









## Research Article

# AntDroidNet Cybersecurity Model: A Hybrid Integration of Ant Colony Optimization and Deep Neural Networks for Android Malware Detection

Riyadh Rahef Nuiiaa AlOgaili<sup>1,\*</sup>,, Osamah Adil Raheem<sup>1</sup>,, Mohamed H Ghaleb Abdkhaleq<sup>1</sup>,, Zaid Abdi Alkareem Alyasserri<sup>2,3</sup>,, Saif Ali Abd Alradha Alsaiedi<sup>1</sup>,, Ali Hakem Alsaeedi<sup>4,5</sup>,, Yousif Raad Muhsen<sup>6,7</sup>,, Selvakumar Manickam<sup>8</sup>,

1 College of Computer Science and Information Technology, Wasit University, Wasit, Iraq

2 Information Technology Research and Development Center (ITRDC), University of Kufa, Najaf, Iraq

3 College of Engineering, University of Warith Al-Anbiyaa, Karbala, Iraq

4 College of Computer Science and Information Technology, University of Al-Qadisiyah, Al-Qadisiyah, Iraq

5 Department of Computer Techniques, Imam Kadhum College, Diwaniyah, Al-Qadisiyah, Iraq

6 Department of Civil, College of Engineering, Wasit University, Wasit, Iraq

7 Technical Engineering College, Al-Ayen University, Thi-Qar, Iraq

8 Cybersecurity Research Centre, Universiti Sains Malaysia, Penang, Malaysia

## ARTICLE INFO

### Article history

Received 25 Oct 2024

Accepted 16 Jan 2025

Published 07 Feb 2025

### Keywords

Android Malware

Malware detection

AntDroidNet

Cybersecurity

ACO & DNN



## ABSTRACT

Malware detection is a vital problem, and efficient methods that can efficiently detect malware are needed. The increasing use of mobile computers makes malware detection a vital part of security in an era where smartphones have come to play a key role in many of our daily lives. Earlier approaches, however, suffer from high false positive rates; they are not scalable for larger databases, or they are not amenable to adapt well to novel zero-day malware. For these reasons, the demand for more sensitive and flexible detection models is high. In this study, we develop a hybrid mobile malware detection framework that leverages ant colony optimization (ACO) and deep neural networks (DNNs) to improve detection accuracy, reduce the rate of false positives, and make the model resilient to new malware. AntDroidNet is a novel ACO-enabled feature selection model that dynamically reduces the feature dimensionality by selecting single instances to include the most informative properties and avoid dimensionality. A DNN is consequently constructed to train the determined set of features, improving the identified classification performance and decreasing the number of instances with false discoveries. In this way, a self-optimizing feedback loop can iteratively improve the feature selection process given the performance of the DNN, leading to a dynamic and efficient detection model. Using the CICMalDroid2020 dataset, the proposed AntDroidNet model achieves a remarkable accuracy of 99.89% and an excellent false positive rate of only 0.13% and outperforms the classical machine learning algorithms in terms of accuracy and efficiency. AntDroidNet is a scalable and powerful mobile malware detection model that eclipses all state-of-the-art methods and shows important enhancements in efficiency and reliability. By prototyping whitelisting systems, this work opens new avenues in mobile security and lays the groundwork for future work on building real-time detection components and system components able to scale to the fast pace of evolution of mobile malware in new connected ecosystems.

## 1. INTRODUCTION

Mobile devices have become ubiquitous since their mass adoption; however, mobile malware has also begun to infiltrate the mobile cyber threat landscape. This malware is designed to interrupt or take over a mobile device that just goes through malicious app stores, middleware, or counterfeit apps. Once installed, mobile malware can be a mess, logging in credentials, hitting up on unauthorized messages, changing device settings or cutting performance [1][2]. Such advanced and large classes of mobile malware are a threat to the current systems of mobile malware detection, especially with the evolution of polymorphic malware, which entirely adopts a case to avoid detection applied by most signature-based approaches [3][4][5].

\*Corresponding author. Email: riyadh@uowasit.edu.iq

In light of the continually evolving online environment, machine learning (ML) represents a novel method to address these issues, providing more flexible and scalable solutions than traditional approaches do. Malware writers exploit machine learning algorithms to avoid detection; thus, we need better techniques to fight these evolving threats[6][7]. Hackers are becoming well aware of AI and advanced analytics, which are paramount in cybersecurity, a booming field that has already scored many points with its use of adversarial learning and deep learning models to bridge security challenges [8][9]. Although there has been great progress in leveraging machine learning techniques for mobile malware detection, there are still numerous challenges to overcome[21]. Owing to the rapid growth and diversity of mobile malware, traditional malware detection methods, such as signature-based methods, have fallen short. Moreover, with the incorporation of IoT devices further increasing the attack surface, issues surrounding privacy breaches and data compromises remain at the top of mind [10][11]. As malware becomes more complex over time, detection becomes even more difficult, and new sophisticated techniques to discover obfuscated and mutated threats are needed [12][13]. Given the extensive use of mobile devices, particularly those of the Android platform, they have become prime targets for malware that targets mobile devices, creating an expectation for an efficient detection method[27]. One possible answer lies in combining machine learning and deep learning models; however, few studies have addressed the implementation of adaptive, efficient, and scalable detection systems [14][15]. Mobile devices, especially Android platforms, are prime targets for malware due to their widespread use, highlighting the urgent need for robust detection methods[31]. The integration of machine learning and deep learning models offers a potential solution, but this area remains underexplored, particularly for developing adaptive, efficient, and scalable detection systems [16][17].

In this paper, we propose a new hybrid integration model called AntDroidNet Cybersecurity, which incorporates ACO and a DNN, for the mobile malware detection system. Dynamically selecting appropriate features not only ensures dimensionality reduction but also optimizes the feature set for improved detection speed. The proposed AntDroidNet cybersecurity model employs ACO to dynamically select only those features that contribute most to effective detection. Using ACO-optimized characteristics integrated with DNNs contributes to enhanced classification accuracy along with reduced false positives. Moreover, the proposed AntDroidNet cybersecurity model includes a novel self-learning feedback technique in which the performance metrics of the DNN are utilized for this purpose to sequentially adjust the features accordingly to have an adaptable system over a time period.

This study aims to achieve the following objectives:

- To design a dynamic model that detects mobile malware by combining ACO and DNN.
- Improving feature selection can improve detection accuracy and reduce false positives.
- To create a self-improving feedback mechanism system that is able to adapt to the dynamicity of mobile malware.

Compared with traditional techniques, the proposed AntDroidNet cybersecurity model is a promising approach for mobile malware detection, with various benefits. The integration of ACO and DNN ensures reduced training/inference time, resulting in speed and potential applicability in real-time scenarios in mobile environments. Moreover, owing to dynamic feature selection, the framework can be adapted to new types of malware, allowing an effective response to future cybersecurity issues. In addition to contributing to the active research of mobile security solutions, this research will lay the groundwork for further investigation into marrying advanced optimization techniques with deep learning in the field of real-time malware detection for mobile and IoT environments.

## 2. RELATED WORKS

Android malware detection has become a crucial concern in cybersecurity. Detection methods are primarily categorized into static and dynamic analyses. Static analysis examines code without execution, whereas dynamic analysis observes malware behavior in a controlled environment. Researchers have employed various machine learning algorithms for malware detection, including naive Bayes, support vector machine (SVM), K-nearest neighbors (KNN), and decision trees. More recently, deep learning techniques have gained attention for their potential to extract high-level features directly from raw data. The field continues to evolve, with ongoing research exploring both traditional and advanced machine learning approaches to increase the accuracy and efficiency of Android malware detection. This highlights the complex and dynamic nature of the challenge, necessitating continued innovation in cybersecurity strategies.

The authors of [18] proposed a novel approach for Android malware that encompasses the use of machine learning. Dynamic analysis is performed on the dataset comprising benign and malicious CICMalDroid 2020 applications with the aim of increasing the detection accuracy. They focus on feature extraction and outlier management, achieving 95% accuracy with a 60.2% reduction in the feature subset. The centerpiece of their approach is hierarchical classification based on ensembles of classification algorithms, including random forest, k-nearest neighbors, and support vector machines, which are fused at the final stage via a weighted voting mechanism. A comparison with other techniques demonstrates the superiority of this approach, especially in multiclass problems of classifying the extensions of multithreat malware, which is a great improvement in the protection of Android devices from accessing new cybercriminal functionalities in the future. In [19], a

novel method for autonomously detecting Android malware was presented by MDADroid, and it is based on having a Functionality-API mapping, which enhances its resource management and environmental adaptability. This technique uses permission APIs to draw a graph on devices about what APIs correspond to what functionalities, hence boosting feature representation from the API level up to an abstraction level. The calculation of how two APIs are similar constitutes an API mapping and is a substantial component since, during the introduction of newer versions of android, there are changes made to the API; hence, the components have minimal adaptation costs. Furthermore, these greedy techniques reduce the time needed for adaptation, allowing models to be more easily retrained and thus improving performance. Extensive experiments on many datasets, such as AndroZoo, CICAndMal 2017, CICMalDroid 2020 and Drebin, have confirmed that MDADroid achieves high performance and is invariant to API changes. Furthermore, this method enables greater detection capabilities, reduces the total model training and evaluation costs, and holds great potential for mobile security since it offers better performance, lower costs and greater robustness. In [20], the authors presented MTDroid, which is an improvement on the methods of Android malware detection through the use of moving target defense technology. This novel approach implements a large, complex and persistent changing set of classifiers, making identical system exploitation targets quite difficult for attackers to predict. With the use of diversifying the adversarial training approach and optimized ensemble learning, MTDroid is able to greatly enhance evasion attack resistance. Additionally, the ability to update the stock of models in the framework in real time according to the performance metrics makes the framework more versatile and robust. Our extensive testing across various datasets shows that MTDroid outperforms current methods in a wide range of malware detection tasks for Android operating systems, essentially providing all necessary functionalities for Android malware detection while sustaining strong effectiveness in the presence of manipulative disturbances. [22] proposed a new scheme for classifying Android malware by using audio features obtained from sonified APKs. By sonifying the application data, the authors capture the proprietary features in the audio for investigation. The work assesses different nature-inspired algorithms for feature selection, and the genetic algorithm performed the best, yielding a 99.72% accuracy level. This strategy appears to be effective in solving known problems in malware detection, such as resource consumption, scaling, and difficulty in circumventing. This study promotes the idea that the use of optimized feature selection techniques through metaheuristics can improve the classification performance while minimizing the computational load. Such an approach also provides a different angle for addressing mobile security, which can be used in addition to, or even instead of, the usual detection methods. One of the recent developments I have developed is a way to detect malware on android devices via machine learning and audio features [23]. Starting by transforming the apk files into waveforms and extracting audio features, researchers have applied metrics such as mel-frequency cepstral coefficients (MFCCs) and gamma-frequency cepstral coefficients (GFCCs). They combined these metrics to account for the narrow downside of concentrating on few features. The researchers carried out experiments using the CICMalDroid 2020 dataset and achieved an accuracy of 98.96%, recall of 99.65%, and F1 score of 99.33%. The results suggest that using multiaudio feature fusion enables a deeper and fuller understanding of android applications and enhances the strength of crystalline systems. [24] described the development of GA-StackingMD, which is a new approach to Android malware detection that leverages several classifiers. The system, in its basic form, uses a stacking technique that combines five different classifiers, where genetic algorithms (GAs) are used for optimization. This concept design helps address two longstanding issues in the field: the curse of feature space and low accuracy rates of detection. The framework consists of two sequential feature selection stages to reduce the aforementioned limitations. First, informed gain analysis is used to determine the most relevant features. A chi-square test follows and removes duplicate feature subsets, thereby identifying the most significant features. This epidemiologic study of feature selection establishes the basis for an improved detection model owing to precise feature selection. According to experimental trials, the GA-StackingMD model outperforms the individual classifiers in terms of performance. For example, the model was tested on the CIC-AndMal2017 dataset and obtained an accurate detection rate of 98.43%. Similarly, the same evaluations on the CICMalDroid2020 database yielded a model accuracy of 98.66%. These findings highlight the suitability and effectiveness of the framework within various datasets.

### 3. PROPOSED ANTROIDNET CYBERSECURITY MODEL

The proposed AntDroidNet cybersecurity model is systematically executed by the following constituent components. First, the raw data are preprocessed via normalization and feature scaling processes. Second, ant colony optimization is implemented to select a set of relevant features. Here, the principles of global exploration and local search are utilized in the proposed model. Ant colony optimization considers both heuristic and pheromone information to choose a collection of influential features. For this purpose, a longitudinal dataset is decomposed into K-hop neighborhood communities. Upon identifying a collection of influential features, the reduced dataset is reconstructed through the extracted features. Finally, the constructed hybrid deep neural network comprises input layers, three hidden layers, and one output layer. A rectified linear unit function is employed as an activation function. The backpropagation algorithm is adopted in training the developed hybrid deep neural network. The significance of the proposed model is tested via a diverse set of performance metrics. The model focuses on detecting mobile malware with high accuracy while minimizing false positives and false negatives. Figure 1 shows the steps of the proposed model.

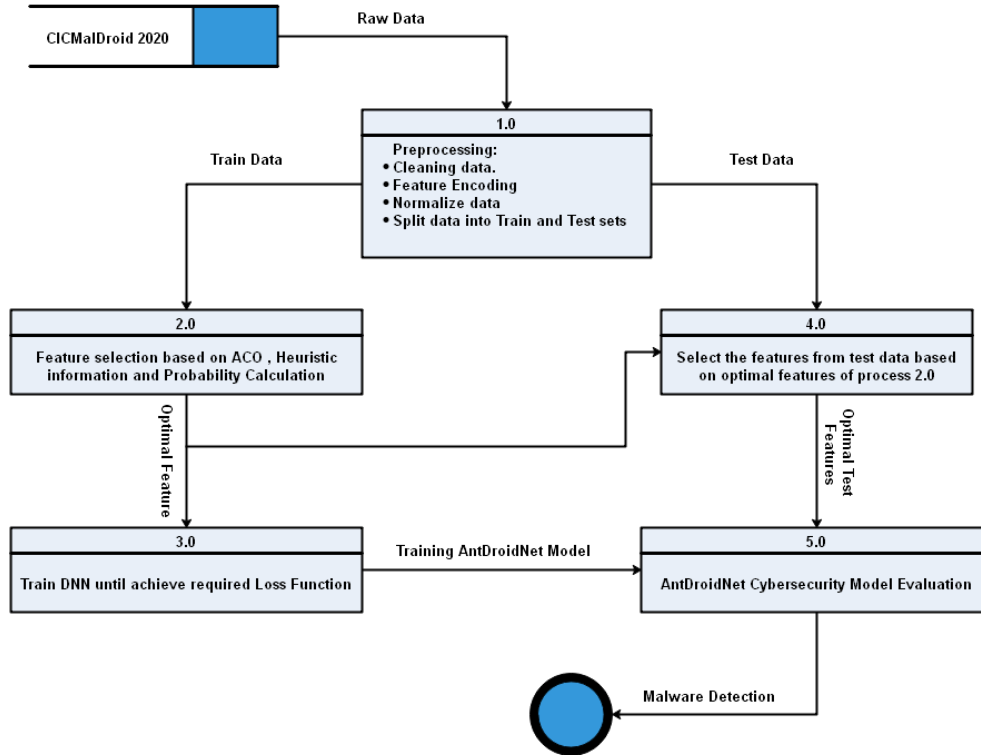


Fig. 1. A block diagram of the proposed AntDroidNet cybersecurity model

### 3.1 Data Preprocessing

For successful mobile malware detection, mobile malware data must be properly transformed into an appropriate representation. This fundamental process is called data preprocessing. First, it is very important for the data to be of high quality for effective training of the ML model. Data that have missing values or imbalances in label values are not suitable for direct input to DNNs. To address these issues, several data preprocessing steps were conducted. Data Preprocessing for Data Inputs Solutions for Missing Data: Delete NA Data or Imputation.

Data Preprocessing for the Feature Selection Feature Extractor: The most significant values for target labels can be used when the extracted values are significantly different from those for benign samples. To select the optimal feature values, the t test method was used.

Data Preprocessing for Data Pretraining Min–Max normalization: The purpose of data normalization is to convert the dataset into a particular distribution or a normal form as input for a deep neural network. By doing so, the performance of the optimization in the next stage is significantly improved. Data transformation as augmentation: Another preprocessing step is to use the augmentation function to increase the size of the dataset. The dataset could be augmented with noise, brightness, contrast, and saturation gain.

By proper data preprocessing, the model components receive credible input values, resulting in decreased bias, lack of significance, or false interpretation of the learning model. Consequently, achieving tremendous enhancements in the training quality of the CNN model because of these preprocessing stages is highly likely. The output of this stage is the separated data in accordance with the proposed approach: one for training, validation, and testing with proper data, one for the pretraining of the RBMs, and the other for data in the fine-tuning stage.

- ❖ Min–max normalization: Mobile malware datasets typically contain raw features such as permissions, API calls, and network traffic. These features can have different ranges of values, so normalization is critical.

Let:  $X = [x_1, x_2, \dots, x_n]$

represents the feature set for each sample, where n is the total number of features. To normalize the dataset, Min–Max normalization is applied to ensure that all features fall within the range [0, 1]:

$$x'_i = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)} \quad (1)$$

where  $x_i$  is the original value of feature  $i$ ,  $x'_i$  is the normalized value, and  $\min(x_i)$  and  $\max(x_i)$  are the minimum and maximum values of feature  $i$ , respectively.

- ❖ Feature Encoding (One-Hot Encoding): If the dataset contains categorical features (e.g., permissions or API calls), they need to be encoded. One-hot encoding is often used:

$$f_i^{(k)} = \begin{cases} 1 & \text{if sample } i \text{ has the } k^{\text{th}} \text{ permission / API call,} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Algorithm 1 presents the steps of preprocessing for the CICMalDroid 2020 dataset.

*Algorithm 1 Data Preprocessing Algorithm for CICMalDroid2020 Dataset*

*Require: Raw CICMalDroid2020 dataset  $X$  with  $m$  samples and  $n$  features, where  $X = \{x_1, x_2, \dots, x_n\}$*

*Ensure: Preprocessed dataset  $X_{\text{encoded}}$  with normalized and one – hot encoded features*

*1: Initialization: Initialize the preprocessed dataset  $X_{\text{encoded}} = \emptyset$*

*2: Step 1: Min-Max Normalization*

*3: for each feature  $x_j$  in  $X$  do:*

*4:   Compute the minimum value of feature  $x_j$ :  $x_{j,\min} = \min(x_j)$*

*5:   Compute the maximum value of feature  $x_j$ :  $x_{j,\max} = \max(x_j)$*

*6:   Normalize feature  $x_j$  using equation number 1*

*7:   Append  $x_j^{\text{norm}}$  to  $X_{\text{encoded}}$*

*8: end for {All features are now normalized between 0 and 1}*

*9: Step 2: One-Hot Encoding of Categorical Features*

*10: for each feature  $x_k^{\text{norm}}$  in  $X_{\text{encoded}}$  do*

*11:   if  $x_k^{\text{norm}}$  is a categorical feature then*

*12:     Perform one – hot encoding on  $x_k^{\text{norm}}$*

*13:     Append the resulting encoded features to  $X_{\text{encoded}}$*

*14:   endif*

*15: end for {Dummy variables are created for each categorical feature}*

*16: Step 3: Return the Preprocessed CICMalDroid2020 Dataset*

*17: return  $X_{\text{encoded}}$*

### 3.2 Feature Selection Using Ant Colony Optimization (ACO)

After the dataset containing many features from hardware and API calls is obtained, the most important step is to determine the feature selection for optimizing the classification process. The proposed method extracts the top-k relevant features from a dataset to facilitate the classification process in the experiments. The definition of the top-k relevant features is determined via a modified ant colony optimization algorithm. The concept of ACO reflects the foraging behavior of ants. From the strategy used in ACO, where each strategy is equipped with its own description, the feature selection method using ACO supports handling a large feature space, minimizing the phenomenon of irrelevant and redundant features. The process of this method will be discussed via the following figures, which include the following topics: phases in the FSD-ACO project, the ant colony system, feature subset evaluation, and modified feature subset evaluation. However, the workload required will not be discussed in depth. This subsection proposes the feature selection framework using modified ACO to support the selection of the most relevant attributes that will be used in the classification process from the results using state-of-the-art algorithms. The main purpose of using the feature selection framework is to determine more efficient ways of identifying a small number of top-k relevant features from a dataset that can be used in the subsequent classification stages. These top-k relevant features are fed into the classifier to verify the performance produced.

To dynamically select the most relevant subset of features via ACO, the dimensionality is reduced, and the efficiency of the DNN is improved. Therefore, the main task in this stage is to select the most important features via ACO. Let:

- $F = \{f_1, f_2, \dots, f_n\}$  is the feature set, where  $n$  is the number of features.
- $S \subseteq F$  is a subset of features selected by an ant.
- $A$  is the number of ants in the population.
- $\tau_{ij}$  is the pheromone level associated with feature  $f_j$  after iteration  $i$ .
- $\eta_j$  is the heuristic desirability of selecting feature  $f_j$ , where  $y$  is the target class.
- where  $\alpha$  and  $\beta$  are constants that control the influence of pheromone and heuristic information, respectively.

### 3.2.1 Probability of Feature Selection

The use of probabilities for selecting certain items, which in our case is the selection of features, is a well-known aspect of the ACO algorithm. Here, we can describe each feature as a solution candidate; thus, for each feature, ant agents must choose the value "yes" or "no" to select this feature for detection or not. The probability of choosing each feature (which is related to the value of "yes" for each feature) is then determined according to a combination of the pheromone trails and the heuristic information describing the relevance of the feature for detection. The feature that has higher pheromone trail values is considered more appropriate for detection, whereas the heuristic parameter guides the detection in a direct way to the most likely selected features. This probability-based mechanism is robust enough to adapt the feature subsets according to the detection scenario, so it can automatically defend against newly emerging malware.

Notably, the calculation of the initial value of the probabilities could be profoundly impacted by the parameter settings. Each parameter can assume different value ranges, which may result in different feature probabilities and consequently different detection efficacies. Therefore, proper parameter setting is crucial for establishing the selection probability of each feature. Both the heuristic and the pheromone parameters are fundamental. We cannot simply set a specific value. It needs to be carefully determined and configured on the basis of the value range of the input features. Setting a proper value for heuristic information and pheromone trails can enhance the robustness and effectiveness of the feature selection process. Usually, we balance this by preconfiguring these parameters on the basis of the range of the input features. Another highlight of this approach is that the balance between pheromone trails and heuristic information can be quickly reached, thus leading to fast progress in feature selection.

$$P_{ij}(S) = \frac{\tau_{ij}^\alpha \eta_j^\beta}{\sum_{k \in F} \tau_{ik}^\alpha \eta_k^\beta} \quad (3)$$

where:

- where  $P_{ij}(S)$  is the probability that feature  $f_j$  is selected by ant  $i$  at step  $S$ .
- $\tau_{ij}$  is the pheromone level associated with feature  $f_j$  after iteration  $i$ .
- where  $\eta_j$  is the heuristic desirability of selecting feature  $f_j$  (e.g., correlation of feature  $f_j$  with the target  $y$ ),
- where  $\alpha$  and  $\beta$  are constants that control the influence of pheromone and heuristic information, respectively.
- $F$  is the set of all features.
- $\eta_j$  can be calculated as  $\frac{1}{\text{correlation}(f_j, y)}$ .

### 3.2.2 Heuristic Information

Heuristic information is important for feature selection via ACO. It is generated on the basis of the probability of selecting a feature that reduces the cost function. The heuristic information is between zero and one and influences the cost function value. Higher heuristic information leads to lower cost function values, making it useful for selecting features. The influence of the pheromone value and heuristic information aims to improve search performance in the DNN architecture. The decision for feature selection is dynamic and flexible. Heuristic information plays a role in selecting optimal features but can also increase the ability of mobile malware detection. Balancing heuristic information's increase in ability with feature coverage is necessary. Higher heuristic values in ACO lead to faster exploration.

$$\eta_j = \frac{1}{\text{correlation}(f_j, y)} \quad (4)$$

where  $y$  is the target (malignant or benign) and where  $\eta_j$  represents the correlation between feature  $f_j$  and the target.

### 3.2.3 Pheromone Update

Ant colony optimization (ACO) is an evolutionary computation method for solving optimization problems. The key to enhanced ACO is the pheromone update rule, which balances accumulation and evaporation. ACO techniques automatically accumulate and evaporate pheromones on the basis of selected features. Efficient automatic design is necessary for dynamic adaptation. ACO evolution involves increasing and decreasing pheromone levels on the basis of feature quality. Higher pheromone levels reinforce good selection, whereas lower levels allow the removal of low-quality features. This feature-specific approach enables ACO to select and evolve good features while adaptively removing the worst features on the basis of pheromone effects.

Once ants construct their solutions (subsets of features), the pheromone is updated to reflect the quality of the solution.

$$\tau_{ij}(t+1) = (1 - P) \cdot \tau_{ij}(t) + \sum_{k=1}^A \Delta \tau_{ij}^k \quad (5)$$

where:

- where  $P$  is the pheromone evaporation rate.
- where  $\tau_{ij}(t)$  is the pheromone level associated with feature  $f_j$  at iteration  $t$ .
- where  $\Delta\tau_{ij}^k$  represents the change in pheromones contributed by ant  $k$ .
- where  $A$  is the total number of ants in the population.
- $\Delta\tau_{ij}^k$  is the pheromone deposited by ant  $k$  on feature  $f_j$ :  $\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{\text{cost}(S_k)} & \text{if feature } f_j \in S_k, \\ 0 & \text{otherwise.} \end{cases}$  (6)

### 3.2.4 Aco Cost Function

Feature selection is dependent on the ACO cost function. It assesses an optimal subset of features and relies on the optimization procedure. The best feature subset can be obtained via the ACO-based strategy. Note that the cost function is also optimized to balance several objectives and evaluate distinct choices of feature-set subsets. It seeks to maximize the accuracy of the detection output relative to the accuracy of the maximal classifier, in turn yielding a more compact, yet equally effective, detection model. The fast calculation of computational complexity values is inferred from complex decision models. To follow the structure of the system and build a function on the basis of the results, we need to understand the cost function. Therefore, the cost function in ACO is typically based on the classification performance of the DNN. The classification error or accuracy of the DNN when the selected feature subset is used is as follows:

$$\text{cost}(S_k) = 1 - \text{accuracy}(S_k) \quad (7)$$

where  $\text{accuracy}(S_k)$  is the classification accuracy of the DNN trained on the feature subset ( $S_k$ ).

---

Algorithm 2 presents feature selection via ant colony optimization (ACO).

---

Algorithm 2 *Ant Colony Optimization for Feature Selection*

Require: *Set of features F*

$= \{f_1, f_2, \dots, f_n\}$ , *number of ants num\_ants, number of iterations num\_iterations,*

*parameters  $\alpha$  and  $\beta$  for pheromone and heuristic influence, evaporation rate P*

Ensure: *Best subset of selected features  $S_{best}$*

1: Step 1: Initialize the Pheromone

2:  $\text{pheromone} \leftarrow \{\tau_1 = 1, \tau_2 = 1, \dots, \tau_n = 1\}$  {Equal pheromone values for all features}

3:  $S_{best} \leftarrow \emptyset$  {Initialize the best accuracy}

4:  $\text{best\_accuracy} \leftarrow 0$  {Initialize the best accuracy}

5: for each iteration  $k = 1$  to  $\text{num\_iterations}$  do

6:   Step 2: Construct Solutions for Each Ant

7:    $\text{Ants} \leftarrow \emptyset$

8:   for each ant  $i = 1$  to  $\text{num\_ants}$  do

9:      $S_i \leftarrow \emptyset$  {Initialize feature subset for ant  $i$ }

10:     for each feature  $f \in F$  do

11:       Calculate the probability of selecting feature  $f$  using eq (3)

12:       if randomly generated number  $\leq P(f)$  then

13:          $S_i \leftarrow S_i \cup \{f\}$  {Add feature  $f$  to the subset}

14:       end if

15:     end for

16:      $\text{Ants} \leftarrow \text{Ants} \cup \{S_i\}$

17:   end for

18:   Step 3: Evaluate Solutions Using DNN

19:   for each ant  $i = 1$  to  $\text{num\_ants}$  do

20:      $\text{accuracy}(S_i) \leftarrow$  Evaluate the performance of DNN on  $S_i$

21:   end for

22:   Step 4: Update the Best Solution

23:    $\text{best\_ant} \leftarrow \text{argmax}_i \text{accuracy}(S_i)$

24:   if  $\text{accuracy}(S_{\text{best\_ant}}) > \text{best\_accuracy}$  then

25:      $S_{best} \leftarrow S_{\text{best\_ant}}$

26:      $\text{best\_accuracy} \leftarrow \text{accuracy}(S_{\text{best\_ant}})$

27:   end if

28:   Step 5: Pheromone Update

29:   for each feature  $f \in F$  do

30:     Update pheromone value using eq(5)

31:   end for

32: end for

33: Step 6: Return the Best Feature Subset

34: return  $S_{best}$

---

## 4. DEEP NEURAL NETWORK (DNN)

DNN training is the process of constructing a robust mathematical model through learning. The network architecture defines the structure. Training estimates functional mappings by adjusting connection weights through iterative alternation. The initial parameters are randomly sampled, and the learning process adjusts the weights iteratively. DNNs have deep layer-by-layer structures for extracting intricate feature transformations. Training aims to minimize model error via forward propagation and backpropagation. It competes between minimizing training error and explaining test error. The model is capable and discriminative because of its significant improvement in predictive ability.

Once the optimal feature subset  $S$  is selected, the DNN is constructed and trained for classification. After ACO selects the optimal subset of features, the DNN is trained to classify whether a sample is malware or benign. The DNN architectures are as follows:

- **Input Layer:** The input layer corresponds to the selected feature subset  $S$  from ACO, which has  $d$  features.
- **Hidden Layers:** A series of hidden layers are used for feature transformation. Each hidden layer computes a linear transformation followed by a nonlinear activation function (ReLU).
- **Output Layer:** The output layer computes a binary classification via the sigmoid activation function.

Once ACO selects the optimal feature subset  $S$ , the DNN uses these features for classification. The DNN consists of multiple layers where each layer performs linear transformations followed by activation functions.

### 4.1 Input Layer

Let  $X = [x_1, x_2, \dots, x_d]$  be the input vector, where  $d$  is the number of selected features (from ACO). The input to the DNN is the selected feature vector, which is passed through the first hidden layer.

### 4.2 Hidden Layers

#### 4.2.1 Forward ARD Propagation (DNN)

Forward propagation in DNNs is crucial for predicting output scores or labels. The data are processed layer by layer, adjusting the weights and features. The activation functions add nonlinearity and facilitate weight adjustment. Without activation functions, it is similar to one deep layer. Proper activation function selection reduces negative functions and improves model training. For each hidden layer  $l$ :

$$Z^l = W^l \cdot A^{l-1} + b^l \quad (8)$$

where  $Z^l$  is the linear combination of inputs at layer  $l$ ;  $W^l$  is the weight matrix for layer  $l$ ;  $A^{l-1}$  is the activation from the previous layer (or input features for the first hidden layer); and  $b^l$  is the bias for layer  $l$ .

#### 4.2.2 Activation Function (ReLU for Hidden Layer)

The ReLU is a commonly used activation function in hidden layers for several reasons: it is computationally efficient and speeds up training progress. It overcomes the vanishing gradient problem of the sigmoid function. It can learn complex representations. However, other activation functions, such as sigmoid, Softmax, and tanh, have advantages. While ReLU performs well in training progress and when equipped with increasing layers, it is best to use Sigmoid and Softmax for classification tasks. The ReLU is suitable for providing positive values close to real-world features and can also incorporate sparsity in hidden layers.

$$A^l = \max(0, Z^l) \text{ (ReLU Activation)} \quad (9)$$

### 4.3 Output Layer

The sigmoid function is used in the output layer for binary classification in DNN frameworks. Sigmoid maps previous layer outputs to binary classifications, resulting in a probability value between 0 and 1. A value close to 1 or 0 indicates confident predictions. The properties and interpretability of the sigmoid function make it popular for binary decision-making. Its gradients aid in gradient-based optimization. While there are trade-offs, using the sigmoid function improves detection accuracy in mobile malware detection.

$$\hat{y} = \sigma(W^o \cdot A^l + b^o) = \frac{1}{1+e^{-Z^l}} \quad (10)$$

where  $W^o$  is the weight matrix of the output layer;  $A^l$  is the activation from the last hidden layer;  $b^o$  is the bias vector of the output layer;  $\sigma(z) = \frac{1}{1+e^z}$  is the sigmoid function; and  $Z^l$  is the preactivation output of layer  $l$ .



## 4.4 Loss Function

### 4.4.1 Binary Cross-Entropy Loss Function

There are many metrics used for measuring the prediction power of a DNN on a binary classification task, but one very important loss function that we can compute for measuring the error is the binary cross-entropy loss function. It calculates the gap between the true labels (0 or 1) and the predicted likelihoods (i.e., sigmoid output). This is calculated as the log loss for each instance; the greater the difference between the ground truth and predicted probabilities is, the higher the penalty. To ensure that the model correctly predicts, the ideal case is the predicted probability, which resembles the actual probability. In other words, the smaller the binary cross-entropy value is, the better the model fits the data, indicating less discrepancy between the predicted and actual outcomes and better prediction accuracy. Going slightly deeper, the binary cross-entropy may actually converge slower if one has imbalanced classes, so choosing the proper loss function is extremely important in both accuracy and convergence. Therefore, we want to know how to update our model's weights in backpropagating with the right information such that the model is guided toward performing better binary classification tasks; now, we add binary cross-entropy to guide us through.

$$L(\hat{y}, y) = 1 = -\frac{1}{N} \sum_{i=1}^N (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (11)$$

where  $y_i$  is the true label (malware/benign) (0,1) for the  $i$  –  $th$  sample;  $\hat{y}_i$  is the predicted probability for the  $i$  –  $th$  sample; and  $N$  is the number of samples.

## 4.5 Optimization and Backpropagation

Backpropagation is a mechanism that allows iterative updating of the weights and biases of a DNN, pushing the model closer to minimizing its loss function. Through a process called backpropagation, the model uses the chain rule to propagate the gradient backwards and update its parameters on the basis of any error it has made. The computation of the gradient is very important for optimizing the objective function and making precise predictions and incorrect calculations of the gradient can produce all erroneous results. Learning factors such as the 'learning rate' have a very important effect on the convergence rate and training time. A learning rate of 0.001 is used in this study, as it provides a good compromise between the speed of convergence and stability. When combined with the Adam optimizer, this learning rate facilitates effective weight adjustments while inhibiting problems such as divergence or slow convergence. The empirical results confirm the efficiency of the details of the model, especially when it is applied to the optimized feature subset generated from ACO, where it enhances the model performance overall.

### 4.5.1 Gradient Computation

For a given layer  $l$ , the gradients of the loss function with respect to the weights and biases are calculated as follows:

- A- Error at the output layer:** This error is related to the backpropagation learning algorithm. It computes the difference between the predicted output  $\hat{y}$  and the true label  $y$ , which is fed as an input to the backpropagation algorithm. This error provides guidance for the whole learning process by telling the network what to change to update its weight. Therefore, the model can obtain a smaller overall loss function by minimizing  $\delta^L$ , which can enhance the prediction ability of the model. This calculated error is then propagated back through the layers of the network, enabling each layer to adjust its weights according to its contribution to the ultimate error in the output, thus optimizing the network's performance and ensuring convergence during the training process.

$$\delta^L = \hat{y} - y \quad (12)$$

- B- Backpropagation:** This equation is essential for backpropagation, as it determines the error at each layer  $l$  on the basis of the error of the next layer  $\delta^{l+1}$ , its weights  $W^{l+1}$ , and the derivative associated with the activation  $\frac{\partial A^l}{\partial z^l}$ . It helps propagate the output error back through the neural network such that each hidden layer can relate its contribution to the final error. By enabling the network to adjust weights at every layer according to the degree to which each neuron contributes to the overall output, this results in effective learning and speedier convergence. The network can thus achieve a better fit in the sense of capturing complex patterns in the data, as the loss function is reduced by adjusting the weights iteratively in this manner.

$$\delta^l = (\delta^{l+1} \cdot W^{l+1}) \odot \frac{\partial A^l}{\partial z^l} \quad (13)$$

where  $\odot$  denotes elementwise multiplication and where  $\frac{\partial A^l}{\partial z^l}$  is the derivative of the activation function.

## 4.5.2 Gradients of Weights and Biases

**A- Gradient of weight:** Optimizing DNN performance requires adjusting weights to balance neurons. Weight updates depend on the learning rate, which affects precision. The update speed can be adjusted by adjusting the learning rate. While updating weights during backpropagation, the model becomes confounding. The precise calculation of the weight gradient is crucial for optimal measurement. The learning rate follows the gradient to assign values to weights. Increasing the learning rate increases the weight closer to the optimal value. High learning rates impact convergence. Model neglect occurs without a detailed learning rate analysis, which affects accuracy. Underfitting arises when weights lack precision, resulting in incorrect classification. Decreasing the learning rate emphasizes overfitting.

$$\frac{\partial L}{\partial w_l} = \delta_l \cdot (A_{l-1})^T \quad (14)$$

**B- Gradient of biases:** The bias update process in DNN training requires gradient calculation. Biases have a similar role to weights in DNNs. The calculation of the unfrozen bias can be straightforwardly performed. The biases are beneficial, as they allow the network to shift the activation function and learn quickly. The gradient of biases determines the shift direction and proportion. Adjusting biases can increase network activity. The joint update of weights and biases enhances the network's power. The optimization of biases aims to balance the step size between layers and improve generalization. Biases are essential for the DNN architecture and training process.

$$\frac{\partial L}{\partial b_l} = \delta_l \quad (15)$$

## 4.6 Weight Update Using an Optimizer

### 4.6.1 First and Second Moment Estimates

$$\text{The } m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w} \quad (16)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left( \frac{\partial L}{\partial w} \right)^2 \quad (17)$$

### 4.6.2 Bias-corrected estimates and weight update

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (18)$$

$$W = W - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \quad (19)$$

where  $\alpha$  is the learning rate;  $\beta_1$  and  $\beta_2$  are decay rates for the moment estimates; and  $\epsilon$  is a small constant to prevent division by zero.

---

Algorithm 3 presents the DNN initialization and training stage.

---

"Algorithm 3 " Deep Neural Network Initialization, Training, and Evaluation

"Require: " Input training data  $X_{train}$ , training labels  $y_{train}$ , test data  $X_{test}$ , test labels  $y_{test}$ , selected feature subset  $S$ ,

number of epochs  $num\_epochs$ , learning rate  $\eta$

"Ensure: " Trained DNN model and its evaluation accuracy on test data

" 1: Step 1: Initialize DNN Architecture"

" 2: " Define a sequential DNN model  $model$

" 3: " Add input layer with  $|S|$  input neurons and ReLU activation function

" 4: " Add hidden layer with 64 neurons and ReLU activation function

" 5: " Add second hidden layer with 32 neurons and ReLU activation function

" 6: " Add output layer with 1 neuron and Sigmoid activation function

" 7: "  $model \leftarrow \{Input: |S|, Hidden1: 64, Hidden2: 32, Output: 1\}$

" 8: Step 2: Compile the Model"

" 9: " Set optimizer to Adam with learning rate  $h$

"10: " Set loss function to Binary Cross-Entropy

"11": Set metrics to monitor as Accuracy

"12: "  $model.compile(optimizer=Adam(\eta), loss='binary_crossentropy', metrics=['accuracy'])$

"13: Step 3: Train the DNN"

"14: " Extract training features corresponding to  $S: X_{train}(S) \leftarrow X_{train}[S]$

"15: " Train the model for  $num\_epochs$  epochs with batch size 32:

"16: "  $model.fit(X_{train}(S), y_{train}, epochs=num\_epochs, batch\_size=32)$

---

---

```

"17: Step 4: Evaluate the DNN on Test Data"
"18: " Extract test features corresponding to  $S: X_{(test,S)} \leftarrow X_{test} [S]$ 
"19: " Calculate evaluation accuracy:
"20: "  $accuracy \leftarrow model.evaluate(X_{(test,S)}, y_{test})$ 
"21: Step 5: Return the Trained Model and Accuracy"
"22: return " model, accuracy

```

---

The fourth pseudocode describes the proposed AntDroidNet cybersecurity model and illustrates how the ACO integrates with the DNN and the stages of developing the proposed AntDroidNet cybersecurity model in detail.

## 5. RESULTS AND DISCUSSION

This section presents a comprehensive analysis of the experimental results obtained from implementing the proposed AntDroidNet cybersecurity model. Multiple performance metrics are included in this evaluation framework, where more attention is given to the confusion matrix, which is the initial method that can show the predictive and classification performance of the model. This matrix serves to effectively illustrate both the model's capacity to classify applications accurately into malware or benign categories and to identify any misclassifications during the evaluation process.

### 5.1 Dataset

By correctly classifying Android applications as malware, cybersecurity researchers contend that developing effective classification and detection methodologies is a major challenge. The current study uses the CICMalDroid 2020 dataset [25] for training and evaluating the proposed model to overcome this issue. The complete datasets include different types of potentially harmful software (adware, banking trojans, riskware and SMS-based threats) as well as normal benign apps for comparison. Having malicious and benign samples allows one to know the features that mention differences in malware. The numerical distribution of the samples is detailed in Table I, which provides quantitative insight into the composition of the dataset and draws attention to the diversity of threats represented in this study.

---

#### Algorithm 4 Proposed AntDroidNet Cybersecurity Model

---

```

"Require: " Raw dataset  $X$  with  $m$  samples and  $n$  features, labels  $y$ , number of Ants  $num\_ants$ , number of iterations  $num\_iterations$ ,
ACO parameters  $\alpha, \beta$ , evaporation rate  $P$ , DNN parameters: number of epochs  $num\_epochs$ , learning rate  $\eta$ 
"Ensure: " Best subset of selected features  $S_{best}$  and trained DNN model with optimal classification accuracy
" 1: Step 1: Initialize Pheromone Levels for ACO"
" 2: "  $pheromone \leftarrow \{\tau_1=1, \tau_2=1, \dots, \tau_n=1\}$  {Initial pheromone levels are equal for all features}
" 3: "  $S_{best} \leftarrow \emptyset, best\_accuracy \leftarrow 0$ 
" 4: for " each iteration  $k=1$  to  $num\_iterations$ " do
" 5:   Step 2: Feature Subset Construction by Ants"
" 6:   for " each ant  $i=1$  to  $num\_ants$ " do
" 7:     "  $S_i \leftarrow \emptyset$  {Start with an empty feature subset}
" 8:     for " each feature  $f \in F$ " do
" 9:       " Calculate selection probability using equation number 3:
"10:      if " randomly generated number  $\leq P(f)$ " then
"11:        "  $S_i \leftarrow S_i \cup \{f\}$ 
"12:      end if
"13:    end for
"14:  end for
"15:  Step 3: Evaluate Feature Subsets Using DNN"
"16:  for " each ant  $i=1$  to  $num\_ants$ " do
"17:    " Extract relevant features  $X_{(train, S_i)} \leftarrow X_{train} [S_i]$ 
"18:    " Define and compile DNN with  $|S_i|$  input neurons, hidden layers (64 and 32 neurons), and sigmoid output layer
"19:    " Train DNN on  $X_{(train, S_i)}$  and "  $y_{(train, S_i)}$  for "  $num\_epochs$ 
"20:    " Evaluate model on  $X_{(test, S_i)}$  to obtain  $accuracy(S_i)$ 
"21:  end for
"22:  Step 4: Update Best Feature Subset and Accuracy"
"23:  "  $best\_ant \leftarrow \underset{i \in \{1, \dots, num\_ants\}}{\text{argmax}} accuracy(S_i)$ 
"24:  if "  $accuracy(S_{(best\_ant)}) > best\_accuracy$ " then

```

---

---

```

"25:   "  $S_{best} \leftarrow S_{(best\_ant)}$ ,  $best\_accuracy \leftarrow accuracy(S_{(best\_ant)})$ 
"26:   end if"
"27:   Step 5: Pheromone Update"
"28:   for " each feature  $f \in F$  " do"
"29:       " Adjust pheromone level using equation number 5:
"30:   end for"
"31: end for"
"32: Step 6: Final Training and Testing of the Optimized DNN"
"33: " Extract features from  $S_{best}$ :  $X_{(train, S_{best})} \leftarrow X_{train} [S_{best}]$ 
"34: " Define, compile, and train DNN using  $S_{best}$  as input features
"35: " Final evaluation on  $X_{(test, S_{best})}$ :  $final\_accuracy \leftarrow model.evaluate(X_{(test, S_{best})}, y_{test})$ 
"36: return "  $S_{best}$ ,  $model$ ,  $final\_accuracy$ 

```

---

TABLE I. CICMALDROID 2020 DATASET

App Class	Family	No. of App
Malware	Adware	1253
	Banking	2100
	SMS	3904
	Riskware	2546
Benign	Benign	1795

## 5.2 Evaluation Metrics

In the methodology section of the performance assessment, four standard metrics are defined: accuracy, precision, recall and the F1 score. That is, along with the metric MCC. These established metrics were selected to allow for a comprehensive and objective assessment of the proposed AntDroidNet cybersecurity model in terms of malware detection capabilities and benign versus malicious application classification ability. The MCC is particularly useful for imbalanced datasets, as it considers all the elements of the confusion matrix (true positives, true negatives, false positives and false negatives) and provides a balanced and comprehensive evaluation. In cases where classes are not evenly distributed, the MCC is a better metric for measuring both accuracy and the ability to classify both positive and negative samples. The robustness and reliability of the proposed AntDroidNet cybersecurity model are validated by these performance statistics. This careful structure allows for the systemic evaluation of the success of the proposed AntDroidNet cybersecurity-based model and provides valuable insights into defining its practical implications in the real-world sense. The four confusion matrices are depicted below in Figure 2 and Table II of the proposed AntDroidNet cybersecurity model.

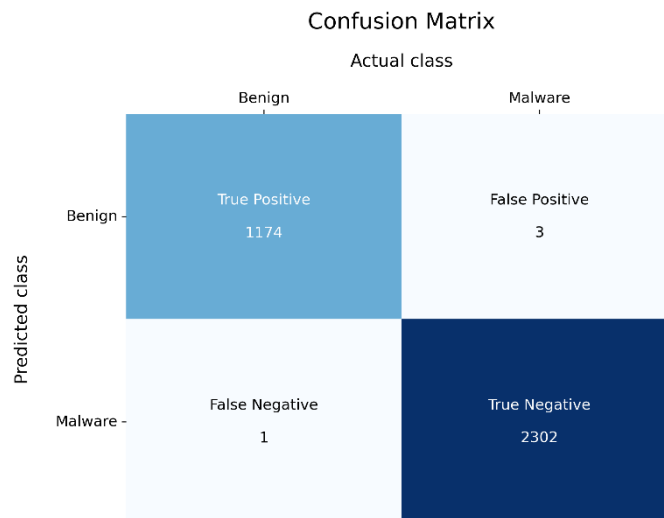


Fig. 2. Confusion matrix for the proposed AntDroidNet cybersecurity model.

TABLE II. TP, FP, FN, AND TN FOR THE PROPOSED ANTDRROIDNET CYBERSECURITY MODEL.

TP (%)	FP (%)	FN (%)	TN (%)
99.91	0.13	0.09	99.87

A confusion matrix was employed to evaluate the performance of the proposed model in classifying mobile applications as either malware or benign. The empirical results demonstrate the model's exceptional predictive capabilities. Specifically, the model achieved a remarkable 99.91% TP in correctly identifying malware applications. With respect to FPs, the model exhibited outstanding precision, with only 0.13% of benign applications being incorrectly classified as malware a rate significantly below 1%, indicating superior discriminative capabilities between malware and benign applications. Furthermore, the model demonstrated exceptional TN in correctly identifying benign applications, achieving a 99.87% true negative rate. The FN was particularly impressive at merely 0.09%, representing cases where malware applications were incorrectly classified as benign. This exceptionally low rate further validates the model's robustness. These confusion matrix results underscore the proposed model's remarkable ability to accurately distinguish between malware and benign applications. These performance metrics indicate the model's potential for reliable deployment in real-world mobile security applications, where precise discrimination between malware and benign software is crucial.

Table III shows the detection accuracy of the proposed AntDroidNet cybersecurity model in addition to the other four performance metrics: precision, recall, F1 score, and MCC.

TABLE III. PERFORMANCE METRICS FOR THE PROPOSED ANTDRROIDNET CYBERSECURITY MODE.

Method	Performance Metrics (%)				
	Accuracy	Precision	Recall	F1score	MCC
Classification	98.99	98.39	98.64	98.51	97.75
Detection	99.89	99.75	99.91	99.83	99.74

Table III presents the performance metrics for malware classification and detection utilizing the CICMalDroid 2020 dataset. Through optimization of the feature selection process, AntDroidNet is the most advanced model with classification and detection results. The method obtained an optimized dataset extracted from the original CICMalDroid 2020 dataset by utilizing the most impactful features to present a direct influence on classification rates. The metrics for performance reflect the fact that identifying and preserving the most relevant image characteristics greatly reduces redundancy or lowers impact features as an optimization approach. This fine-tuning approach is critical for improving the model's detection capabilities and classification performance. This methodological advancement in feature selection can provide configure accuracy in terms of mobile malware detection because this ability to detect the most suitable and relevant level of features can be advantageous and allows high-level classification in target classes. For both the detection and classification of malware and benign applications during the performance evaluation phase of the proposed AntDroidNet cybersecurity model, MCC is used as its well-balanced implementer for evaluation metrics after classifying the malware data. MCC is a holistic statistic collapsing all classification results positive or negative: TP, TN, FP and FN from the confusion matrix into a single score that represents all four values. The importance of this metric is that it describes an equal assessment of the AntDroidNet cybersecurity model on the basis of how well it correctly determines whether a given piece of software is malware while also avoiding the misclassification of benign applications that would not be otherwise labelled as such. This exhaustive end-to-end reasoning vector guarantees unwavering quality execution measurements and prompts fewer bogus alerts and overlooks malware programs. Therefore, using MCC is a powerful approach for evaluating the overall performance of the AntDroidNet cybersecurity model, which furnishes more in-depth information from common performance measures while providing a more truthful assessment of its effectiveness in real applications involving malware detection.

### 5.3 Comparing Results in the Literature

The final performance evaluation of the proposed AntDroidNet cybersecurity model is compared against the literature, which uses the same CICMalDroid2020 dataset and evaluation measures for the detection of Android malware. Table IV and Figure 3 summarize the comparative analysis, revealing significant performance advantages of the proposed AntDroidNet cybersecurity model, especially in terms of accuracy, precision, recall, and F1-score metrics, among the other research comparisons. The comparative analysis underpins the excellent discriminative performance of the proposed AntDroidNet cybersecurity model for malware detection. By using an extensive, standardized evaluation framework, in terms of metrics and datasets, this method allows for strong validation of the model capabilities. As they provide a robust model for Android

malware detection, the proposed AntDroidNet cybersecurity model demonstrates an overall improvement in adversarial robustness, as reflected specifically in the empirical findings. This study not only validates the effectiveness of the model but also contributes to the progress in the mobile security domain, especially android malware detection systems.

TABLE IV. PERFORMANCE METRICS FOR THE PROPOSED ANTDRROIDNET CYBERSECURITY MODE.

Method	Performance Metrics (%)			
	Accuracy	Precision	Recall	F1score
[24]	96.7	99.16	96.54	97.84
[23]	98.66	99.15	99.06	99.10
[25]	98.70	98.7	98.7	98.7
[22]	98.96	99	99.65	99.33
[26]	98.97	99.23	99.50	99.36
[27]	99	99	99	99
[28]	99	99	100	99
Proposed AntDroidNet	99.89	99.75	99.91	99.83

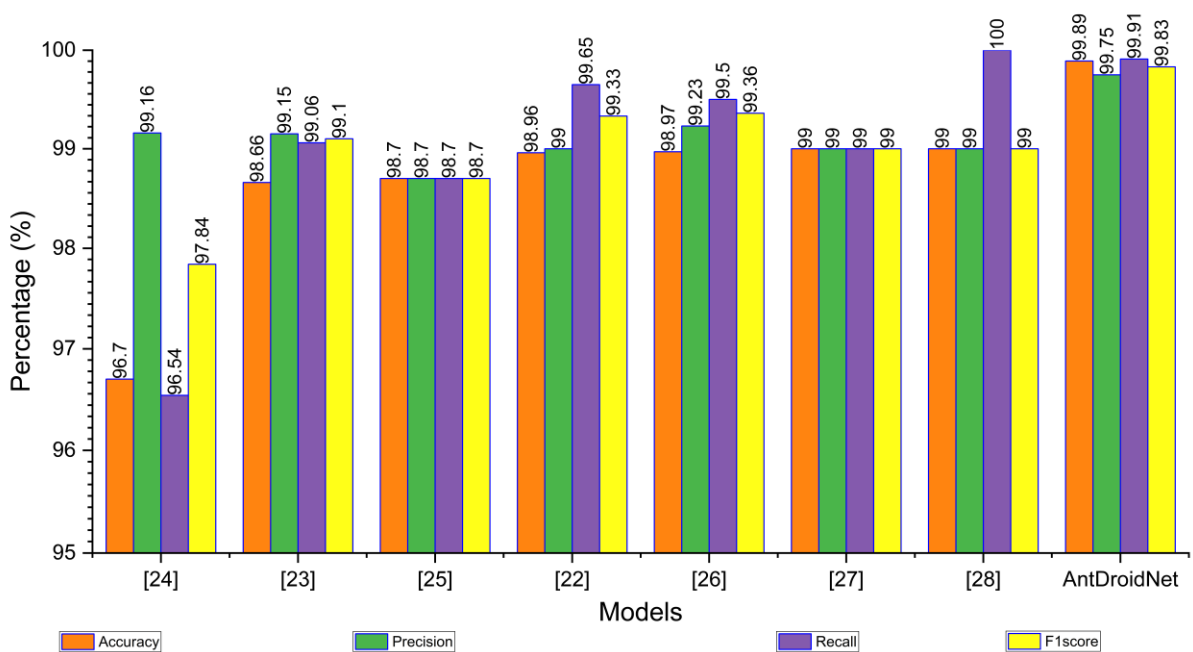


Fig. 3. Results of the comparison of the proposed AntDroidNet cybersecurity model with related models.

According to the figure above, when the proposed AntDroidNet cybersecurity model is compared with existing frameworks in the literature, under comparable conditions utilizing identical CICMaDroid 2020 datasets and performance metrics such as accuracy, precision, recall, and F1-scores, the improvements in detection accuracy achieved by the proposed AntDroidNet cybersecurity model are noteworthy. Specifically, compared with models [25][24][26][23] and [28], the enhancements in accuracy represent a significant advancement over previous approaches. With respect to models [29] and [30], the proposed AntDroidNet cybersecurity model demonstrated a marked improvement in detection accuracy, surpassing these models by 0.89%. This enhancement underscores the model's robust predictive ability in identifying cyber threats.

In terms of precision, while all compared models exhibited similar results, the proposed AntDroidNet cybersecurity model achieved a 0.5% percentage point improvement, reflecting enhanced classification accuracy in its predictions.\

For the recall metric, model [30] achieves a perfect score of 100%, with the proposed AntDroidNet cybersecurity model closely following at 99.91%. In terms of the final F1 score metric, the proposed AntDroidNet cybersecurity model once again performs excellently, achieving the highest F1 score of approximately 99.83% among all the compared models and approaching near-perfect performance.

Moreover, the false positive rate of the proposed AntDroidNet cybersecurity model is remarkably low at 0.13%, which is ideal in practical applications.

These results suggest that the proposed model outperforms the compared frameworks and holds promise for real-world implementation, potentially contributing significantly to addressing cybersecurity challenges.

## 6. CONCLUSION

In this paper, a novel AntDroidNet cybersecurity model based on hybrid deep neural networks with ant colony optimization is proposed for feature selection to detect comprehensive mobile malware effectively. The experimental outcomes demonstrate the effectiveness of combining deep learning and swarm intelligence and exploiting them for the field of mobile malware detection. It is an advanced and novel mobile malware detection system based on a hybrid DNN and ACO. The historical outcome of our findings will be thoroughly documented, and the response rates, clarity and MCC will be assessed. The abovementioned immense contributions of our proposed AntDroidNet cybersecurity model can be summarized as follows: (i) It has been proven to be an unprecedented viable technique of hybrid DNN amalgamation along with ant ACO for feature selection. Combining the strengths of DNN and ACO, our system has been shown to have high precision. The successful experimental results clearly show that ACO is an improved technique for mobile malware detection. These pioneering research results also establish that hybrid DNNs with ACO have only a few applications not only in the case of mobile malware but also in other cybersecurity cases, resulting in an excellent and reliable detection model. (ii) Additionally, one of the strongest advantages of our AntDroidNet cybersecurity architecture is that it serves as an amazing architecture guidance tool for malware-based architectures. This novel AntDroidNet model can be estimated to be effective in the industry for more than just mobile malware detection systems. By doing so, its efficiency and adaptability can remarkably increase the performance of real-time detection, which gives it strong utility in real-time detection systems, especially in sectors that require continuous security monitoring, including but not limited to mobile apps and Internet of Things (IoT) hardware. Since IoT networks are expanding at an unprecedented pace and the sophistication of cyber attacks continues to evolve, this work may contribute to the establishment of security architectures targeted toward large-scale, real-time environments. Our research has the capacity to aid the development of more resilient, scalable and adaptable security solutions in the areas of mobility and the IoT because it addresses the rapidly changing landscape of these threats.

Future work might explore additional optimization of the model to increase its real-time detection efficiency and make the system deployable on a wider range of mobile devices and future malware families. Furthermore, you would focus on making the computation of the model efficient enough to be used beyond the model itself, such as in dated versions of certain mobile and IoT environments for widespread, real-world deployment. These upgrades help keep the model relevant as the cybersecurity industry continues to advance, tackling mobile and IoT security threats that become more sophisticated over time.

### Conflicts of interest

No conflicts of interest exist. We wish to confirm that there are no known conflicts of interest associated with this publication.

### Funding

The lack of funding acknowledgement in the paper indicates that no financial support was provided by any institution or sponsor.

### Acknowledgement

The author's is grateful to the institution for their collaboration and provision of necessary facilities that contributed to the successful completion of this research

### References

- [1] C. S. Yadav *et al.*, "Malware analysis in IoT & android systems with defensive mechanism," *Electronics*, vol. 11, no. 15, p. 2354, 2022.
- [2] A. Kovács, "Ransomware: a comprehensive study of the exponentially increasing cybersecurity threat," *Insights into Reg. Dev.*, vol. 4, no. 2, pp. 96–104, 2022.

- [3] S. M. Hadi, A. H. Alsaeedi, M. I. Dohan, R. R. Nuijaa, S. Manickam, and A. S. D. Alfoudi, "Dynamic Evolving Cauchy Possibilistic Clustering Based on the Self-Similarity Principle (DECS) for Enhancing Intrusion Detection System," *Int. J. Intell. Eng. Syst.*, vol. 15, no. 5, pp. 252–260, 2022, doi: 10.22266/ijies2022.1031.23.
- [4] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, "A survey on ransomware: Evolution, taxonomy, and defense solutions," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 1–37, 2022.
- [5] M. Al-Hawawreh, M. Alazab, M. A. Ferrag, and M. S. Hossain, "Securing the Industrial Internet of Things against ransomware attacks: A comprehensive analysis of the emerging threat landscape and detection mechanisms," *J. Netw. Comput. Appl.*, p. 103809, 2023.
- [6] L. Chen, C. Xia, S. Lei, and T. Wang, "Detection, traceability, and propagation of mobile malware threats," *IEEE Access*, vol. 9, pp. 14576–14598, 2021.
- [7] A. S. A. Albahri, M. G. Yaseen, M. Aljanabi, A. H. A. H. Ali, and A. Kaleel, "Securing tomorrow: navigating the evolving cybersecurity landscape," *Mesopotamian J. CyberSecurity*, vol. 4, no. 1, pp. 1–3, 2024.
- [8] Ö. Aslan, S. S. Aktuğ, M. Ozkan-Okay, A. A. Yilmaz, and E. Akin, "A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions," *Electronics*, vol. 12, no. 6, p. 1333, 2023.
- [9] Z. Wang, Q. Liu, and Y. Chi, "Review of android malware detection based on deep learning," *IEEE Access*, vol. 8, pp. 181102–181126, 2020.
- [10] P. G. Meenakshi and P. Shrivastava, "Machine learning for mobile malware analysis," *Cyber Crime Forensic Comput. Mod. Princ. Pract. Algorithms*, vol. 11, p. 151, 2021.
- [11] M. Gopinath and S. C. Sethuraman, "A comprehensive survey on deep learning based malware detection techniques," *Comput. Sci. Rev.*, vol. 47, p. 100529, 2023.
- [12] R. R. Nuijaa, A. H. Alsaeedi, S. Manickam, D. E. J. Al-Shammary, A. H. Alsaeedi, and D. E. J. Al-Shammary, "Evolving Dynamic Fuzzy Clustering (EDFC) to Enhance DRDoS DNS Attacks Detection Mechanism," *Int. J. Intell. Eng. Syst.*, vol. 15, no. 1, pp. 509–5019, 2022, doi: 10.22266/ijies2022.0228.46.
- [13] S. P. Rao, H.-Y. Chen, and T. Aura, "Threat modeling framework for mobile communication systems," *Comput. Secur.*, vol. 125, p. 103047, 2023.
- [14] R. R. Nuijaa et al., "Enhanced PSO Algorithm for Detecting DRDoS Attacks on LDAP Servers.," *Int. J. Intell. Eng. Syst.*, vol. 16, no. 5, 2023.
- [15] H. AlOmari, Q. M. Yaseen, and M. A. Al-Betar, "A comparative analysis of machine learning algorithms for android malware detection," *Procedia Comput. Sci.*, vol. 220, pp. 763–768, 2023.
- [16] R. R. N. Al Ogaili et al., "Malware cyberattacks detection using a novel feature selection method based on a modified whale optimization algorithm," *Wirel. Networks*, pp. 1–17, 2023.
- [17] L. A. E. Al-saeedi, F. J. Shakir, F. K. Hasan, G. G. Shayea, Y. L. Khaleel, and M. A. Habeeb, "Artificial Intelligence and Cybersecurity in Face Sale Contracts: Legal Issues and Frameworks," *Mesopotamian J. Cybersecurity*, vol. 4, no. 2, pp. 129–142, 2024.
- [18] R. Islam, M. I. Sayed, S. Saha, M. J. Hossain, and M. A. Masud, "Android malware classification using optimum feature selection and ensemble machine learning," *Internet Things Cyber-Physical Syst.*, vol. 3, pp. 100–111, 2023.
- [19] J. Yang, H. Li, L. He, T. Xiang, and Y. Jin, "MDADroid: A novel malware detection method by constructing functionality-API mapping," *Comput. Secur.*, vol. 146, p. 104061, 2024.
- [20] Y. Zhou, G. Cheng, S. Yu, Z. Chen, and Y. Hu, "MTDroid: A Moving Target Defense based Android Malware Detector against Evasion Attacks," *IEEE Trans. Inf. Forensics Secur.*, 2024.
- [21] A. I. Gide and A. A. Mu'azu, Trans., "A Real-Time Intrusion Detection System for DoS/DDoS Attack Classification in IoT Networks Using KNN-Neural Network Hybrid Technique ", *BJIoT*, vol. 2024, pp. 60–69, Jul. 2024, doi: 10.58496/BJIoT/2024/008.
- [22] P. Tarwireyi, A. Terzoli, and M. O. Adigun, "Meta-SonifiedDroid: Metaheuristics for Optimizing Sonified Android Malware Detection," *IEEE Access*, 2024.
- [23] P. Tarwireyi, A. Terzoli, and M. O. Adigun, "Using multi-audio feature fusion for android malware detection," *Comput. Secur.*, vol. 131, p. 103282, 2023.
- [24] N. Xie, Z. Qin, and X. Di, "GA-StackingMD: Android malware detection method based on genetic algorithm optimized stacking," *Appl. Sci.*, vol. 13, no. 4, p. 2629, 2023.
- [25] S. Mahdavarfar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic android malware category classification using semi-supervised deep learning," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCCom/CyberSciTech)*, IEEE, 2020, pp. 515–522.
- [26] H. H. R. Manzil and S. Manohar Naik, "Android malware category detection using a novel feature vector-based machine learning model," *Cybersecurity*, vol. 6, no. 1, 2023, doi: 10.1186/s42400-023-00139-y.



- [27] H. A. Salman and A. Alsajri, “The Evolution of Cybersecurity Threats and Strategies for Effective Protection. A review”, SHIFRA, vol. 2023, pp. 73–85, Aug. 2023, doi: 10.70470/SHIFRA/2023/009.
- [28] A. R. Nasser, A. M. Hasan, and A. J. Humaidi, “DL-AMDet: Deep learning-based malware detector for android,” *Intell. Syst. with Appl.*, vol. 21, p. 200318, 2024.
- [29] F. Ullah, S. Ullah, M. R. Naeem, L. Mostarda, S. Rho, and X. Cheng, “Cyber-threat detection system using a hybrid approach of transfer learning and multi-model image representation,” *Sensors*, vol. 22, no. 15, p. 5883, 2022.
- [30] R. Surendran, M. M. Uddin, T. Thomas, and G. Pradeep, “Android Malware Detection Based on Informative Syscall Subsequences,” *IEEE Access*, 2024.
- [31] Y. L. Khaleel, M. A. Habeeb, and H. Alnabulsi , Trans., “Adversarial Attacks in Machine Learning: Key Insights and Defense Approaches ”, *Applied Data Science and Analysis*, vol. 2024, pp. 121–147, Aug. 2024, doi: 10.58496/ADSA/2024/011.