to recognize harmful code, and can be located using traditional signature- and rule-based detection methods. The study aims at the creation of malware detectors based on deep learning and optimization algorithms. The present study will propose a more advanced malware detection system that employs deep learning models and the Grey Wolf Optimization (GWO) algorithm to select essential features. This model used data balancing Synthetic Minority Oversampling Technique (SMOTE), and the models were tested on the CICMalDroid2020 dataset. Using two different models were tested: Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) with Grey Wolf Optimization. The experimental findings indicate that the CNN-GWO model has an model achieves 93% accuracy, 92% precision, and 92% recall, and the LSTM-GWO model has an 90% accuracy, 90% precision, and 90% recall, respectively. These findings show that both models can be effective in detecting and classifying malware, but CNN has a few higher performance rates. The novelty of this method is the independent application of GWO-based feature selection to CNN and LSTM architectures, enhancing the detection accuracy and efficiency of these architectures in comparison to the latest research.

1. INTRODUCTION

Malware, or malicious software, is intentionally written to damage digital devices. It is known by numerous other names such as Trojan horses, worms, viruses, bots and botnets, ransomware, adware, and spyware. Malicious users not only target individual computers and networks but also exploit security loopholes. The primary concerns of legitimate users are privacy and the leakage of personal information. Losses resulting from cyber fraud have continued to increase, reaching nearly twice those of conventional fraud [1][2].

Malware defense solutions are becoming more advanced as malware diversity increases. Previously, malware was written using primitive code that was easily identified. However, modern malware distributors tend to increase code complexity, making even sophisticated detection methods ineffective. A major challenge is next-generation malware designed to execute at the kernel level. Such malware can evade various security mechanisms such as firewalls and antivirus systems. It can remain permanently within a system or network, create multiple extensions, and attack targeted users.

Malware detection approaches vary and include several baseline techniques such as signature-based detection, heuristic-based detection, machine-learning-based detection, and deep-learning-based detection. Therefore, conventional methods such as pattern-matching techniques are no longer sufficient for malware detection. Malicious attacks on networks can severely disrupt services and operations, highlighting the need for intelligent detection systems capable of recognizing both known and unknown threats. Consequently, it becomes essential for security providers and end users to deploy complex security measures [3][4].

More novel methods have also been proposed based on different characteristics such as heuristics and model checking. These technologies incorporate specialized data-mining and deep-learning algorithms to identify malware. Innovative

*Corresponding author. Email: ms202310749@iips.edu.iq

technologies are generally more efficient than traditional approaches, and new methods have been designed to detect malicious software on mobile devices and computer networks, thereby preventing potential damage. This development responds to the growing and virulent nature of malware attacks. More recently, machine-learning (ML) and deep-learning (DL)-based research has been widely used to identify and analyze malware [5].

Optimization methods play a crucial role in increasing convergence speed and improving solution quality. Previous research has shown that combining Harmony Search and the Meerkat Clan Algorithm improves local search capacity and solution quality in complex scheduling problems. In line with such studies emphasizing the effectiveness of hybrid optimization techniques, this paper adopts the Grey Wolf Optimization (GWO) algorithm to optimize feature selection for malware detection. Optimization techniques are essential in various fields such as engineering, data science, and artificial intelligence. These algorithms aim to balance exploration of the search space with exploitation of promising solutions to accelerate convergence and achieve better outcomes [6][7].

This paper aims to develop an effective, efficient, and computationally less complex malware detection system for the Android environment using two deep-learning techniques, CNN and LSTM, to learn spatial and sequential patterns. Because the dataset initially contained a large number of features, direct training would have been computationally expensive and time-consuming due to redundancy and irrelevant information. To address this issue, feature selection was performed using the GWO algorithm, which minimizes the feature space while preserving the most meaningful attributes, thereby enhancing training efficiency and model generalization. In addition, data imbalance was addressed using the Synthetic Minority Oversampling Technique (SMOTE). The CNN and LSTM models were trained and tested on the CICMalDroid2020 dataset, a comprehensive benchmark comprising 17,341 samples of Android applications.

The remainder of this paper is organized as follows: Section 2 presents related work on malware detection. Section 3 describes the methodology, including dataset collection, data preprocessing, feature selection, and classification. Section 4 presents the results and discussion, and Section 5 concludes the paper.

2. RELATED WORK

In this section, related studies that have used deep-learning methods for malware detection are summarized. Progress in malware detection over the past few years has been based on the principles of machine learning (ML) and deep learning (DL), which have significantly improved prediction performance and accuracy in malware identification. Previously, many approaches relied on handcrafted features with statistical classifiers for detection and diagnosis; however, they often failed to capture complex patterns. In recent years, the use of Convolutional Neural Networks (CNNs) has increased considerably. According to [8], Android malware detection was addressed using a CNN-based deep-learning model built on static analysis of APK permissions. The authors extracted permission features from the *AndroidManifest.xml* file, converted them into a 10×20 pixel feature map, and applied an improved version of the LeNet-5 model. They utilized datasets from CICMalDroid2020, CIC-InvesMal2017, and CIC-InvesAndMal2019, totaling over 17,000 samples. The proposed CNN model achieved an accuracy of 90.07%, demonstrating strong efficiency and effectiveness in detecting unknown malware and showing suitability for Android-based mobile devices.

In [9], the authors presented an Android malware detection approach using a hybrid deep-learning model that combines static and dynamic analysis. The study employed two datasets: a static dataset derived from Drebin (15,031 samples with 215 features) and a dynamic dataset from CICMalDroid2020 (11,598 samples with 470 features). Both models were built using LSTM networks. The final static model achieved an accuracy of 92.94%, indicating strong performance and highlighting the advantage of hybrid modeling over standalone approaches.

In addition, [10] focused on analyzing mobile applications for malware detection by introducing a methodology that emphasizes the application design level rather than the code level. Reverse-engineering techniques were used to generate UML class diagrams of APK files, which were later transformed into OWL ontologies. Using SPARQL queries, specific malware types such as *Data_Send_Trojan* were identified. The authors relied on a dataset of 600 malicious Android applications from CICMalDroid2020. The proposed model achieved a precision of 92% and a recall of 91%, providing strong evidence of the effectiveness of modern code-analysis methods.

In [11], a hybrid malware detection method for Android applications was proposed by combining CNN techniques. Two datasets were used: Drebin and CICMalDroid2020, with benign samples collected from the Google Play Store. The proposed model achieved 92% accuracy and a 92.01% F1-score on the Drebin dataset, and 94.6% accuracy with a 94.6% F1-score on CICMalDroid2020, outperforming traditional machine-learning and fuzzy-logic-based methods while maintaining low computational complexity and parameter count.

In [12], a dynamic Android malware detection method based on analyzing system-call sequences was presented. The approach combines n-gram-based grouping of system calls with local TF-IDF weighting to capture both sequential relationships and feature importance. These features were then fed into a deep-learning model for classification. The system was evaluated on the CICMalDroid2020 dataset, achieving 91.06% accuracy and a 91.02% F1-score for multi-class

classification. Additionally, it demonstrated strong generalization capability, achieving 81.74% accuracy on the unseen 2019 Androzoo dataset.
In [13] targeted Android Operating System in malware detection and related information extraction using ViT Attention Mechanism. They employed AndroZoo and CICMalDroid2020 datasets in detecting malwares. It utilized the methodology of the ViT model (Vision Transformer and considered the attention mechanism to transform the DEX files into pictures and analyze them as images to recognize the malicious activity. although used the same dataset, their representation is image-based, whereas our model uses encoded features. This difference in representation may influence performance, so the comparison should be interpreted with caution. The accuracy of the proposed model was 80.27%, with an F1-score of 0.77, which proves the effectiveness of the approach compared with other methods in the same field.
In [14], the study focused on Android malware detection and classification using a hybrid deep-learning approach based on static features, system calls, and network flows from the CICMalDroid2020 dataset. For multi-class classification, the best-performing model based on static features using MLP achieved 94% accuracy, precision, recall, and F1-score. Meanwhile, the multi-view model combining system calls, network flows, and static features using Bi-LSTM + MLP + MLP and Bi-GRU + MLP + MLP achieved 83% accuracy, 83% recall, and up to 84% precision and F1-score. Further improvements could be made to multi-class categorization.
In [15], the authors presented an Android malware detection method by visualizing static features and classifying them using a CNN model. They used the CICMalDroid2020 dataset consisting of 17,341 applications and extracted static features such as permissions, intents, receivers, and services using AndroPyTool. These features were transformed into embedding vectors using BERT and then combined into 157×45-pixel images. The CNN model classified these images into benign or malware classes. The proposed approach achieved 91% accuracy, an F1-score of 89%, and a recall of 91%, demonstrating effective performance compared with several other CNN-based and vision-transformer models in the same domain.
Another study [16] proposed MAPAS, an Android malware detection system that leverages CNN to analyze API call graphs and extract common behavioral patterns in malicious applications. MAPAS outperforms the state-of-the-art MaMaDroid by classifying applications 145.8% faster while using ten times less memory. It also achieves 91.27% accuracy in detecting unknown malware, compared with MaMaDroid's 84.99%.

In [17] proposed a better Transformer-based malware detection model of Android in Internet-of-Vehicles environments. The proposed system combines an improved Transformer encoder with a recurrent neural network (RNN) and uses Grey Wolf GWO in the selection of the features and Snake Optimizer Algorithm (SOA) to optimize the hyperparameters. This hybrid structure can effectively be used to detect more complicated semantic and sequence patterns in malicious programs and hence be used to classify them more accurately. The dataset in the study was a collection of Android applications (APK files) consisting of benign and malicious applications, where behavioral and API-call features were extracted from the APKs to build the feature set for classification. Nonetheless, the paper observed that the computational cost and training overhead of the model are still issues that make the model difficult in real-time implementation.
In [18] introduced a framework of malware detection based on call-graphs was where a Hierarchical Attention Pooling Graph Neural Network is utilized to enhance the coverage of structural and semantic relationships in Android applications. The framework improves the identification of malicious activity-related behavioral patterns by modeling each app as a call graph and using multi-level attention mechanisms. The study used the Microsoft Malware Classification Challenge (BIG 2015) dataset from Kaggle contains nine well-known malware families and they collected 3368 unpacked malware samples from VirusShare and combined them with 6809 benign Windows 10 executables to form a binary detection dataset. The research established that hierarchical attention is a more effective way of enabling GNN models to distinguish between malicious and benign applications.
Another study [19] proposed a dynamic malware detection framework based on Dynamic Graph Neural Networks (DGNNs) which model API call sequences as evolving graphs to capture both temporal and structural behavioral patterns. They used the dataset consisting 42,797 malware samples and 1,079 goodware samples. The DGNN-based models achieved high accuracy, outperforming traditional machine-learning classifiers.
New study [20] focused on Android malware classification using a multimodal deep learning approach that combines image and graph representations. The technique fused predictions from CNNs and Graph Neural Networks (GNNs) using a late fusion strategy with a meta-classifier (MLP). The best-performing model, combining DenseNet121 and GIN, achieved an accuracy of 90.6%, an F1 score of 90.6%, and a recall of 90.6%, outperforming unimodal and earlier fusion models in terms of effectiveness and robustness for multi-class malware classification. TABLE I illustrates the summary of the related studies from study goal, dataset used, and limitation.
The reviewed studies can be grouped into two main categories based on their relation to the architecture proposed in this study: (A) the studies using models that are closer to the CNN -LSTM -Attention framework that is discussed in this paper and (B) the ones using different deep learning architectures. A number of studies have used the models of deep learning in line with our strategy. The analysis and visualization of features using CNN and neural networks was discussed in [8] and

[15] and DNN and hybrid models were presented in [9], [11] and [14]. Research such as [12] and [16] trained deep learning on sequences of system-calls or single feature descriptions. In other research works, different architectures are used fundamentally differently. Systems based on ViT- and Transformer-based and graph-based methods with the help of GNNs were introduced in [13] and [17], [18] and [19], respectively. Broader reviews including Transformer and GNN discussions [1], [3] also fall into this category.

TABLE I . THE SUMMARY OF THE RELATED STUDIES.

Author(s)/Years	**Methods/Techniques**	**Brief Description**	**Limitations**
Tang et al. (2021) [8]	CNN Modified LeNet-5	A CNN-based approach that detects Android malware using static analysis of app permissions extracted from APK files and classified via a modified LeNet-5 model.	lacks evaluation on large-scale real-time data or broader malware classes.
Wakhare (2021) [9]	LSTM (Static + Dynamic Models)	A hybrid Android malware detection approach using LSTM for both static (permissions, API, intents) and dynamic (system calls) features.	lacks evaluation against real-time unknown malware.
Aboshady et al. (2022) [10]	Ontological Semantic Environment	The study proposed a novel method for detecting malware in Android applications before the coding stage. APK files were reverse-engineered to UML models, converted into OWL ontologies, and analyzed using SPARQL to identify the malware.	Limited scalability and generalizability due to semi-manual process and focus on a single malware type
Atacak k et al. (2022) [11]	Hybrid CNN + ANFIS	Proposed a hybrid model that combines convolutional neural networks for feature extraction and ANFIS (Adaptive Neuro-Fuzzy Inference System) for classification, using permission information from Android APK files to detect malware	Limited to permission-based static analysis; lacks real-world deployment or integration with dynamic analysis.
Hung et al. (2022) [12]	TF-IDF + Deep Learning	A dynamic malware detection approach that analyzes Android system call sequences by extracting behavioral patterns using n-gram and TF-IDF, then classifies them using a deep neural network to distinguish between benign and malicious applications.	untested on real-time detection or larger-scale heterogeneous environments
Jo et al. (2023) [13]	ViT Attention Mechanism	The study developed multiple models, including Vision Transformers and deep learning approaches, for Android malware detection, with the ViT Attention Mechanism outperforming others in accuracy.	The study lacked comparison with other existing models for malware detection
Prayoga et al. (2023) [14]	Bi-LSTM + Bi-GRU + MLP (Multi-view DL)	The system employs a multi-view architecture combining Bi-LSTM or Bi-GRU with MLP networks to enhance multi-class Android malware classification performance through feature-level fusion.	Lower performance in multi-class tasks; high model complexity

Kiraz and Doğru (2024) [15]	CNN + BERT embeddings	A CNN-based Android malware detection approach using static features (permissions, intents, receivers, services), embedded with BERT and converted into images.	Relies only on static features, lacks evaluation in real-world environments and limited dataset scope
Kim et al. (2022) [16]	Deep Learning CNN	the study proposed a malware detection system using CNN model and static analysis of API call graphs extracted from benign and malicious Android apps	limited by static analysis constraints.
Almakayeel (2024) [17]	Improved Transformer + RNN, BGWO (Binary Grey Wolf Optimization), SOA (Snake Optimizer Algorithm)	Proposed an enhanced hybrid Transformer–RNN model for Android malware detection in IoV environments. The technique employs Grey Wolf Optimization in feature selection and Snake Optimizer Algorithm in hyperparameter optimization, resulting in high accuracy and better classification.	The model is very computationally intensive and might not be able to support real-time application in resource constrained systems.
Guo et al., (2025) [18]	Hierarchical Attention Pooling Graph Neural Network (HAPGNN)	Introduced the MalHAPGNN model that constructs and processes Android call graphs with hierarchical attention pooling to learn structural and semantic malware patterns in a more effectively. Obtained better detection rates than traditional GNN.	Computational expensiveness of graph construction and GNN training, and needs a lot of preprocessing and labeled data of high quality.
Kulkarni et al. (2025) [19]	Dynamic Graph Neural Networks (DGNNs)	Developed a DGNN-based framework to model temporal and structural API-call relationships to detect malware achieving high accuracy and F1-score.	Very expensive to compute, not scalable to real-time, relies on labeled data and cannot be easily understood.
Arrowsmith et al. (2025) [20]	DenseNet121 + GIN	a multimodal Android malware classification method combining binary image analysis using CNNs (e.g., DenseNet121) and function call graph (FCG) analysis using GNNs.	Limited to a small curated dataset; some misclassification remains for obfuscated malware

TABLE I illustrates the summary of the related studies from study goal, dataset used, and limitation. Based on the analysis of the related studies a number of limitations were identified: the reliance on static features, lack of powerful feature selection mechanisms and the limited generalization of the models to the datasets. In response to these challenges the proposed study suggests the use of two deep learning frameworks CNN and LSTM with each being used separately to identify the spatial and temporal patterns of malware data.
In addition, the GWO was incorporated to solve the inefficiency and redundancy of high-dimensional features and minimize the training time and enhance the generalization. Using the given methods on the CICMalDroid2020 dataset which offers a large and heterogeneous sample of Android malware the proposed system will mitigate the limitations of the previous studies and can provide more robust scalable and accurate malware detection.

3. METHODOLOGY

The method proposed in this study for Android malware classification involves a series of structured steps. First, steps such as giving class labels using encodings and scaling numerical features to prepare the data for learning. Following that a feature selection process is conducted using the enhanced GWO algorithm to identify the most informative subset of features and reduce computational complexity.

SMOTE is used on the selected features to balance the training data which has an imbalance class distribution. These enhanced and equally important characteristics are given as input to two deep learning models, a CNN and an LSTM to perform classification. Finally, Modes of evaluation applied to the prepared models are accuracy, precision, recall value, F1-score and time used for processing each test case. These stages are discussed in the relevant subchapters. Figure 1 shows how the whole pipeline works including preprocessing, feature selection and classification.

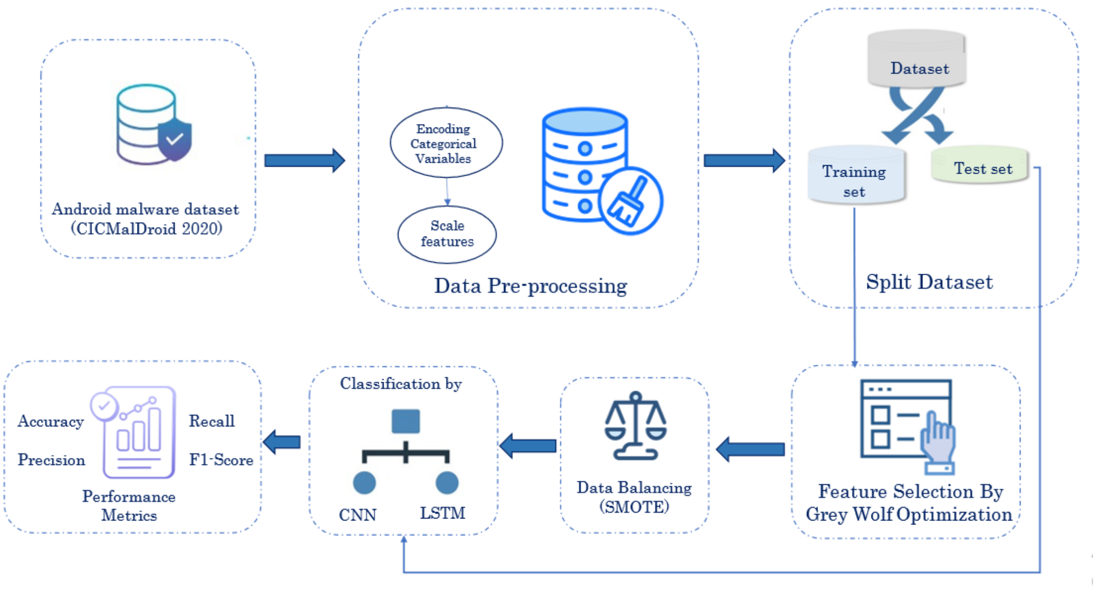


Fig. 1 . The proposed system for Android malware classification

3.1 Dataset Description

The CICMalDroid2020 dataset contains a wide range of Android malware and was developed by the Canadian Institute for Cybersecurity. can download from link https://www.unb.ca/cic/datasets/maldroid-2020.html Its consists a total of 17,341 Android application samples collected from multiple sources between December 2017 and December 2018. After dynamic analysis using the Copper Droid framework, 11,598 samples were successfully analyzed and labeled into five categories: Adware, Banking malware, SMS malware, Riskware, and Benign as show in figure 2. For each APK 470 static and dynamic behavioral features were extracted, including system calls, binder calls, and composite behaviors. In addition to the main feature set. the dataset also includes multiple CSV file formats: one with 470 features another with 139 features focusing solely on system calls and an extended version with 50,621 features capturing detailed static attributes such as permissions, intent actions, APIs, services and more [21,22]. Each malware category is briefly described as follows in Figure 2.

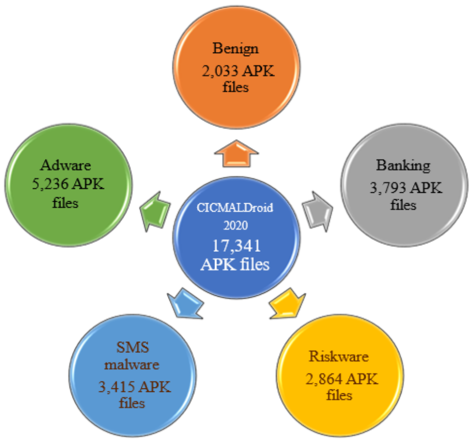


Fig. 2 Categories of groups comprising the Dataset CICMalDroid 2020

TABLE II . CICMALDROID 2020 DATASET DESCRIPTION

Families	No. of Apps	Description
Adware	1,253	hides in legitimate apps to repeatedly display unwanted ads, and may steal personal data
Banking	2,100	Authenticates their internet banking services to steal sensitive data.
Riskware	2,546	any legitimate app that, if exploited, can cause harm.
SMS	3,904	Conducts cyber-attacks uses the SMS service to send, intercept, or steal messages
Benign	1,795	non-malicious applications verify by Virus Total

Further information regarding the dataset presented TABLE II .it refers to the number of samples that were successfully analyzed using the Copper Droid framework, with their features extracted and documented in CSV files. These samples are considered usable for training and testing in machine learning and deep learning techniques experiments.

3.2 Preprocessing

Preprocessing forms an essential part of every intelligent system using machine learning or deep learning techniques. It aspires to clean up data to become more ready for training, which ensures achieving more effective and accurate results. This stage includes next steps, the encoding categorical variable and scaling features. The dataset was divided into two subsets 80% was used in training and the remainder 20% was used in testing. (SMOTE) method was also implemented on the training subset only to solve the problem of class imbalances and test subsets integrity. In addition to this split, there was a (3-fold stratified cross-validation) procedure that was applied to the feature selecting and model optimization steps. Cross-validation was only carried out based on the 80 % of the training subset to make a solid estimate of model performance and also to ensure there is no over-fitting to a particular split. The independent 20% test set was never used during training, feature selection, SMOTE oversampling, or cross-validation. This methodology will avoid information leakage and ensure generality of the model.

A. Encoding Categorical Variables

Data Encoding is important to perform Data Encoding before starting to build your Deep and machine learning model. Without this step, algorithms may not be able to deal with textual data correctly. his process to do this means turning data from text or categories into numbers so that algorithms have a way to access it. Because few learning algorithms understand text or non-numerical labels, changing them to numbers allows them to work together better. In addition, it contributes to reducing model complexity and reduces the time needed for training the model. This stage consists of two steps:

1. An integer is associated with each distinct malware type (Adware, Trojan,etc.) As an example, Adware = 0, Riskware = 1, Banking = 2. This step formulates a one-to-one correspondence between string labels and integers that the labels can be comprehended in the model.
2. Once the encoding has taken place, the mapping between each original class name to its resultant integer value are shown. Also, the size of the unique encoded classes is verified and all types are represented properly.

B. Scale Features

In order to make features contribute equally to learning process and to fasten the convergence of the model. the feature scaling was used by using the Standardization method. This does play a crucial role in deep learning and machine learning algorithms where the values of features matter. Standardization approach also does the same by putting the means of each feature in the dataset to zero and the standard deviations to one. Hence, all the values will be between [0,1]. All feature values undergo the process of scaling. Except for the column with the labels of the classes is not changed.

The standardization is done as follows: standardization of each feature converts them all to a mean of 0 and a standard deviation of 1 as of (1):

$$Z = \frac{(x - \mu)}{\sigma} \tag{1}$$

Where:

x is the original feature value

μ is the mean of the feature in the training set

σ is the standard deviation of the feature in the training set

Z is the standardized value

3.3 Feature Selection by GWO Algorithm

In malware detection tasks, datasets can have a very large number of features, many of which are redundant or irrelevant. Direct use of all of these features not only exponentially increases the computational cost and training time, but also the probability of overfitting, since the model can be taught noise rather than meaningful patterns. In overcoming this difficulty, feature selection is used to decrease the number of dimensions of the data set keeping the most informative elements. This paper was conducted using GWO algorithm, to choose the best subset of features of the CICMalDroid2020 dataset in order to optimize the malware performance. as show in Figure 3 The following steps illustrates the proposed feature selection process.

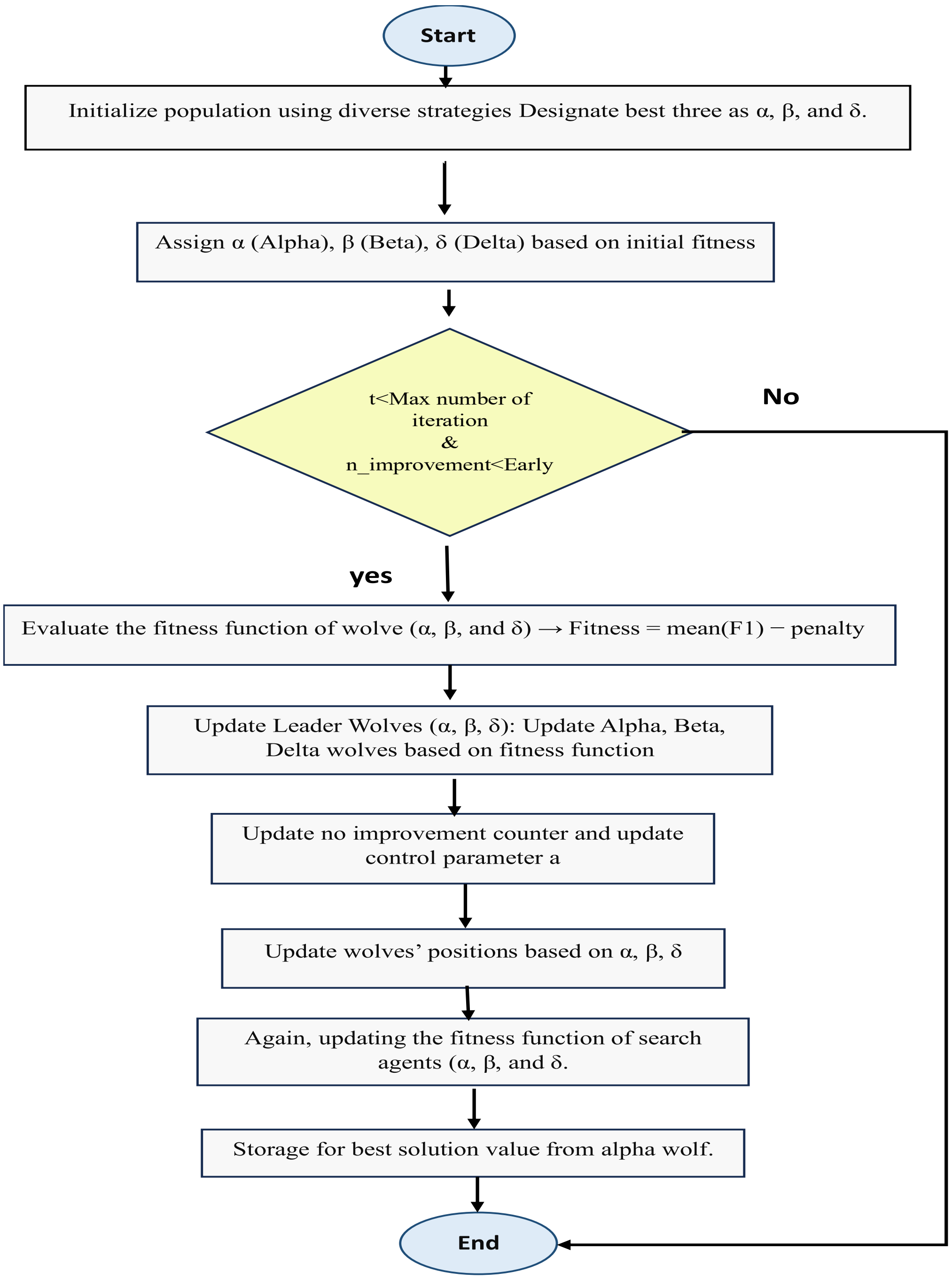

Start
Initialize population using diverse strategies Designate best three as α, β, and δ.
Assign α (Alpha), β (Beta), δ (Delta) based on initial fitness
t<Max number of iteration & n_improvement<Early
No
yes
Evaluate the fitness function of wolve (α, β, and δ) ⟶ Fitness = mean(F1) − penalty
Update Leader Wolves (α, β, δ): Update Alpha, Beta, Delta wolves based on fitness function
Update no improvement counter and update control parameter a
Update wolves' positions based on α, β, δ
Again, updating the fitness function of search agents (α, β, and δ.
Storage for best solution value from alpha wolf.
End

Fig. 3. Flowchart for proposed GWO algorithm

The GWO algorithm for feature selection focuses on nature and follows the nature hierarchy as well as leadership mechanism of the grey wolves in nature. Within the scope of the present research the GWO algorithm is used to identify an optimum subset of features that would help increase publication accuracy and reduce redundancy. The feature selection process in this study was implemented using the GWO . can be described in the following steps:

1. First, the algorithm simulates the hunting behavior of grey wolves in that it produces a wide range of candidate feature subsets.
2. Fitness Evaluation: Fitness function defines how well the selected feature subset is chosen. The evaluation of these subsets relies on a fitness function which is an evaluation of a balance between classification and feature compactness by cross-validation using stratification on a lightweight Random Forest classifier. The following procedure is executed for each candidate:

   A lightweight Random Forest classifier that is implemented to analyze the capacity of the selected features to predict.

   Cross-Validation Based Performance Evaluation. The cross-validation approach (Stratified K-Fold Cross-Validation (K=3)) is used so that the estimation of performance can be reliable. This is trained in one of the folds with two parts and the other part used to test the model. In each fold, the model is trained on two parts and tested on the third. Fitness Score Computation Then, is computed in (2):

   $$fitness = mean(F1 - score) \ - penalty \tag{2}$$

   where:

   mean(F1-score): refer this represents the average performance of the classification model when using the features selected by the current wolf. It is calculated with a very light Random Forest classifier and Stratified k-fold cross-validation to promote robust assessment.

   Penalty: This term serves as a regularization factor. The objective is to minimize the fitness score in a case when the wolf had a considerable number of chosen features. Thus, a penalty is imposed on the fitness function which is reduced as the number of features increases.
3. The optimizer also examines a change in the best fitness score at the conclusion of every iteration. In case there is no improvement after 3 consecutive runs (early stopping = 3) the algorithm stops to prevent unnecessary calculation. The control parameter a initially begins with a = 2 and decreases in a linear manner to = 0 during 15 iterations. The optimizer involves eight wolves (n wolves = 8) and trains each subset of features on the weighted F1-score with 3-fold cross-validation.
4. Once the fitness of all the wolves has been evaluated the top three solutions are designated as α, β and δ representing the best, second best and third best solutions respectively. In the event that a new wolf has a better fitness the leader positions are revised accordingly and the optimal fitness value is stored to be compared later.
5. In this step Algorithms will keep track of the progress by comparing the current best-fitness and the previous best-fitness. In case no improvement is made the (no improved) counter is increased. At the same time the control parameter (a) is reduced linearly between 2 to 0 with respect to the number of iterations.
6. In this step Each wolf updates its position (feature subset) by referencing the three leaders. For each feature dimension, three directional vectors (X_1, X_2, X_3) are computed based on α, β and δ. The new position is determined as a weighted combination in (3):

   $$Xnew \ = w\alpha \ \cdot X_1 \ + w\beta \ \cdot X_2 \ + w\delta \ \cdot X_3 \tag{3}$$

   where:

   $w\alpha, w\beta \ , w\delta$ the weights are computed based on the fitness scores of

   $X_1 \, , X_2, X_3$ represent the suggested positions for a given feature.
7. Once the position is updated each of the fitness values of all the wolves is recalculated using the new feature subsets. The α, β and δ positions update again in the event that better solutions have been found during this step.
8. Finally, the optimal feature subset is extracted from the (α) wolf, which represents the best-performing solution. This procedure will make sure that the most pertinent and non-redundant features are chosen and this will enhance training efficiency and model generalization.

3.4 Class Imbalance

Class imbalance is one of the main issues that have to be dealt with during the malware classification tasks when the samples are unequally distributed among the target classes. The number of benign applications in CIC-MalDroid2020 is by far larger than any malicious category (SMS malware or Riskware). Such an imbalance can cause biased training of the model, such that the classifiers prefer the dominant class and ignore the minority cases, which will diminish the detection performance on minority malware types.

In order to deal with this problem, we used Technique (SMOTE), which was used to oversample the minority classes by creating random samples of these classes. finding their nearest neighbors, and then new samples are created that lies on the dividing line between the selected sample and its nearest neighbors, as illustrated in Figure 4.

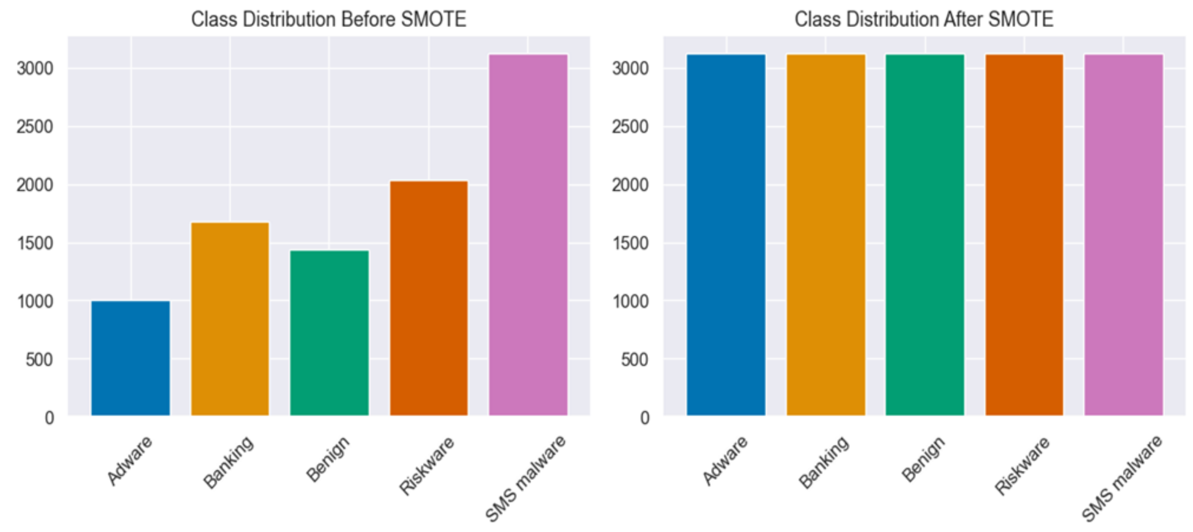


Fig. 4. class distribution before and after SMOTE

The synthetic samples are produced based on SMOTE algorithm, The newly synthesized instance Xnew is computed as:

- For each minority class sample xi, select one or more of its k-nearest neighbors xzi.
- Create a synthetic sample $xnew$ as of (4):

$$xnew = xi + \delta \cdot (xzi - xi) \quad (4)$$

Where:
xi is a sample from the minority class.
xzi is one of its nearest neighbors.
δ is a random value between 0 and 1.

3.5 Classification Modeling

During this phase, various scenarios of the DL will be used as experimental set-ups to produce the best results in terms of accuracy and efficiency. Although CNN has shown to be effective in modeling spatial interrelationships among features, LSTM is good at modeling temporal or sequence dependencies. These will be detailed before delving into the proposed model, where satisfactory results were achieved, as explained in subsequent sections.

A. The Proposed CNN-GWO model

In this paper, the developed a CNN model in its one-dimensional reduced form (1D-CNN) to distinguish between malware samples and categorize them into the corresponding category depending on the extracted features. This CNN architecture was particularly selected because of its ability to automatically learn high dimensional deep and hierarchical representations on input features, which plays a critical role in identifying malware since, in this case, patterns are usually complex and non-linear. The model was utilized in two circumstances: one was performed on the raw feature set, whereas the second one was considering a feature selection (FS) mechanism that reduces the dimensionality by minimizing the negative impact on the performance. The suggested 1D CNN structure built up of a hierarchy of convolutional block systems that aim at obtaining more abstract features of the input vectors. Reshape layer reorganizes the input into a one-dimensional format appropriate to use with convolutional layers.

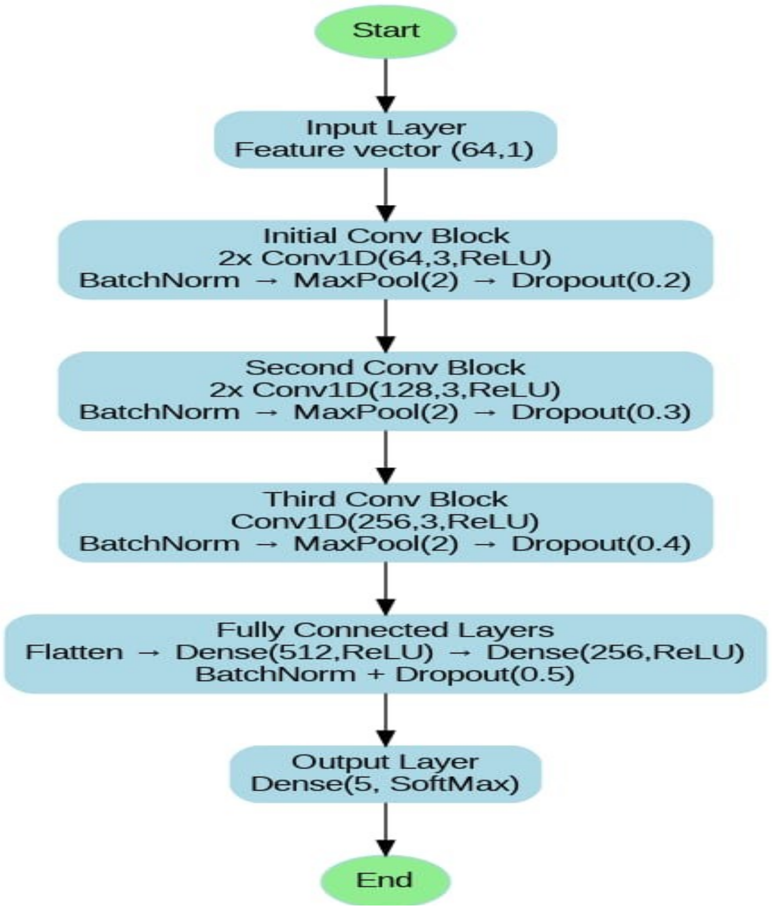


Fig. 5. The Architecture of the Proposed CNN-GWO Model

The Figure 5 as show the architecture of proposed 1D CNN model in malware classification. The proposed CNN-GWO architecture as show Figure 5, The model begins with an input layer that stipulates the structure of the data and three convolutional blocks that extract hierarchical features. Every convolutional block uses a series of Conv1D, ReLU activation and Batch Normalization layers to stabilize the learning then MaxPooling and Dropout layers to decrease the dimensions and avoid overfitting. This third block further enlarges the size of the filter to further establish spatial correlations among the data. Following the convolutional steps, the resulting output is flattened and taken to fully connected layers, which sharpen the learned representations. The last Dense layer that has a SoftMax activation function generates the probability of the classes of five different malware types. The proposed architecture represents a further combination of the CNN model with the GWO algorithm, which improves the detection rate and reduces excess and model convergence time The architecture of proposed CNN-GWO model architecture model in malware classification is given in the algorithm 1.

Algorithm (1): CNN -GWO Architecture for Malware Classification

```
Input: Feature vector of shape (64, 1)
Output: Class prediction (5 classes)
1. Input Layer:
  - Define input layer with shape (features, 1).
2. Initial Convolutional Block:
    -   Apply two Conv1D layers with (64 filters, kernel size 3, ReLU, same padding)
    -   Each Conv1D layer is followed by Batch Normalization.
    -   Apply MaxPooling1D (size = 2).
    -   Apply Dropout (rate = 0.2).

2. second Convolutional Block:
    -   Apply two Conv1D layers with (128 filters, kernel size 3, ReLU, same padding)
    -   Each Conv1D layer is followed by Batch Normalization.
```

- Apply MaxPooling1D (size = 2).
- Apply Dropout (rate = 0.3).

2. Third Convolutional Block:
- Apply one Conv1D layer with (256 filters, kernel size = 3, ReLU activation, same padding).
- Followed by Batch Normalization, MaxPooling1D (pool size = 2), and Dropout (rate = 0.4).

3. Fully connected layer:
- Flatten the output.
- Apply Dense layer (512, ReLU) followed by Batch Normalization and Dropout (0.5).
- Apply Dense layer (256, ReLU) followed by Batch Normalization and Dropout (0.5).
- Apply the final Dense layer with 5 units and SoftMax activation for classification.

End

B. The proposed LSTM-GWO model

LSTMs are significantly more effective in modeling such dependencies, even non-linearity or subtle dependencies which makes the approach highly effective at identifying highly evasive forms of malware. The reason why we have chosen the LSTM model in this study is that it has been shown as the capable to learn properly with temporal data and to generalize sufficiently across malware families. The proposed architecture uses LSTM is provided by reshaping the main input vector into the form relatable with the input to LSTM layers, where each feature will represent a single time step.
The Figure 6 as show the architecture of LSTM model in malware classification. The proposed LSTM-GWO architecture as show Figure 6, The model starts with a layer of input that takes the feature or size (64) followed by two sequential LSTM layers to capture temporal dependencies in the data. The information is then processed by a Bidirectional LSTM layer to process information both forward and backward. Then, the most significant features are brought to the fore by an attention mechanism. The output is sent through fully connected layers with ReLU activation, L2 regularization, Batch normalization and dropout to improve generalization. Lastly, a SoftMax layer generates the classification output of five classes and the GWO optimizes model parameters to achieve the best performance

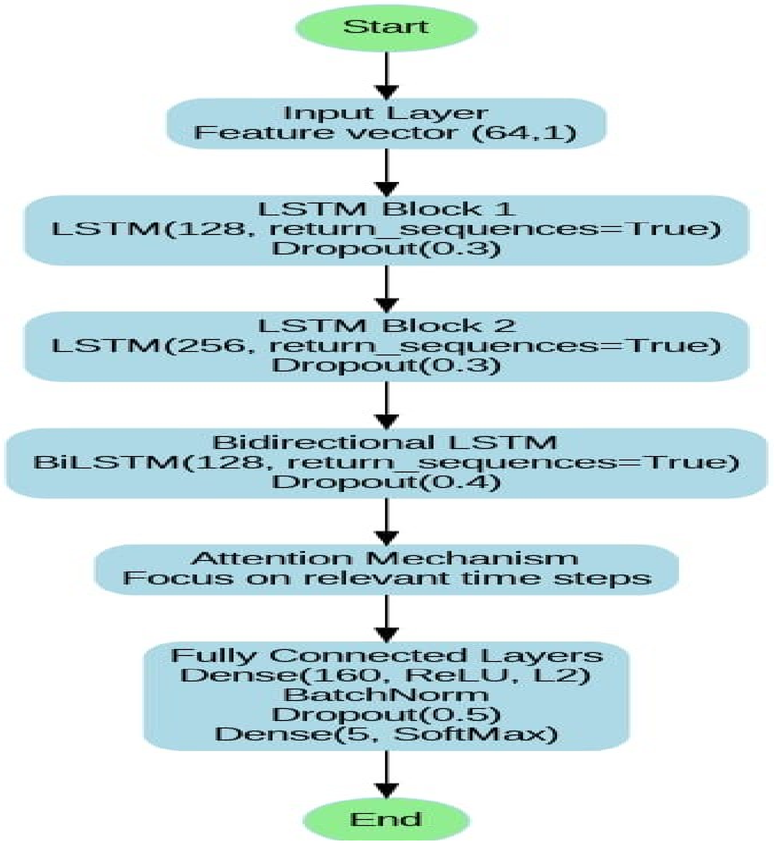


Fig. 6. The Architecture of the Proposed LSTM-GWO Model

The architecture of proposed LSTM-GWO model architecture model in malware classification is given in the algorithm 2.

```
Algorithm (2): LSTM-GWO Architecture Malware Classification
Input: Feature vector of shape (64, 1)
Output: Class prediction (5 classes)
Input Layer:
    -   Define input layer with shape (64, 1)

LSTM Block 1:
    -   Apply LSTM layer with 128 units and return_sequences=True
    -   Apply Dropout with rate 0.3

LSTM Block 2:
    -   Apply LSTM layer with 256 units and return_sequences=True
    -   Apply Dropout with rate 0.3

Bidirectional LSTM:
    -   Apply Bidirectional LSTM layer with 128 units and return_sequences=True
    -   Apply Dropout with rate 0.4

Attention Mechanism:
    -   Apply attention layer to focus on the most relevant time

Fully Connected Layers:
    -   Apply Dense layer with 160 units, ReLU activation, and L2 regularization
    -   Apply Batch Normalization
    -   Apply Dropout with rate 0.5
    -   Apply final Dense layer with 5 units and SoftMax activation to predict class probabilities

End
```

3.6 Evaluation Metrics Methods

For the purpose of evaluating every one of the models several measures of the assessment were used such as accuracy, precision, F1-score, and recall. The parameters of the confusion matrix were employed in assessing the outcomes of the proposed system, in which Figure 7 shows the values of true positive (TP), true negative (TN), false positive (FP), and false negative (FN).

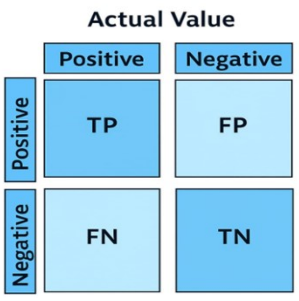


Fig.7. the confusion matrix used in the classification process

Evidence of the proposed system was measured by using four parameters as the following points:

1. *Accuracy:* The performance measure of accuracy is quite broadly used and it has been described as the division of the number of samples that have been correctly classified to an overall number of samples [17]. based on (5).

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \times 100\% \tag{5}$$

2. *Precision:* the has been computed as a proportion between the number of positive results that have been accurately predicted and the number of positive predictions [2]. Precision can be computed based on (6).

$$Precision = \frac{TP}{TP+FP} \tag{6}$$

3. *Recall:* The recall may be denoted as a degree of the completeness where measurements are made on the proportion of the positive observations that have been accurately predicted with all the observations in actual class. the true positive instances divided by the total positive instances [2]. This parameter may be calculated in (7).

$$Recall = \frac{TP}{TP+FN} \tag{7}$$

4. **F1 score**: in malware classification provides a consistent mean of precision and recall and a balanced indicator of the accuracy of the model used in detecting malicious App, this parameter may be calculated in (8).

$$F1\_Score_i = 2 \times \frac{Precision \times Sensitivity}{precision + Sensitivity} \tag{8}$$

4. RESULT AND DISCUSSION

This section contains the experimental results of the proposed malware detecting models and its analysis. Data preprocessing, selection of features by means of QWO algorithm and a CNN , LSTM classification strategy. The CNN and LSTM architectures were tested against different measures, which were accuracy, loss, and the confusion matrix. The Google Colab platform was used to conduct the experiments with Python 3.12.12 and the TensorFlow 2.19.0 framework According to the results the deep learning methods prove to be efficient in recognizing and categorizing various malicious software types with a high accuracy.

4.1 Encoding Categorical Result

All the categorical labels in the target column (i.e. Class) were converted to numerical codes. The encoding process successfully converted the five distinct malware classes into integer labels ranging from 0 to 4. The analysis confirmed that all five classes were accurately encoded and preserved without any data loss. The following TABLE IV shows the Encoding Categorical Variables for five class.

TABLE IV . ENCODING CATEGORICAL VARIABLES ERESULT

Classes	Encoded
Adware	0
Banking	1
Benign	2
Riskware	3
SMS malware	4

4.2 Scale Features Result

This preprocessing made each feature have a standard deviation of 1 and a mean of 0. The impact of standardization, the scaling effect, was confirmed as the distribution of the features was observed to lie within a normalized range after the transformation occurred. This finding greatly enhanced the stability and performance of the training process by making the model converge quicker and not bear bias on features that have larger values. The following TABLE V shows the Summary of Pre-processing Results.

TABLE V . SUMMARY OF PRE-PROCESSING RESULT

Metric	Before Pre-processing	After Pre-processing
Total Samples in Dataset (Before Split)	11,598	11,598
Label Encoding of Target Variable	Adware, Banking, Benign, Riskware, SMS malware (text)	0, 1, 2, 3, 4 (numeric)
Mean of Features (example before normalization - first 5 features)	5.59702522e+01	-6.50960687e-18
Standard Deviation of Features (example before normalization - first 5 features)	4.28508307e+02	1

4.3 Feature Selection Results

In this study, the GWO algorithm was used in selecting the features in this study. The overall goal of GWO was to eliminate those features that were irrelevant or redundant and hence increase the global performance in the classification and achievement in decreasing the complexity of the models. The dataset included 470 features then, by using GWO its feature set was minimized to that containing the most informative elements (e.g., 110 features), hence the dimensionality became significantly lower. This led to quicker training of the models, fewer chances of overfitting and better able accuracy of the classification. The GWO algorithm was executed with 8 wolves and 15 iterations, optimization reached the end at the 10 iteration whose best score was fitness 0. 9301.The TABLE VI Summarizes the feature Selection results including the number of wolves, total iterations, best score, number of selected features, and execution time (488.62 seconds).

TABLE VI . THE GWO ALGORITM HYPERPARAMTER

Item	**Value**
Number of wolves	8
Total iteration	15
Iteration at which best score was achieved	10
Best fitness score	0.9301
Number of selected features	110
Feature Selection Time	488.62 sec.

In addition, the dimensionality reduction effect of GWO can be represented by Figure 8 which indicates a reduction in the number of features of 470 to 110. the feature selection process done under the guidance of the GWO algorithm was critical in the dimensionality reduction without the performance of the model being spoilt as the model concentrated on the most relevant features. This methodology not only improves accuracy but also provides a faster processing time.

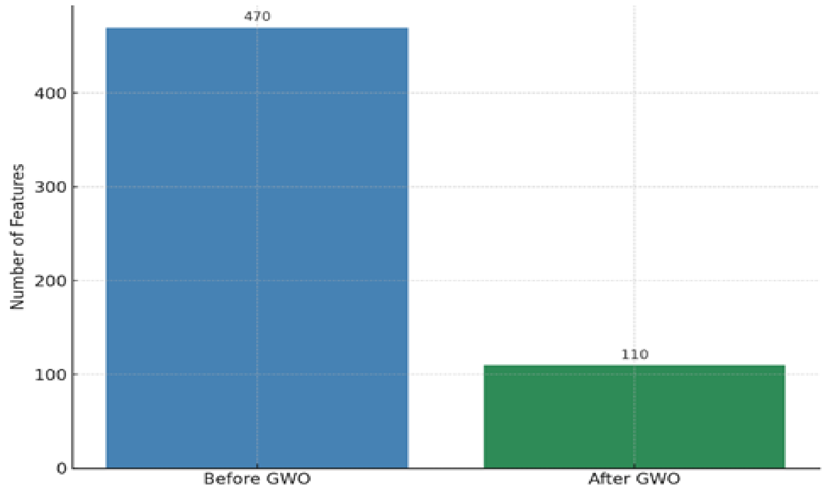


Fig.8. Number of Selected Features Before and After GOW

4.4 Classification Result

After preparing the data and passing through the pre-processing stages, including steps such as data balancing and others. Finally, the feature selection technique was implemented. The data is now ready for classification.

4.4.1 Results of the proposed CNN-GWO model

The selected features obtained using the GWO algorithm were used to train the CNN model. Figure **9** shows the training and validation accuracy and loss graphs of the CNN model across 50 epochs. The model had a fast trajectory of improvement as can be observed in the accuracy plot as shown the model took the first epochs to exponentially improve with the validation accuracy almost reaching 100% after the first ten epochs. The performance gradually increased and stabilized around 96.97%, around later epochs, which is evidence to the fact that the model obtained high generalization potential without severe over-fitting.
Equally, the loss graph shows that the downwards trend is sharp between the initial training phases falling off a scale of above 2.0 to below 0.3 between the 1st and 10th epoch. Training and validation losses gradually declined to the values of approximately 0.1 during the last epochs. The similarity plot of the training and validation curves in both plots indicates the CNN model had a well-balanced learning process, as it managed to achieve the lowest levels of training the loss functions without reducing the levels of predictive accuracy during the test.
These findings point to the stability and robustness of the proposed CNN architecture, once again ascertaining the worth of feature selection procedure in dimensionality reduction and performance boosting of the model.

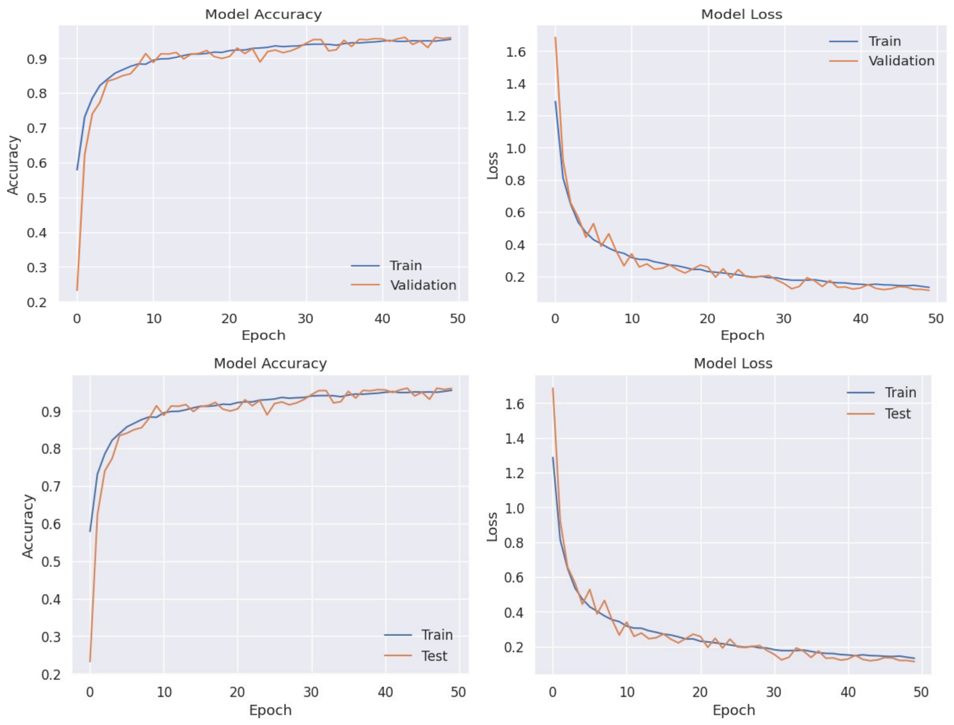
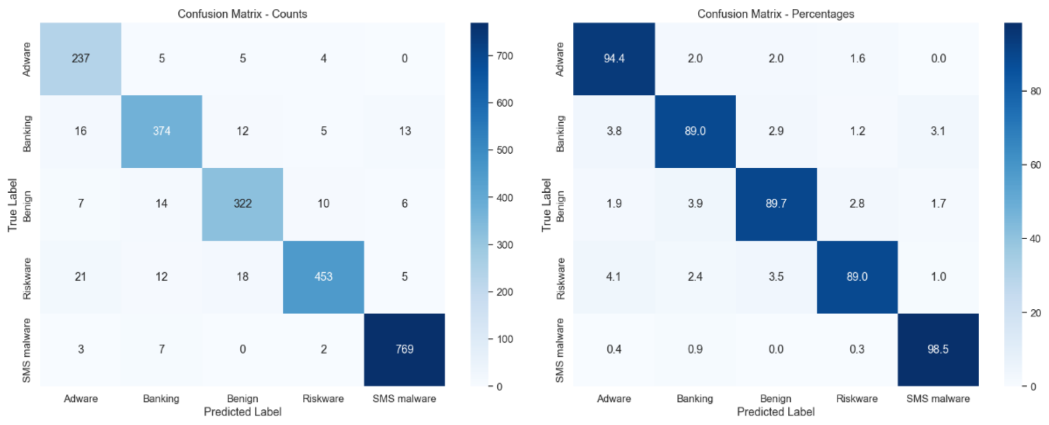


Fig. 9. Model accuracy and loss for proposed CNN-GWO model

The Figure 9 show the result of training and the test shows that the model has a consistent and predictable performance in all training epochs. The classification accuracy of the training and the test data showed an increased rate of increase at the initial stages, which points to the fact that the model has a high capacity to learn the important patterns in the data. Moreover, the values of the loss continued to drop gradually and to low values, which indicated the performance of optimization process and the efficiency of the learning process. The high correspondence between the training and test curves is a clear indication that the model is not overfitting, meaning that it has obtained a proper balance between the training data and its good generalization to the unseen data.



Confusion Matrix - Counts

True \ Predicted	Adware	Banking	Benign	Riskware	SMS malware
Adware	237	5	5	4	0
Banking	16	374	12	5	13
Benign	7	14	322	10	6
Riskware	21	12	18	453	5
SMS malware	3	7	0	2	769

Confusion Matrix - Percentages

True \ Predicted	Adware	Banking	Benign	Riskware	SMS malware
Adware	94.4	2.0	2.0	1.6	0.0
Banking	3.8	89.0	2.9	1.2	3.1
Benign	1.9	3.9	89.7	2.8	1.7
Riskware	4.1	2.4	3.5	89.0	1.0
SMS malware	0.4	0.9	0.0	0.3	98.5

Fig .10. Confusion matrix of proposed CNN-GWO model with five class

The Figure 10 the performance of the presented model is presented in terms of confusion matrices in absolute count (left) and percent (right). The outcomes show good levels of classification accuracy and most of the samples are correctly predicted. Percentage-based matrix on the right shows that the model has a high level of discrimination capacity since the accuracy predicting each class is above 90%, which proves the effectiveness of the given approach in providing excellent classification results.

TABLE VII. DETAILED CLASSIFICATION REPORT FOR CNN AND PROPOSED CNN-GWO MODEL

Class	**CNN**				**Proposed CNN-GWO**			
	Accuracy	**Precision**	**recall**	**f1-score**	**Accuracy**	**Precision**	**recall**	**f1-score**
Adware	83%	70%	83%	76%	92%	83%	94%	89%
Banking	89%	81%	89%	85%	88%	91%	89%	90%
SMS	93%	98%	93%	96%	99%	97%	98%	98%
Riskware	87%	90%	87%	88%	89%	96%	89%	92%
Benign	84%	90%	84%	87%	89%	90%	90%	90%
accuracy	88%	88%	88%	88%	93%	93%	93%	93%
macro avg	86%	87%	86%	86%	92%	91%	92%	92%
weighted avg	89%	88%	89%	89%	93%	93%	93%	93%

The classification report is shown in TABLE VI, which was carried out after the trained model has been evaluated on its independent test set.

TABLE VIII. RESULT FOR CNN AND PROPOSED CNN-GWO MODEL

	Without grey wolf CNN	**With grey wolf CNN**
Time train Inference Time	234.43 seconds. 0.9309 seconds	440.85 seconds 0.6812 seconds
Accuracy	88%	93%
Precision	88%	93%
Recall	88%	92%
F1-Score	88%	92%

TABLE VIII shows the results of comparing the performance of the model of CNN with and without the feature selection. It can be observed that after feature selection all metrics improved significantly with the best metric being the accuracy, precision, recall, and F1-score which increased to almost 93% in the results. The training time was reduced by a little extent whereas the inference time went down. Such results emphasize that the feature selection process not only increase the predictive accuracy of the model but also made it more efficient in inference.

4.4.2 Results of the proposed LSTM-GWO model

The features that were selected by means of the GWO algorithm were used to train the LSTM model. The Figure 11 shows training and validation accuracy and loss curves of the LSTM model throughout the 80 epochs. An accuracy plot indicates a rapid increase of accuracy in the early epochs with the validation accuracy being sharply hiked up even in the first 10 epochs and nears 90 %. Performance kept improving after each epoch, reaching its optimal level of 96.97 % in the later times showing that the model had a good generalization capability with minimum overfitting. Similarly, the loss plot shows a steeply decreasing trend during the first stages, with the numbers shifting to the left by more than twofold, and to the right by 0.3, by the 10th epoch. Both train and validation losses slowly dropped to about 0.1 at the end epochs.
The fact that the training and validation curves are co-aligned in terms of accuracy and loss indicates that the LSTM model at hand had an accurate and stable learning process. These results reinforce the effectiveness of the model architecture and the feature selection process in enhancing performance and reducing dimensionality.

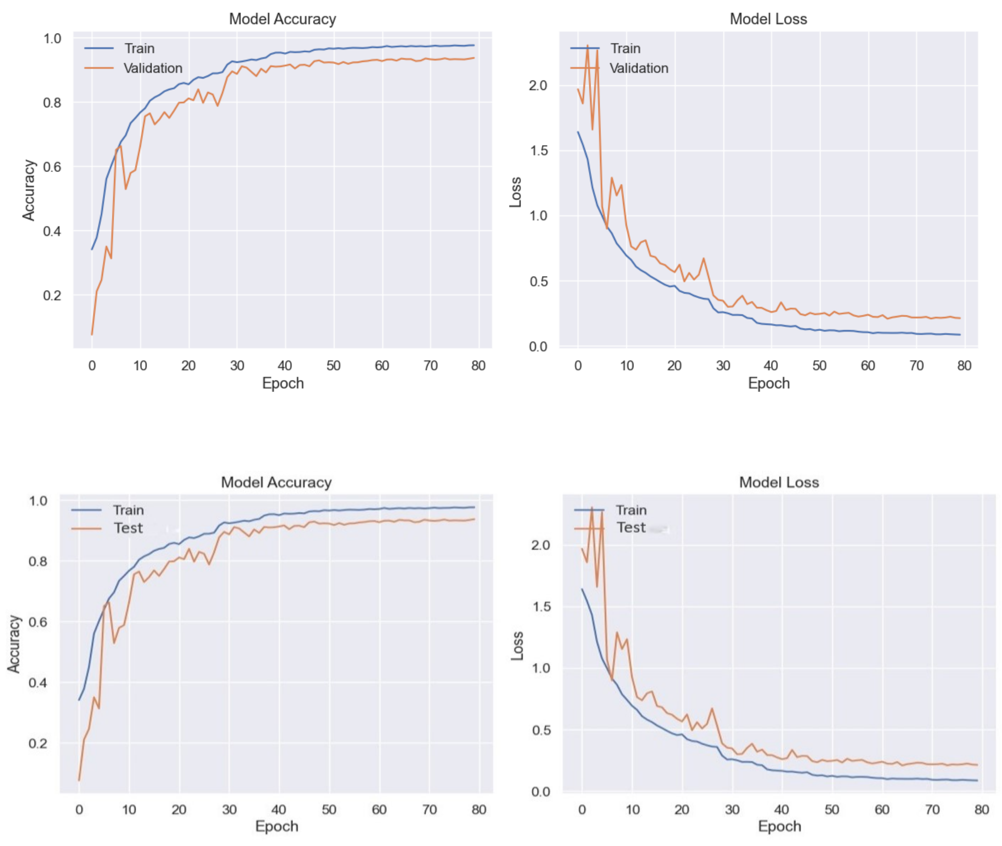


Fig. 11. Model accuracy and loss for Proposed LSTM-GWO mode

Figure 11 show the result of training and the test shows that the model has a consistent and predictable performance in all training epochs. The classification accuracy of the training and the test data showed an increased rate of increase at the initial stages, which points to the fact that the model has a high capacity to learn the important patterns in the data. Moreover, the values of the loss continued to drop gradually and to low values, which indicated the performance of optimization process and the efficiency of the learning process. The high correspondence between the training and test curves is a clear indication that the model is not overfitting, meaning that it has obtained a proper balance between the training data and its good generalization to the unseen data.

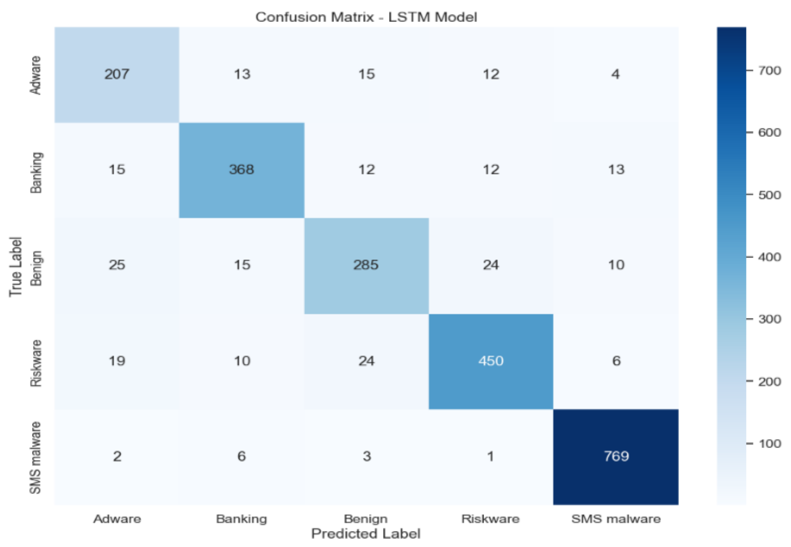


Fig.12. Confusion matrix of proposed LSTM-GWO model with five class

The Figure 12 shows the absolute count version of a confusion matrix representing the performance of the proposed LSTM model. The outcomes show the successful classification accuracy since the majority of the samples are accurately classified as per the various classes. The matrix demonstrates that model has a good accuracy in distinguishing among types of malwares, with scorers being especially high in such types of malwares as SMS malware and Riskware. These findings indicate how empirical the proposed method is in providing trustworthy and valid results when classifying.

TABLE IX. DETAILED CLASSIFICATION REPORT FOR LSTM AND PROPOSED LSTM-GWO MODEL

	LSTM				**Proposed LSTM-GWO**			
Class	**Accuracy**	**Precision**	**recall**	**f1-score**	**Accuracy**	**Precision**	**recall**	**f1-score**
Adware	84%	78%	84%	81%	82%	77%	82%	79%
Banking	86%	85%	86%	88%	87%	89%	87%	88%
SMS	97%	96%	97%	96%	98%	95%	98%	97%
Riskware	88%	91%	88%	89%	88%	90%	88%	89%
Benign	80%	86%	80%	83%	79%	84%	79%	81%
accuracy	89%	89%	89%	89%	90%	90%	90%	90%
macro avg	87%	87%	87%	87%	87%	87%	87%	87%
weighted avg	89%	89%	89%	89%	89%	89%	89%	89%

The classification report is shown in TABLE IX, which was carried out after the trained model has been evaluated on its independent test set.

TABLE X. RESULT FOR LSTM AND PROPOSED LSTM-GWO MODEL

	Without grey wolf LSTM	**With grey wolf LSTM**
Time train Inference Time	5604.55 seconds 4.6959 seconds	7654.66 seconds 6.1339 seconds
Accuracy	89%	90%
Precision	89%	90%
Recall	89%	90%
F1 Score	89%	90%

TABLE X shows the comparisons of the LSTM model performance with and without the feature selection. One can note that following feature selection, the evaluation metrics were all deemed to have improved slightly; the accuracy and precision were raised to 90% alongside recall and F1-score. Training time was enhanced by 1050 seconds with the inference time rising by 1.438 seconds as well. These result shows that despite the minor increase in the computational expenditure. Feature selection activity helped in modifying the model predictive performance and created better classification results.

TABLE XI. RESULTS OF THE PROPOSED MODEL AND OTHER DL MODELS

Algorithm	**Accuracy**	**Precision**	**recall**	**f1-score**
CNN	88%	88%	88%	88%
LSTM	89%	89%	89%	89%
the Proposed LSTM-GWO model	90%	90%	90%	90%
the Proposed CNN-GWO model	93%	93%	92%	92%

TABLE XI. shows the results related to the classification report (accuracy, Precision, and F-score) for all algorithms that were conducted. Experiments have proven that the proposed model outperforms with accuracy, efficiency, and computational complexity.

TABLE XII. COMPARISON BETWEEN THE PROPOSED SYSTEM AND THE LITERATURE REVIEW PAPERS

Author / Year	**Method**	**Accuracy**	**F1-score**	**Precision**	**Recall**
Tang et al. (2021) [8]	CNN(LeNet-5 based)	90%	–	–	–
Wakhare (2021) [9]	LSTM (Static + Dynamic Models)	94%	96%	98%	–
Aboshady et al. (2022) [10]	Ontological Semantic Environment	-	-	92%	91%
Atacak et al. (2022) [11]	CNN + ANFIS	94%	94%	94%	94%
Hung et al. (2022) [12]	TF-IDF + Deep Learning	91%	91%	–	91%
Jo et al. (2023) [13]	ViT Attention Mechanism	86%	87%	87%	86%
Prayoga et al. (2023) [14]	Bi-LSTM + Bi-GRU + MLP	83%	83%	83%	83%
Kiraz and Doğru (2024) [15]	CNN	91%	89%	90%	91%
Arrowsmith et al. (2025) [20]	CNN (DenseNet121) + GNN (GIN) with MLP	90%	90%	–	90%
Proposed system grey wolf +LSTM	grey wolf +LSTM	90%	90%	90%	90%
Proposed system grey wolf +CNN	grey wolf +CNN	93%	93%	92%	92%

TABLE XII presents an analysis and comparison of earlier research and studies on the topic of Detecting malware attacks using various methods and techniques. Findings unequivocally demonstrate that, when compared to alternative approaches and comparable research that employ DL techniques for categorical classification using Dataset CICMalDroid 2020, This table compares different deep learning and machine learning models based on their performance measures: Accuracy, F1-score, Precision, and Recall. The table shows methods from studies done between 2022 and 2024. It allows for an easy comparison with the new system which uses a CNN design along with GWO algorithm. In addition, Jo et al. [13] utilized the same dataset, their study employed an image-based representation, whereas the proposed model relies on encoded feature vectors. Such a difference in feature representation can have an effect on the end performance measures. That is why the comparison with their work should be considered carefully, considering the possible influence of such varying input forms. The experimental findings validate the evident superiority of CNN with GWO as compared to LSTM with GWO. The CNN model had a high accuracy of 93**%** and much lower training time (440.85 seconds) than LSTM which attained only 90 % and far longer training time (7654.66 seconds). The results emphasize that CNN with GWO did not only improve the precision but also demonstrated superior computational efficiency. It brings to light the fact that CNN with GWO is a more stable and efficient method of detecting malware.

4.5 Statistical Analysis Results

A statistical analysis was done to determine whether the performance gains achieved by GWO-enhanced models (CNN-GWO and LSTM-GWO) are statistically significant with reference to the performance of their counterparts who had no GWO enhancement. A total of five repeated runs per model were performed using the same training and testing data. Two statistics measures have been calculated:

1. Paired t-test: is employed to find out whether the performance variation between two models is statistically significant when the two models are tested on the same repeated runs. It makes the determination as to whether there is a greater difference in the means of paired observations than it would be with mere chance. The test generates a t-statistic, which is used to measure the strength of the difference, and p-value, which shows whether the difference is statistically significant or not. This parameter may be calculated in (9):

$$t = \frac{\acute{d}}{\sqrt{\frac{n}{SD}}} \tag{9}$$

Where:

$\acute{d}$: mean of the differences between the two models.
SD : standard deviation of these differences.
n : number of paired runs.

2. The confidence interval (CI) : gives the range within which the true population parameter should be within a given confidence level, which is usually 95%. It is applied to analyze the consistency of the model work with repeated executions and to measure the uncertainties of the estimated measures. The 95 percent confidence interval of the mean is calculated by the following in (10):

$$CI = \acute{x} \pm Z(\frac{\acute{\partial}}{\sqrt{n}}) \tag{10}$$

Where:

$\acute{x}$: the sample mean of the performance metric across repeated runs.
∂ : standard deviation of these differences.
z : standard deviation of these differences.
n : number of paired runs.

The statistical comparison of the base models (CNN and LSTM) and their GWO-optimized models is summarized in the following TABLE XIII:

TABLE XIII. SUMMARIZED STATISTICAL ANALYSIS RESULT

Metric	**Comparison**	**T-statistic**	**P-value**	**Confidence Interval**
Accuracy	CNN vs CNN-GWO	–7.52	0.0000	(87%, 89%) vs (92%, 93%)
	LSTM vs LSTM-GWO	–4.47	0.0000	(89%, 89%) vs (91%, 91%)
Precision	CNN vs CNN-GWO	–2.980	0.0000	(87%, 89%) vs (92%, 93%)

	LSTM vs LSTM-GWO	−4.000	0.0000	(89%, 89%) vs (91%, 91%)
recall	CNN vs CNN-GWO LSTM vs LSTM-GWO	−5.01 −1.000	0.0000 0.0000	(88%, 88%) vs (92%, 92%) (89%, 89%) vs (92%, 92%)
f1-score	CNN vs CNN-GWO LSTM vs LSTM-GWO	−4.743 −4.000	0.0000 0.0000	(88%, 88%) vs (92%, 92%) (89%, 89%) vs (92%, 92%)

All p- values less than 0.05, which means that GWO improves are statistically significant. The confidence intervals have made it quite clear that there is a steady increase in the performance following the application of GWO. In metrics where each run gave the identical best score (e.g., Precision, Recall, F1 of most models), the variance will be zero, meaning the t-test will give a t-test significance (p = 0.0000) and the confidence interval will reduce to a single value, this is normal. Statistical analysis proves that the application of GWO to CNN and LSTM models has led to a significant improvement in the accuracy, precision, recall, and F1-score. This shows how effective and reliable the proposed method of optimization can be.

5. CONCLUSION

This paper introduces an efficient and thorough malware classification strategy based on the methodology which involves deep learning and feature selection strategy. It was proposed that CNN and LSTM with grey wolf Optimization algorithm specifically would be optimized in terms of high accuracy and low computation load. The effectiveness of this model is proven by the results of the experiment, which produced high metrics in terms of accuracy, F1-score, precision and recall. Notably, the proposed CNN-GWO model achieved a 93% accuracy score because of its implementation with cross-validation techniques. This method of evaluation on different data subsets reduced overfitting and ensured that new data samples would yield results similar to the current data sample that would replicate its ability to reliably differentiate. In addition, the feature selection process done under the guidance of the GWO algorithm was critical in the dimensionality reduction without the performance of the model being spoilt as the model concentrated on the most relevant features. This is more than the accuracy; this methodology offers a faster processing time. Despite its promising results, the study acknowledges limitations related to dataset variability and generalizability. Resampling the data set with the SMOTE technique led to improved accuracy in the results and the performance of the model. The directions of the future work might lie in scaling the approach to larger and more diverse data and trying to trace the potential of detecting things in a dynamic environment in real time.

Conflicts of Interest

The authors declare no conflicts of interest.

Funding

No funding was received.

Acknowledgment

We would like to express our sincere to University of Information Technology and Communications – Institute of Informatics for Postgraduate Studies and Mustansiriya University the for their valuable support and contributions to this work.

References

[1] Y. Song, D. Zhang, J. Wang, Y. Wang, Y. Wang, and P. Ding, "Application of deep learning in malware detection: a review," *J. Big Data*, vol. 12, art. no. 99, Apr. 22, 2025, doi: 10.1186/s40537-025-01157-y.

[2] D. Kshirsagar and P. Agrawal, "A study of feature selection methods for Android malware detection," *J. Information and Optimization Sciences*, vol. 43, no. 8, pp. 2111-2120, Dec. 2022, doi:10.1080/02522667.2022.2133218.

[3] A. Redhu, P. Choudhary, K. Srinivasan, and T. K. Das, "Deep learning-powered malware detection in cyberspace: a contemporary review," *Front. Phys.*, vol. 12, 1349463, 2024, https://doi.org/10.3389/fphy.2024.1349463

[4] M. M. Issa, M. Aljanabi, and H. M. Muhialdeen, "Systematic literature review on intrusion detection systems: Research trends, algorithms, methods, datasets, and limitations," *J. Intell. Syst.*, vol. 33, no. 1, p. 20230248, 2024, https://doi.org/10.1515/jisys-2023-0248

[5] N. Waleed Abdulsattar and A. Abdulmajeed Abdulrahman, "Detecting Android attacks based on deep learning techniques: Status and future directions, " *AIP Conf.* Proc., vol. 3207, art. no. 050004, Sep. 19, 2024, doi: 10.1063/5.0234163.

[6] B M. M. Jawad, M. T. Younis, and A. T. Sadiq, "Solving flexible job-shop scheduling problem using harmony search-based meerkat clan algorithm," Int. *J. Artif. Intell. (IJ-AI),* vol. 11, no. 2, pp. 423–431, June 2022, doi:10.11591/ijai.v11.i2.pp423-431.

[7] M. T. Gaata, M. T. Younis, J. N. Hasoon, and S. A. Mostafa,"Hessenberg factorization and firework algorithms for optimized data hiding in digital images,"*J. Intell. Syst*., vol. 31, no. 1, pp. 440–453, Apr. 2022, doi:10.1515/jisys-2022-0029.

[8] B. H. Tang, Q. Kang, Z. X. Ni, H. Da, J. H. Xu, T. B. Liang, and Q. S. Bai, "Android malware detection based on deep learning techniques, " in Proc. *4th Int. Conf. Pattern Recognit. Artif. Intell. (PRAI),* 2021, pp. 20–21, doi: 10.1109/PRAI53619.2021.9551073.

[9] A. A. Wakhare, "Malware Detection in Android platform using DNN," M.S. thesis, National College of Ireland, Dublin, Ireland, 2021. Available: https://norma.ncirl.ie/5130/

[10] D. Aboshady, N. Ghannam, E. Elsayed, and L. Diab, "The malware detection approach in the design of mobile applications, " *Symmetry*, vol. 14, no. 5, art. no. 839, 2022, doi:10.3390/sym14050839.

[11] İ. Atacak, K. Kılıç, and İ. A. Doğru, "Android malware detection using hybrid ANFIS architecture with low computational cost convolutional layers, " *PeerJ Comput. Sci*., vol. 8, art. e1092, 2022, doi:10.7717/peerj-cs.1092.

[12] C.-H. Hung, Y.-m. Chen, and C.-C. Wu, "Detecting Android malware by combining system call sequence relationships with local feature calculation, " *in Proc. Int. Comput. Symp. (ICS), Commun. Comput. Inf. Sci.,* vol. 1723, S.-Y. Hsieh, L.-J. Hung, S.-L. Peng, R. Klasing, and C.-W. Lee, Eds. Singapore: Springer, 2022, pp. 362–373, doi:10.1007/978-981-19-9582-8_32.

[13] J. Jo, J. Cho, and J. Moon, "A malware detection and extraction method for the related information using the ViT attention mechanism on Android operating system, " *Appl. Sci*., vol. 13, no. 11, art. no. 6839, 2023, doi.org/10.3390/app13116839

[14] A. Prayoga, R. B. Hadiprakoso, R. N. Yasa, and G. Girinoto, " Deep learning for Android malware detection and classification using hybrid-based analysis: A comparative study, "*in Proc. IEEE Int. Conf. Cryptography, Informatics, and Cybersecurity (ICoCICs), 2023*, pp. 309–313, doi: 10.1109/ICoCICs58778.2023.10277613.

[15] Ö. Kiraz and İ. A. Doğru, "Visualising static features and classifying Android malware using a convolutional neural network approach, " *Appl. Sci*., vol. 14, no. 11, art. no. 4772, May 31, 2024, doi:10.3390/app14114772.

[16] J. Kim, Y. Ban, E.Ko, H. Cho &Jeong Hyun Yi, "MAPAS: a practical deep learning-based android malware detection system, *" Int. J. Inf. Secur*., vol. 21, pp. 725–738, Feb. 2022, doi:10.1007/s10207-022-00579-6.

[17] A. Almakayeel, "Deep learning-based improved transformer model on Android malware detection and classification in Internet of vehicles," *Sci. Rep*., vol. 14, no. 1, 2024, https://doi.org/10.1038/s41598-024-74017-z

[18] W. Guo, W. Du, X. Yang, J. Xue, Y. Wang, W. Han, and J. Hu, "MalHAPGNN: An enhanced call graph-based malware detection framework using hierarchical attention pooling graph neural network," *Sensors*, vol. 25, no. 374, 2025, https://doi.org/10.3390/s25020374

[19] A. Kulkarni and S. O'Shaughnessy, "Malware detection using dynamic graph neural networks," *Eur. Conf. Cyber Warf. Secur.*, vol. 24, no. 1, pp. 830–837, 2025, https://doi.org/10.34190/eccws.24.1.3459

[20] J. Arrowsmith, T. Susnjak, and J. Jang-Jaccard, "Multimodal deep learning for Android malware classification," *Mach. Learn. Knowl. Extr.,* vol. 7, no. 1, p. 23, 2025. , https://doi.org/10.3390/make7010023.

[21] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, "Effective and efficient hybrid Android malware classification using pseudo-label stacked auto-encoder," *J. Netw. Syst. Manag*, vol. 30, no. 1, p. 22, 2022,https://doi.org/10.1007/s10922-021-09634-4

[22] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android malware category classification using semi-supervised deep learning," *in Proc. IEEE Int. Conf. Dependable, Autonomic and Secure Computing,* Int. Conf. Pervasive Intelligence and Computing, 2020, pp. 1–8, doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.