

Optimized Deep Learning for Android Malware Detection A PSO-LSTM Hybrid Approach

Mokri Miloud Aboubakeur El Sadek¹, Bensaid Tayeb², Souidi Mohamed Abdou³, Zairi Khadidja⁴, Abbas Imad Eddine⁵

¹Department of Computer Science, Faculty of Mathematics, Computer Science, and Telecommunications, Dr. Tahar Moulay University of Saida, Saida, Algeria.

²Department of Computer Science, Faculty of Mathematics, Computer Science, and Telecommunications, Dr. Tahar Moulay University of Saida, Saida, Algeria.

³Department of Computer Science, Faculty of Mathematics, Computer Science, and Telecommunications, Dr. Tahar Moulay University of Saida, Saida, Algeria.

⁴Laboratory of Computer Science and Mathematics, Department of Computer Science, Faculty of Science and Technology, University Amar Telidji of Laghouat, Laghouat, Algeria.

⁵Department of Mathematics, Faculty of Mathematics, Computer Science, and Telecommunications, Dr. Tahar Moulay University of Saida, Saida, Algeria.

ARTICLE INFO

ABSTRACT

Received: 06 Aug 2026

Accepted: 22 Sept 2026

Published: 03 Oct 2026

Introduction: The widespread adoption of Android as the most used mobile operating system has made it a prime target of malicious applications that exploit its permission-based access control system. While this access system helps to protect user data, it is misused by malware to gain unauthorized access to sensitive information. As a result, getting an efficient Android malware detection has become a critical area of research.

Objectives: This paper addresses the problem by exploring multiple machine learning approaches to detect malware based on permission. Furthermore, to enhance the detection performance and model robustness, we propose a hybrid approach that combines PSO and LSTM neural networks.

Methods: In our hybrid approach, PSO is employed to automatically optimize the hyperparameters of the LSTM model. We evaluate our approach using three publicly available Android malware datasets. The first is the Malware/Benign Permissions Android dataset (C.Urcuqui and A.Navarro 2016, U.Christian 2016), the second is the Drebin dataset (Soju n.d.), and the third is the Droid det malware dataset (F. Akbar 2025).

Results: The experimental results demonstrate that the hybrid model PSO-LSTM outperforms traditional machine learning techniques in terms of detection performance. We also compare our results with other existing studies in the field, showing that our method offers a more effective and scalable solution for Android malware detection.

Conclusions: The PSO-LSTM hybrid model enhances the detection of Android malware by the optimization power of PSO in the selection of hyperparameters of the LSTM. This method offers a scalable solution for malware detection, providing an effective approach that surpasses existing models in the field.

Keywords: Android, permission, malware, machine learning, PSO, optimization, LSTM.

INTRODUCTION

Mobile devices have significantly evolved over the years, offering more features and capabilities than many traditional computing devices. This advancement has led to the rise of smartphones. Android has emerged as the most popular and widely used mobile operating system, it powers millions of devices, not only mobile phones but also TVs, smartwatches and other smart devices [7]. In the third quarter of 2011, Android acquired over 50% of smartphone sales [8]. Google Play, the official app marketplace for Android, drives the economy of mobile applications. For example, Google Play has exceeded more than 50 billion downloaded apps, producing over 5 billion dollars in revenue

in 2013 [9]. With more than a billion activated devices and over a billion monthly active users, the platform has unfortunately also become a prime target of malicious exploitation, which is presented in the various attacks from hackers to spy on user data. Cyber attackers have increasingly focused on Android, using mobile malware to bypass device protection mechanisms and access sensitive user data. These attacks often aim to paralyze the security and exploit permissions to spy on users or extract private information. A recent report highlighted a “400% increase in Android-based malware since the summer of 2010” [10]. By the end of 2011, Android had become the most targeted mobile platform for malware due to its rapid growth and largest adoption [11].

Android implements an access control model through an application permissions mechanism, allowing users to define how applications interact with system resources. This model aims to ensure data security by asking for explicit permission before performing sensitive operations like accessing contacts, location, camera, and messages [12]. However, malicious applications frequently exploit this by requesting excessive or unnecessary permissions in order to perform harmful actions such as stealing data, installing spyware, and even abusing personal information [13].

To prevent such intrusions, there is a growing need for effective solutions that are able to detect breaches. In this context, multiple researchers have proposed a variety of techniques, including artificial intelligence, data mining algorithms, deep learning approaches, and even bio-inspired methods to improve Android malware detection systems. As a result, researchers and security professionals continue to try to explore new technologies aimed at Android malware detection and enhancing user security [14]. In our project, we had used three different datasets for Android malware detection mentioned previously, the first is named as the Malware/Benign Permissions Android dataset [1,2], the second dataset is named as Drebin [3], and the third is named the Droid det malware dataset [4]. Our proposed model, which is a hybrid model, is tested with these three datasets. The obtained results were compared to other works already done. For the first dataset, we didn't find multiple researches using this benchmark, so we applied classification methods using selected data mining algorithms from the library MLlib in Python [15] to compare and evaluate our work. In our proposed model, a key challenge in using LSTM is determining the optimal hyperparameters to achieve the best classification performance. To address this, we combined a bio-inspired algorithm, which is the PSO, with a deep learning technique, which is the LSTM. The PSO algorithm was used to optimize the hyperparameters of the LSTM network, the optimal hyperparameters obtained are used to classify malware based on application features. The obtained results are compared and evaluated with other works already done and with other algorithms. The objective of this comparison was to evaluate the performance of our proposed solution and identify the most efficient Android malware detector by comparing it with existing works in the field applying to the same datasets.

Currently, Android is operating on many personal as well as professional devices, its dominance in the mobile OS market having made it a major target of malware attacks. Over the years, in order to secure sensitive information, a range of Android malware detection techniques has been proposed, mainly divided into spanning static analysis, dynamic behavior monitoring, deep learning, and hybrid methods.

This section categorizes the most impactful and high-quality studies across these paradigms and critically compares them to our proposed PSO-LSTM hybrid model.

Static analysis methods leverage features like permissions, API calls, and manifest data extracted without running the app. Felt et al. (2011) developed Stowaway, a system that detects overprivileged applications by checking requested permissions against observed APIs. Their research highlighted the fact that numerous applications over request permissions, which results in higher security threats [16]. Similarly, Arp et al. (2014) proposed Drebin, a lightweight malware detection system that quickly extracts static features and applies support vector machines (SVM) for classification [17]. Sahs and Khan (2012) used Naïve Bayes and Support Vector Machines (SVM) models based on permissions for malware detection [18]. While these methods are computationally efficient, they often struggle against code hiding and cannot capture runtime behaviors exploited by malware.

Dynamic analysis approaches monitor app behavior during execution. Shabtai et al. presented a framework for detecting malware on Android mobile devices. His proposed framework realizes a host-based malware detection system that monitors an Android user to continuously capture various features and events from the mobile device and after that employs machine learning anomaly detectors that help in classifying the captured information as either

benign or augmentative [19]. Dini et al. introduced a multi-level anomaly detector for Android malware. At the kernel and user levels, MADAM-policing analyzes Android kernel and user levels to find real malware infections using machine learning to tell the difference between normal actions and harmful ones [20]. Dynamic analysis improves detection robustness but suffers from high computational overhead and challenges in scalability.

Deep learning-based detection models have been increasingly applied due to their ability to learn complex feature interactions. Huang et al. (2018) developed a deep neural network (DNN) that automatically learned feature-based permissions. His model bested the traditional classifiers because it incorporated the non-linear interaction between permissions and harmful actions [21]. Sgandurra et al. (2016) developed MalwareDetector, which is an LSTM-based recurrent neural network. This model studied patterns of granted permissions, it analyzes permission sequences, achieving better classification accuracy [22]. Fan et al. (2020) used a hybrid deep learning model that is based on permission features and method call sequences to detect malware. The results proved superior compared to the use of ordinary machine learning techniques [23]. These models demonstrate high performance, the effectiveness is often reliant on manual hyperparameter tuning, which is often time consuming and leads to suboptimal results.

Hybrid and optimization-based methods integrate feature types or optimize model parameters for better detection. Aafer et al. (2013) built DroidAPIMiner, which is a system that integrated the analysis of software permission with the sequences of API calls to improve malware detection accuracy [24]. Feizollah et al. (2015) designed a hybrid system that uses permission-based features and network traffic data, showing increased classification performance [25]. However, few studies have incorporated bio-inspired optimization techniques to tune deep learning models hyperparameters.

Our work differentiates itself by applying Particle Swarm Optimization (PSO) to optimize the LSTM hyperparameters, enhancing detection performance while maintaining a lightweight permission-based feature set.

Table 1. Comparative Summary of Works Mentioned.

Study	Approach	Features	Methodology	Strengths	Limitations
Felt et al. [16]	Static	Permissions	Rule-based	Lightweight, interpretable	Limited to static features
Sahs and Khan [18]	Static	Permissions	Naïve Bayes, SVM	Good baseline	Static only, manual feature design
Arp et al. [17]	Static	Permissions, API calls	SVM	Efficient, scalable	Obfuscation vulnerability
Shabtai et al. [19,20]	Dynamic	Runtime events	Anomaly detection	Real-time detection	High overhead
Huang et al. [21]	Deep Learning	Permissions	DNN	Nonlinear feature interaction	Manual hyperparameter tuning
Sgandurra et al. [22]	Deep Learning	Permission sequences	LSTM	Captures temporal relations	No automated hyperparameter tuning

Fan et al. [23]	Hybrid	Permissions + call seq.	Hybrid Deep Learning	High accuracy	Complex, no optimization
Aafer et al. [24]	Hybrid	Permissions + API calls	Static dynamic	+ Improved accuracy	Higher feature extraction cost
Feizollah et al. [25]	Hybrid	Permissions + Net. traffic	ML classifiers	Better accuracy	High computational cost
This work	Hybrid Optimized	+ Permissions only	PSO LSTM	+ Automated tuning, lightweight, accurate	No dynamic features for simplicity

OBJECTIVES

The Android platform has grown at an astonishing rate, now holding the top position as the most widely used operating system across countless devices. This rapid growth has made Android the target of many malware applications that want to steal sensitive information. Many efforts have been made in the direction of intrusion detection. In this project we want to detect malware through a hybrid approach that combines a deep learning technique and a bio-inspired method. Thereafter, we compare the results obtained to find the best detector of Android malware applications based on permissions.

Permissions serve the purpose of protecting the privacy of an Android user. If the user is on Android version 6.0 or higher, they will have to either accept or deny access permissions for sensitive information during the application installation procedure, for example, the access request where we install the VLC application (see **Error! Reference source not found.**), as illustrated.

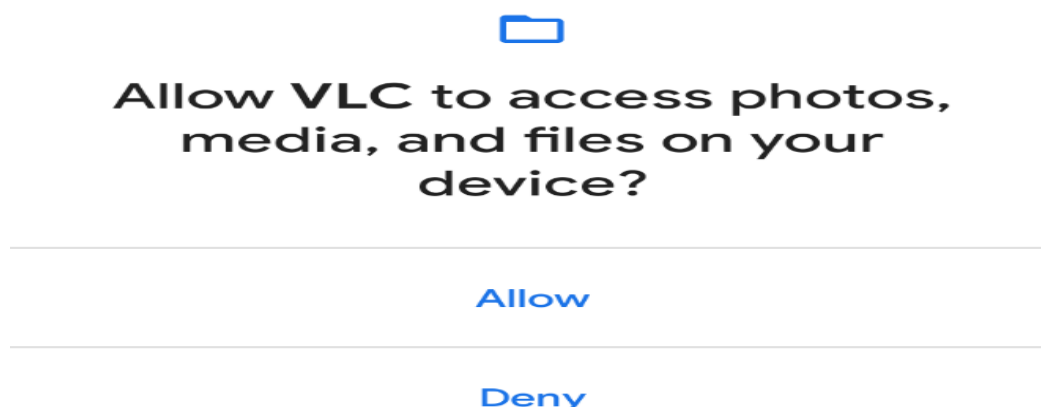


Fig. 1. Example of permission access request [26].

Malware is inappropriate access to data that can be a big problem for Android user's, because of the big use of the Android OS on multiple sensitive devices. This has also caused an increase in a different type of illegal requests [27,28,29,30,31]. Those malwares can spy on people and steal a lot of financial data like banking transfers, also getting the data in illegitimate way...etc [32,33,34,35]. This area of research is an important and sensitive domain following the important of these Android devices. So, increasing the security is a main issue.

The hall process steps that were followed in order to classify the applications and make a decision whether they are a malware application or benign is shown below (see Fig. 2).

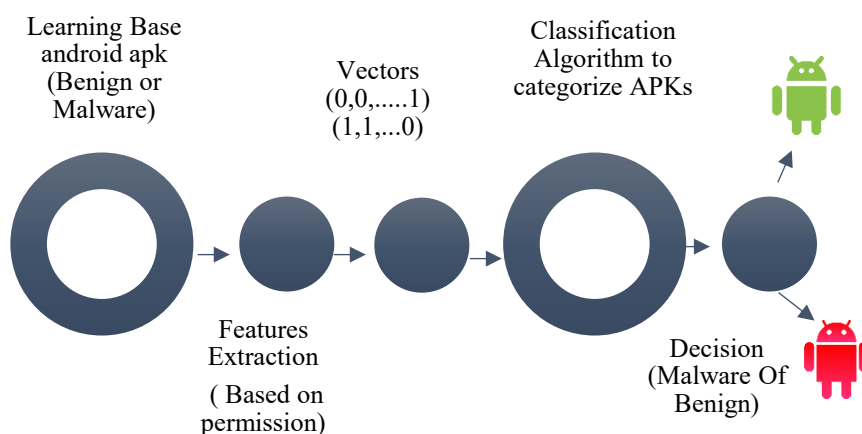


Fig. 2. the process of the android malware detection.

Our analysis experience is based on three Android malware datasets. The first is the Malware/Benign Permissions Android dataset, which was created from prior machine learning efforts and current engineering Android security [1,2]. The data was collected through binary vectorization, where every app was mapped to a set of permissions. The used classification method was “malware type” 1 and benign 0 [1,2]. This created a total of 398 binary applications, of which 199 were malware and the other 199 were benign, with a total of 331 features. The final column specified the application type, which was either malicious or benign [1,2].

The second dataset is the Drebin dataset, which is hosted on Kaggle by Likhitha Durusoju. This dataset is a static Android malware dataset derived from the widely used Drebin benchmark. The dataset includes benign and malicious Android applications represented as feature vectors extracted from the app code and manifest files. The feature representation contains permissions, API calls, hardware components, application components, and network addresses [03]. Each column represents a specific feature extracted from the Android applications, which include Permissions (ACCESS_FINE_LOCATION), API Calls, App Components (services, activities), Hardware Components (GPS, camera), Network Addresses (URLs, IPs), and the Label column, which is a binary label that is either benign (0) or malicious (1) [03]. It contains 15,036 total apps separated into 5,560 malware and 9,476 benign apps and features a total of 215 static attributes (binary/vector features) extracted per app, so there is a total of 216 columns with the label column (215 feature columns + 1 label column) [03].

The third dataset is the Droid det malware dataset [4], which is hosted by Akbar, F and Available online in the GitHub platform. The dataset contains a total of 10,000 APK samples, with 5,000 benign APKs collected from various Play Store application categories and 5,000 malware APKs [4]. The samples are used for cross-validation experiments and training data in classification models to evaluate Android malware detection methodologies. The dataset is available in a format suitable for classification models, including APK features and labels indicating whether each APK is benign or malicious [4].

The three datasets contain numerical values, which are easy to use without preparation, we only load the dataset into our program and used it for the detection of malware applications.

For the first dataset, a set of data mining algorithms was used to classify the Android apps utilizing the library Scikit-Learn [15]. We had done this to evaluate our model with other algorithms because we had found Just one work that used the same dataset as us. For the second dataset we are going to compare the results obtained by our model with other works done on the same dataset. Moreover, we combined by using a blend of two algorithms in a hybrid

classifier for individual apps, a concept that will be elaborated on later in this paper. The same thing for the third dataset.

We had also obtained the test and training set by using the k-fold cross-validation splits to evaluate the classifier (5-fold). Following this step, we began the classification task, and the Android applications based on permissions were automatically classified as benign or malicious.

Bio-inspiration methods had a big weight on the field of research because of their good results in different processes [36,37]. The researchers apply this discipline and get their concepts based on the imitation of nature and the different behaviors of the living organisms found on it [36,37]. From this discipline we find swarm intelligence, which can be considered as a subset of it. Swarm intelligence gets its concept from the collective behavioral patterns of organisms like ants, birds, or schools of fish...etc [38,39,40,41,42].

Particle Swarm Optimization (PSO) is an algorithm inspired by the natural behavior of birds or schools of fish, a set of particles explores the research space, and their position changes according to their past experiences and the collective information of the swarm for optimal or suboptimal solutions of intricate problems [38,39,40,41,42]. These methods are applied to optimization problems. PSO is not as complex as the other algorithms of optimization, its operation doesn't contain a complex equation. Its needs only a few parameters to be set to function, this algorithm avoids getting stuck in local optima, which is one of the biggest strengths of the PSO [38,39,40,41,42]. He is based on strategy, which gets it to its global search implementation, the strategy of having both balance exploration (searching new areas) and exploitation (refining known good solutions). The system has sufficient diversity to guarantee that if the personal best and the global best are worked upon together, the swarm doesn't converge too soon on a suboptimal [38,39,40,41,42].

LSTM is a version of a recurrent neural network that is useful for sequential data analysis and feature extraction, its efficacy is highly reliant on the appropriate selection of hyperparameters like the count of LSTM units, dropout percentage, learning rate, and batch size [43,44,45,46]. During the training phase, LSTMs are capable of learning long-term dependencies in sequential data, especially for time series predictions, speech recognition, language modeling, and even translation tasks [43,44,45,46]. LSTM units are composed of special cells that consist of three primary gates that control the information ingested into the memory: Forget Gate: It decides what information from the cell state should be erased. Input Gate: It chooses what information from the input should be changed in the cell state. Output Gate: It sets what the next hidden state should be. These gates help the LSTM network to keep track of key information across long sequences, improving on the performance of traditional RNNs [43,44,45,46]. As important features for LSTM, we found, firstly, the cell state, which serves as a long-term memory pathway, allowing important information in the sequence to flow through the network with minimal modification. It is affected by the gates in a positive or negative mode. Secondly, we found the hidden state, which captures the result of the task performed by the LSTM unit at that instance of time [43,44,45,46]. LSTM networks are increasingly popular for solving sequence prediction tasks where knowledge retention for long periods is of considerable importance [43,44,45,46].

METHODS

LSTM is useful and a very important algorithm in sequential data processing and feature extraction, its effectiveness is limited by the selection of hyperparameters (for example, the number of LSTM units, dropout rate, learning rate, and batch size) that must be optimally defined [43,44,45,46].

The PSO is an optimization algorithm inspired by the natural behavior of swarm organisms, which can be effectively employed to automatically search for the optimal set of hyperparameters [38,39,40,41,42].

LSTM and PSO are well-established techniques, the novelty of this research is specified in the integration of pso-optimized LSTM architecture for Android malware detection using permission-based features, a combination that has been rarely explored.

Most existing Android malware detection models focus on heavy or complex features such as API calls, network behavior, or system calls. This research has touched all these features and also the permission features when we talk about the three datasets.

The LSTM model had been applied to multiple problems of security tasks, getting its hyperparameters automatically with PSO for the Android malware detection based on features permissions or apps features is a novel approach.

With the combination of PSO and LSTM, we seek to increase the performance of this model according to traditional models that use manually tuned LSTMs or non-sequential models, our purpose is to improve the accuracy, precision, recall, and F1 score of Android malware detection through the optimization of the LSTM architecture (hyperparameters). Manual tuning of LSTM hyperparameters is time-consuming and often suboptimal. By using PSO in the selecting process, the model gains an automated and adaptive mechanism for configuration, enhancing scalability and reducing reliance on expert knowledge.

This research aims to develop a new malware detection system based on permissions and app features in Android applications by employing a hybrid model that uses Particle Swarm Optimization (PSO) and Long Short-Term Memory networks (LSTM).

The proposed model is thoroughly evaluated on real-world Android datasets, the three used datasets are mentioned before [1,2,3,4]. The model classifies the application as either malware or benign.

The model has its own set of hyperparameters that should be configured as the LSTM units (the number of neurons in each layer), dropout rate (controls overfitting), batch size (number of samples processed in a given iteration), and learning rate (delta for the change of weight on each node). In this work we used the PSO to optimize hyperparameters of the LSTM. The PSO sets up a swarm of particles corresponding to LSTM hyperparameters, for every particle (solution), it trains the LSTM model and estimates validation accuracy. The best-known position and the global best are used to modify the position. For a given number of iterations, the process continues to optimize (for example, five iterations with ten particles), from all particles, the most optimized set of hyperparameters is appended to the best particle. After that we train the LSTM model with the optimized hyperparameters, the optimal parameters introduced by the PSO are used for the final training of the LSTM model.

The advantages of this hybrid model are that there is no need for manual work, the LSTM hyperparameters are generated automatically from the PSO optimizer, there is a faster convergence from the PSO to find the best hyperparameters, and it is done efficiently. The PSO-optimized LSTM outperforms traditional models to get better accuracy. Also, this hybrid model can be generated to set it for different other work and other datasets and select the best parameters for itself.

1 PSO Optimizer

Particle swarm optimization (PSO) is an adoption method of social movements of birds and fishes. It targets optimization problems while replicating a swarm of particles looping and trying to find the most ideal solution [38,39,40,41,42].

For the entire system simulation, all particles represent possible solutions, and their movements are guided by their past achievements (personal best) and neighboring particles (global best). The group effort of the swarm led to known factors being added together and new ones being formed, leading towards an optimal or close to optimal solution [38,39,40,41,42]. The core principle of PSO can be briefly described in the following points:

- **Initialization:** Set a specific quantity of particles to random locations and movements within the solution space.
- **Fitness Evaluation:** Evaluate every single particle using a fit function that scores how valuable the provided solutions are.
- **Update Rules:** Move based on the two aspects according to the evaluation:
 - Personal best (the best position encountered).
 - Global best (the optimal solution established by the entire swarm).

Doing these procedures claims to move one step closer to finding a more refined solution to their problem [38,39,40,41,42].

2 LSTM Hyperparameter

LSTM has a set of hyperparameters, we can mention some of them that we are interested in as follows:

Number of Hidden Units (Neurons): This refers to the quantity of components included in the LSTM's hidden layer. It defines how deeply the network can learn patterns, and the information processed increases computational complexity. With more neurons, there is more information capture, but at the same time, there is increased computational difficulty [47,48].

Learning Rate: Learning rate defines the magnitude of the step when performing the gradient descent procedure. Larger learning rates increase the step taken, which improves convergence speed. Smaller learning rates improve precision during steps taken, at the same time being slower when it comes to achieving the desired result [47,48].

Batch Size: Batch size describes the number of training data samples that are processed simultaneously during one forward and one backward pass. Smaller sizes may add instability to the model, while larger sizes slow things down with increased algorithm stability and higher memory requirements [47,48].

Sequence Length: Sequence length refers to the number of steps (or time tokens) the model analyzes in order to make a single prediction. Any process that includes forecasting, as well as speech recognition, will require the entire sequence of data to be passed through the model, therefore, it is critical to establish sequence length prior to training a model [47,48].

Number of Layers: The term describes how deep an LSTM network can go. Even if there are more layers, you have a more complex network in terms of depth. A higher level of complexity carries with it a higher level of computation, and as a result, previously unfit systems may become overfitted [47,48].

Dropout Rate: This term refers to many regularization techniques, including dropping a certain percentage of neurons while training a deep learning model in order to reduce overfitting. Setting the drop rate, ranging from zero to one, determines how many neurons will be dropped [47,48].

Activation Functions: LSTM networks make use of both the sigmoid and tanh activation functions in their gates. The sigmoid function is applied to the input, output, and forget gates. The tanh is used to scale and regulate the values within the cell state, ensuring that the data remains within a manageable range. Functions in each gate determine the extent to which data moves through the system [47,48].

Binary Cross-Entropy or Log Loss is one of the commonly used loss functions in binary classification i.e, the objective is to categorize into two classes, for example (class 0 vs class 1). It is particularly helpful when there is the output of the model in the form of a multi-valued probability, bracketing zero (0) to one (1) where it captures the possibility of the instance being in the positive class (class 1). The loss function calculates how far apart the actual labels are from the predicted probabilities [49,50].

3 Hybrid Algorithm

-Hyperparameter Optimization with PSO:

• **Step one:** We're going to initialize a swarm, for each particle stands for a set of hyperparameters of an LSTM, which is trained for e.g.: (learning rate, number of neurons, dropout rate, etc.).

A particle's position is represented as a vector of values corresponding to hyperparameters, which include, for example, the vector Z.

$$Z = (0.002, 32, 0.2, 16, 200)$$

- Learning rate = 0.002
- LSTM units = 32
- Dropout = 0.2

- Batch size = 16
- Epochs = 200

Each particle possesses a **position**, which is defined by a set of hyperparameters, and a **velocity**, which refers to a step size for movement in the search space. The initial position and velocity are set to random values.

• **Step Two:** For each particle, we are going to train an LSTM and check the model accuracy and performance with the designed schemes with hyperparameters.

The **fitness** of each particle is computed by fitting an LSTM model corresponding to its hyperparameters. The model undergoes fitting on the training dataset, and validation steps are performed on an independent dataset.

- The accuracy is defined as **Fitness = 1 – Accuracy**.

Minimizing this function, which enhances classification accuracy, helps in achieving the objective.

• **Step Three:** Update Particles. The particles are moved according to personal best and global best solutions.

Each **particle** keeps track of the best solution it has encountered at some point, which is labeled as pBest. The best solution of all participants is identified as gBest.

Change in **Position and Velocity**: each particle will have its velocity updated with the following information:

Velocity Update: The velocity of each particle is updated using the following equation:

$$\mathbf{v}_i^{+1} = \mathbf{w} \cdot \mathbf{v}_i + \mathbf{c}_1 \cdot \mathbf{r}_1 \cdot (\mathbf{pBest}_i - \mathbf{x}_i) + \mathbf{c}_2 \cdot \mathbf{r}_2 \cdot (\mathbf{gBest} - \mathbf{x}_i) \quad (1)$$

Where:

- **w**: Inertia weight – Determines balance between promising and chasing revealing information.
- **c₁, c₂**: Known as acceleration coefficients. Usually ranging between 1.5 and 2.5.
- **r₁, r₂**: Known as random numbers. Values can be accepted as ranging from 0 to 1.
- **x_i**: Denote the position of the i particle.
- **v_i**: Denotes velocity of the i particle.

Position Update: After the velocity adjustment of the particle, the new position of the particle is computed as follows:

$$\mathbf{x}_i^{+1} = \mathbf{x}_i + \mathbf{v}_i^{+1} \quad (2)$$

This new position is associated with a new set of hyperparameters corresponding to particle i.

• **Step Four:** Repeat Until Convergence, the PSO keeps looping till he gets the best hyperparameters.

• **Step Five:** Train the final LSTM model. This final step in the hybrid approach will train the LSTM using the best hyperparameters that are found by the PSO.

The algorithm stops at a specified number of cycles, or when performance enhancements become nominal over time, the final best hyperparameter is chosen from the best solution found, referred to as gBest.

4 The Process steps (see Error! Reference source not found.).

- 1- The input dataset comprises features of Android apps.

- 2- The PSO optimizer is used for hyperparameter selection on the LSTM model.
- 3- LSTM training is conducted on every particle solution.
- 4- The hyperparameters that yield the most accurate results are selected.
- 5- These selected parameters are then used to train the final LSTM model.
- 6- The optimized model is deployed for classifying malware or benign applications.

• **In this study**, the final model with the best hyperparameters is evaluated with accuracy, precision, recall, f1-score, AUC, entropy, and the ROC curve.

• The test and the training data from the used data set are gutted by the cross-validation method, as we had mentioned before.

5 Algorithm

Input:

- Dataset with features and labels
- PSO parameters (swarm size, max iterations, inertia weight, acceleration coefficients)
- Hyperparameter search ranges (learning rate, LSTM units, dropout rate, batch size, epochs)

Process

1. Initialize a swarm of particles, each representing a set of random hyperparameters.
2. For each particle:
 - Train an LSTM model using the particle's hyperparameters.
 - Evaluate the model accuracy on validation data.
 - Record the particle's best position (hyperparameters) based on accuracy.
3. Identify the global best position (the hyperparameters that gave the highest accuracy across all particles).
4. Update velocity and position of each particle based on its personal best and the global best.
5. Repeat steps 2–4 until the maximum number of iterations is reached or convergence is achieved.
6. Train the final LSTM model using the best hyperparameters found.
7. Output the optimized hyperparameters and the trained model.

Output:

- Best hyperparameters
- Trained LSTM model with optimized parameters

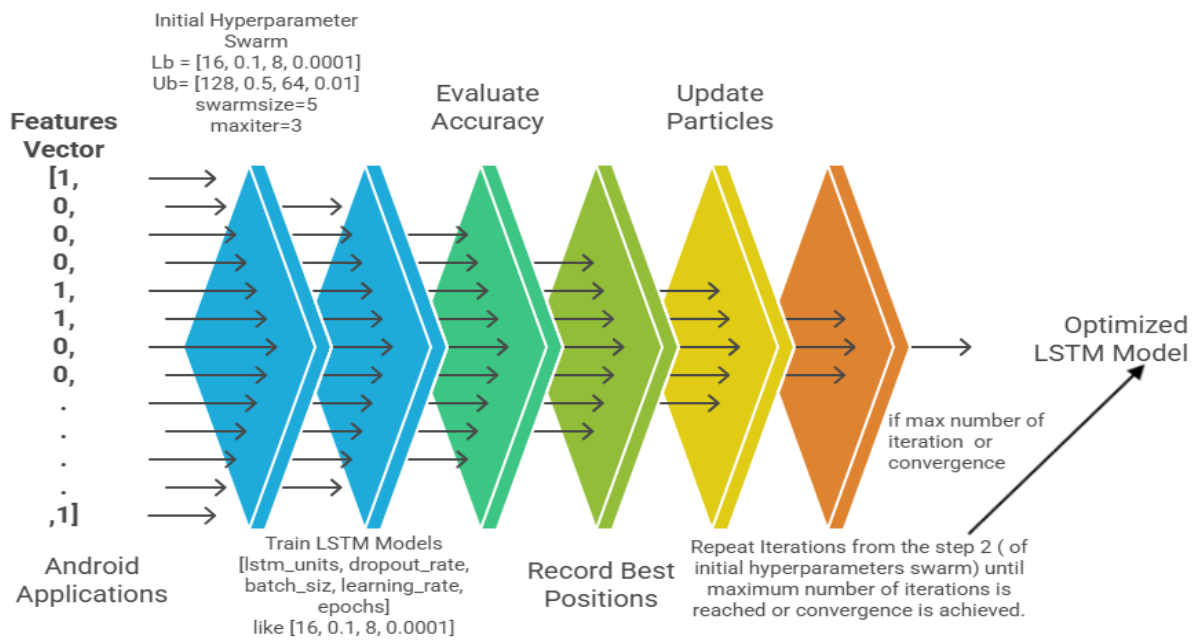


Fig. 3. Optimization process of the hybrid algorithm for Android malware detection.

RESULTS

This work was performed by applying the LSTM RNN using the TensorFlow-Keras library [5] within the Python environment and the PSO optimizer using the pyswarm library [6].

The other algorithms described in the first experience were also executed using Scikit-Learn, which was done in Python. Scikit-Learn provides simple access to a multitude of other classification algorithms. We incorporated SVM, Naive Bayes, Gaussian Naive Bayes, KNN with k equaling three and five, Decision Tree, Random Forest Classifier, Extra Tree Classifier, Gradient Boosting Classifier, and ADA Boost [15].

1 First experience (using the Malware/Benign Permissions Android dataset [1,2])

- Parameters of the algorithms used:

In the next two tables, we are going to mention the hyperparameters of the LSTM, which is selected manually in Table 2, and we had also mentioned the initial hyperparameters for the PSO in Table 3.

Table 2. The hyperparameters selected manually for the LSTM.

Parameters		LSTM	
	VALUE	DESCRIPTION	
lstm_units	64	Number of units (neurons) in each LSTM layer [5]	
dropout_rate	0.3	Fraction of the LSTM outputs dropped to prevent overfitting [5]	
batch_size	32	Number of training samples used per gradient update [5]	

learning_rate	0.001	Step size used by the optimizer to update weights [5]
epochs	10, 20, 50	Total number of training iterations over the entire dataset [5]

Table 3. The initial parameters of the PSO OPTIMIZER.

Parameters		PSO
	VALUE	DESCRIPTION
train_lstm	Function to optimize	Your custom function that returns -accuracy of the model
Lb	[16, 0.1, 8, 0.0001]	Lower bounds for parameters (LSTM units, dropout, batch size, learning rate)
Ub	[128, 0.5, 64, 0.01]	Upper bounds for parameters
swarmsize	3	Number of particles (candidate solutions) in the swarm
Maxiter	2	Maximum number of iterations (generations) to run PSO

So, the PSO optimizer is exploring and searching in this space:

- LSTM units : [16, 128]
- Dropout : [0.1, 0.5]
- Batch size : [8, 64]
- Learning rate : [0.0001, 0.01]

Table 4 below contains the results obtained by all the experimentation done on the Malware/Benign Permissions Android dataset [1,2]. We had begun with the manually selected hyperparameters for the LSTM model, after that, we had applied the hybrid model with different parameters of the optimizer.

Table 4. The results obtained by the LSTM solo model and our hybrid model on the Malware/Benign Permissions Android dataset.

Epoch	Measure	Malware/Benign Permissions Android Dataset
		PSO-LSTM
	LSTM SOLO	(Swarm Size 3, MaxIter 2)

10	Accuracy	0.7337	0.9125
	Precision	0.7804	0.9134
	Recall	0.7219	0.9142
	F1 Score	0.7086	0.9125
	AUC	0.7688	0.9254
	Entropy	0.3602	0.1159
20	Accuracy	0.6049	0.9250
	Precision	0.7645	0.9248
	Recall	0.6088	0.9248
	F1 Score	0.5047	0.9248
	AUC	0.5136	0.8991
	Entropy	0.3883	0.1077
50	Accuracy	0.5374	0.9241
	Precision	0.7171	0.9297
	Recall	0.5591	0.9220
	F1 Score	0.4386	0.9234
	AUC	0.3716	0.9641
	Entropy	0.4802	0.0965

As shown in Table 4, comparing the results obtained by the two models (the LSTM solo model and the PSO-LSTM model applying on the Malware/Benign Permissions Android Dataset), as we show in the table, for the PSO in the hybrid model we had used the parameters (Swarmsize=3 and MaxIter=2). In the first experience with 10 epochs, the hybrid model achieved an accuracy of 91.25% and an F1-score of 91.25%, which demonstrate a balanced performance with modest results. For the AUC value we had obtained a 0.9254, indicating a reasonable ability to distinguish between classes. The model also had an entropy of 0.1159, which reflects some uncertainty in predictions. The results of the hybrid model were optimized and enhanced according to the LSTM solo model.

In the second experience, with 20 epochs, comparing the results obtained by the two models, there was an improvement in the hybrid model, with accuracy reaching 92.50% and the F1-score improving to 92.48%, indicating that the model is continuing to learn effectively and that the PSO had an effect on the model with the selection of the hyperparameters. However, the AUC dropped slightly to 0.8991, suggesting that while the model's overall accuracy increased, its ability to separate classes became marginally less robust. The entropy decreased to 0.1077, showing a small loss of information and that the model is becoming more confident in its predictions.

In the last experience on the table with 50 epochs, the performance of the hybrid model had given an accuracy with 92.41% and F1-score at 92.34%, showing a minimal enhancing on the results compared to epoch 20. However, for the AUC, it has been increased to 0.9641, indicating a notable boost in the model's class separation ability. The entropy has decreased to 0.0965, demonstrating that the model's predictions were becoming increasingly confident and showing a small less of information to the other experiences. In general, the results obtained had been improved when using the hybrid model compared to the LSTM solo model in all epochs.

In conclusion, the PSO-LSTM hybrid model applied to the Malware/Benign Permissions Android Dataset shows an improvement in terms of accuracy, F1-score, and AUC. The decreasing of the value of entropy indicates that the model

is becoming more confident with each increase in the number of epochs. These results highlight the effectiveness of PSO for hyperparameter tuning, providing consistent improvements and efficient model convergence compared to manual approaches. The model's ability to adapt to different problems also makes it highly practical for real-world applications, with good scalability. The next figure (Fig.4) refers to the ROC curves of the two models' performance. Which are used to evaluate classification models with the true positive rate plotted against the false positive rate.

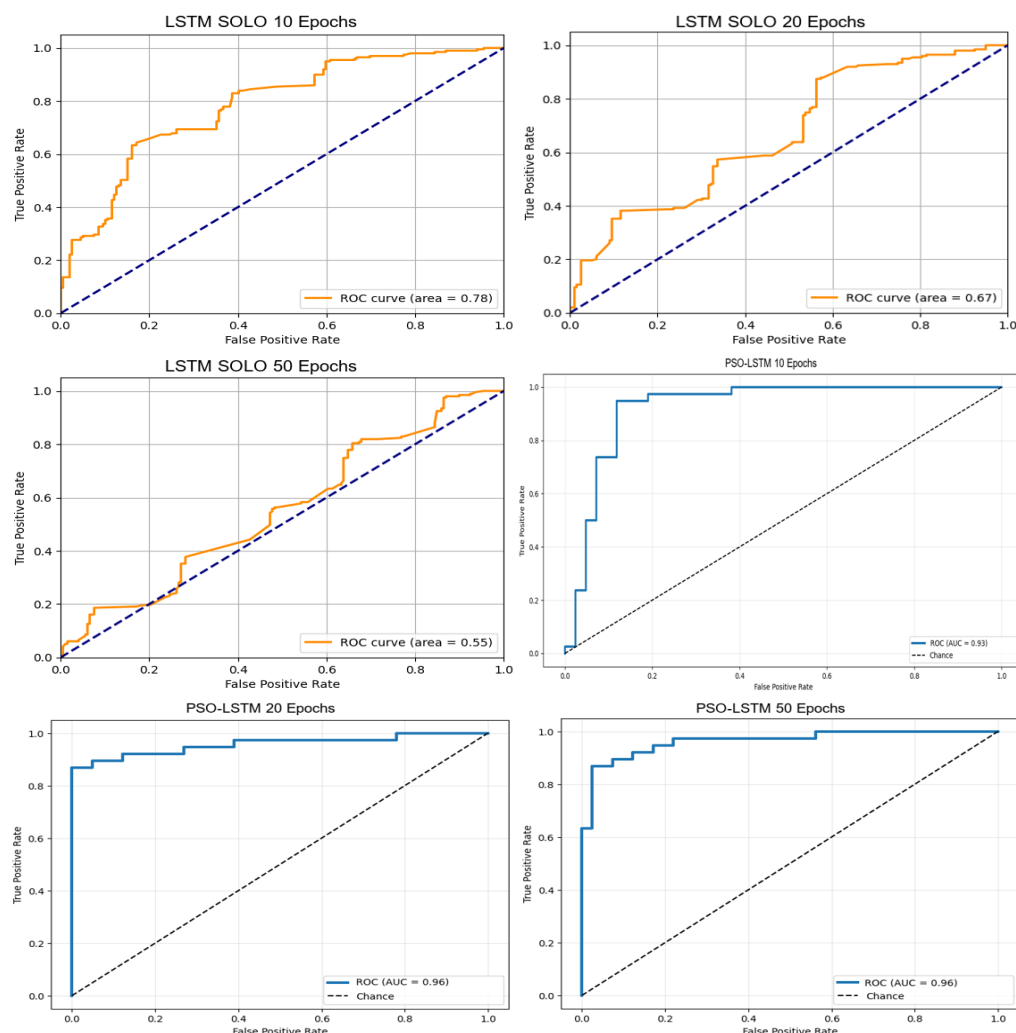


Fig. 4. Comparison of LSTM and PSO-LSTM Performance (ROC curves) with Malware/Benign Permissions Android Dataset.

Table 5 below contains the parameters of the algorithms executed using Scikit-Learn, which was done in Python. We incorporated SVM, Naive Bayes, Gaussian Naive Bayes, KNN with k equaling three and five, Decision Tree, Random Forest Classifier, Extra Tree Classifier, Gradient Boosting Classifier, and ADA Boost [15].

Table 5. Parameter of different algorithms used for comparison (5-fold cross validation)

Model Name		Algorithms
	Class	Parameters
Random Forest	RandomForestClassifier	n_estimators=100, random_state=42
Decision Tree	DecisionTreeClassifier	random_state=42

Support Vector Machine (SVM)	SVC	kernel='linear', probability=True, random_state=42
K-Nearest Neighbors	KNeighborsClassifier	n_neighbors=5
Naive Bayes	GaussianNB	/
Logistic Regression	LogisticRegression	max_iter=1000, random_state=42

After a number of tests and experiences, we selected the best accuracy obtained by the hybrid algorithm. The selected results are compared and evaluated with other algorithms of data mining and other works for the Malware/Benign Permissions Android Dataset.

Table 6. comparison of the results with other work already done.

Algorithms	Measures
	Accuracy
Random Forest	0.9347
Decision Tree	0.9093
SVM	0.9397
KNN	0.9221
Naive Bayes	0.7634
Logistic Regression	0.9423
LSTM (RNN)	0.7337
PSO-LSTM (RNN)	0.9250
Random Forest [52]	0.9750

As shown in Table 6, our optimized hybrid model (PSO-LSTM) achieved an accuracy of 92.50%, outperforming the standard LSTM model (73.37%) and several of the classical machine learning models, such as Naive Bayes (76.34%) and KNN (92.21%). However, it slightly underperformed compared to logistic regression (94.23%), SVM (93.97%), and random forest (93.47%), and more significantly against the random forest model from JIS EURASIP [52], which achieved 97.50% accuracy using only 48 static permission features. The JIS EURASIP model is based on feature selection, which has enhanced performance, the model selects the most informative permissions, ensuring minimal noise and redundancy. In contrast, in our PSO-LSTM model, we had used all features (hundreds of features) without prior selection, which may include irrelevant or less impactful data, potentially reducing its effectiveness despite the use of Particle Swarm Optimization (PSO) for hyperparameter tuning. Furthermore, random forest, is a robust ensemble-based algorithm, it is particularly well-suited for structured, tabular data like static permissions. Its simplicity, low training complexity, and high interpretability make it a strong model, especially for datasets lacking temporal characteristics. On the other hand, LSTM models are good for sequential or time-series data, which is not the case when they are applied to static features like Android permissions, which do not follow a meaningful temporal order, their sequential learning advantage in this type of data may not translate into improved performance. Without talking about its optimization capabilities, the PSO-LSTM model introduces additional architectural complexity without a guaranteed performance gain in static settings. In summary, while PSO-LSTM shows competitive performance among several models, this combination of feature selection, model simplicity, and dataset structure

gives random forest particularly with the optimized model (JIS EURASIP) [52], a clear advantage in permission-based Android malware detection.

- We compared just with accuracy because it was the only measure available.

2 The second experience (using the Drebin dataset [3])

- Parameters of the algorithms used

In the second experience, we are going to use the LSTM SOLO model with the hyperparameters selected manually mentioned before in Table 2 applied on the Drebin dataset. For the hybrid model PSO-LSTM, we had used also the same initial hyperparameters for the PSO mentioned in Table 3, and applied the hybrid model on the Drebin dataset.

Table 7 below contains the results obtained by all the experimentation done on the Drebin Android dataset [3]. We had begun with the LSTM model, after that, we had applied the hybrid model with different parameters of the optimizer.

Table 7. The results obtained by the LSTM solo model and our hybrid model on Drebin Android dataset.

Epoch	Measure	Drebin Dataset	
		LSTM SOLO	PSO-LSTM (Swarm Size 3, MaxIter 2)
10	Accuracy	0.9132	0.9297
	Precision	0.9059	0.9228
	Recall	0.7929	0.8775
	F1 Score	0.8108	0.8973
	AUC	0.7650	0.9785
	Entropy	0.1431	0.1031
20	Accuracy	0.9352	0.9538
	Precision	0.9199	0.9439
	Recall	0.8444	0.9265
	F1 Score	0.8482	0.9348
	AUC	0.7568	0.9880
	Entropy	0.1205	0.0802
50	Accuracy	0.9498	0.9607
	Precision	0.9350	0.9450
	Recall	0.8676	0.9421
	F1 Score	0.8679	0.9436
	AUC	0.7794	0.9899
	Entropy	0.0972	0.0770

As shown in Table 7, comparing the results obtained by the two models (the LSTM SOLO model and the PSO-LSTM hybrid model) applied to the Drebin Android Dataset, we used the following PSO parameters in the hybrid model

(SwarmSize = 3 and MaxIter = 2). In the first experiment with 10 epochs, the PSO-LSTM hybrid model achieved an accuracy of 92.97%, and an F1-score of 89.73%, demonstrating balanced performance with modest improvements over the LSTM SOLO model. For the AUC, the model reached a high value of 0.9785, indicating a strong ability to distinguish between classes. The entropy was 0.1031, reflecting a moderate level of uncertainty in predictions. Overall, the hybrid model outperformed the LSTM SOLO in all metrics.

In the second experiment with 20 epochs, the hybrid model showed further improvement, with accuracy increasing to 95.38% and the F1-score rising to 93.48%, indicating effective learning and demonstrating the impact of PSO in optimizing LSTM hyperparameters. However, the AUC increased to 0.9880 compared to the previous epoch. The entropy decreased to 0.0802, showing reduced uncertainty and improved prediction confidence.

In the final experiment with 50 epochs, the hybrid model achieved the best performance an accuracy with 96.07% and an F1-score of 94.36%, representing a slight improvement over the 20-epoch performance. Notably, the AUC increased to 0.9899, highlighting a significant enhancement in the model's ability to separate classes. The entropy further decreased to 0.0770, indicating growing model confidence and more deterministic outputs.

In general, the results obtained using the PSO-LSTM hybrid model showed consistent improvements across all epochs when compared to the LSTM SOLO model. This was evident in terms of accuracy, F1-score, AUC, and entropy values. The PSO-LSTM hybrid model, applied to the Drebin Android Dataset, demonstrated clear improvements in performance metrics, including accuracy, F1-score, and AUC comparing to the first dataset (Malware/Benign Permissions Android Dataset), which is explained in general that the Drebin dataset contains a more comprehensive set of features and a various types of malicious and benign Android apps, providing richer data for training machine learning models. The decreasing entropy values across training epochs suggest that the model becomes increasingly confident in its predictions as training progresses. These results highlight the effectiveness of PSO for hyperparameter tuning, enabling efficient model convergence and consistent performance gains over manual tuning. Furthermore, the model's adaptability makes it suitable for real-world applications, providing a balance of scalability, accuracy, and robustness.

The next figure (Fig. 5) presents the ROC curves for both models, visualizing their performance by plotting the true positive rate against the false positive rate.

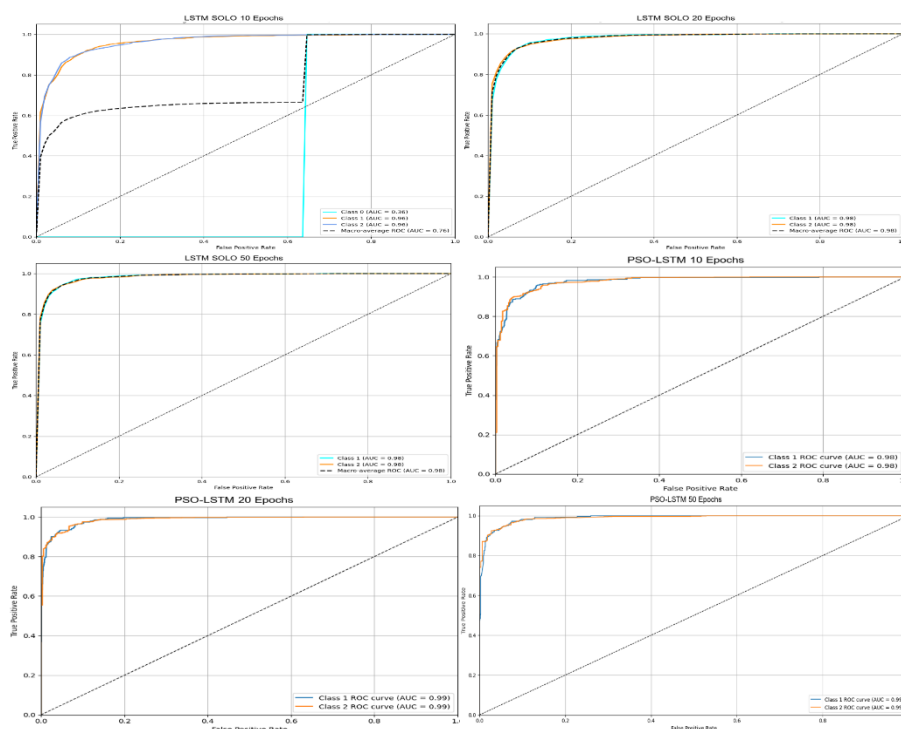


Fig. 5. Comparison of LSTM and PSO-LSTM Performance (ROC curves) with Drebin dataset.

Table 8. comparison of the results with other work already done.

Algorithms	Measures
	Accuracy
Decision Tree [53]	90.2%
Naïve Bayes [53]	89.3%
SVM [53]	94.2%
Neural Networks [53]	93.1%
Hybrid Model [53]	94.2%
GCN [54]	81.0%
GCN GST + EFD [54]	88.8%
GCN-JK [54]	89.7%
GIN [54]	90.0%
GIN-JK [54]	90.0%
GraphSAGE [54]	88.1%
GraphSAGE GST + EFD [54]	89.2%
GraphSAGE-JK [54]	94.4%
HGNN [54]	87.6%
HyperGCN [54]	79.3%
HGNNP [54]	91.1%
UniGIN [54]	89.0%
GraphGPS [54]	90.8%
GraphGPS GST + EFD [54]	92.5%
Expformer [54]	94.0%
GraphCL [54]	87.6%
Inferential SIR-GN [54]	92.0%
DenseNet121 (CNN)[54]	86.3%
GCN (Reproduction) [54]	80.6%

GIN (Reproduction) [54]	89.1%
DenseNet121 + GCN (CNN-GNN Ensemble) [54]	88.9%
DenseNet121 + GIN (CNN-GNN Ensemble) [54]	90.6%
DenseNet121-GIN [54]	94.2%
LSTM SOLO	94.98%
PSO-LSTM	96.07%

As shown in Table 8, Our PSO-LSTM hybrid model achieved the highest accuracy of 96.07%, outperforming other models, including LSTM SOLO (94.98%) and traditional machine learning algorithms [53], such as SVM (94.2%) and Neural Networks (93.1%). The PSO-LSTM hybrid model, which is a combination of Particle Swarm Optimization (PSO) and the LSTM Model. The pso swarm intelligence are used for the hyperparameter optimization, by an automatic selection of this hyperparameters contributing to its superior performance. The PSO-LSTM model's complexity may not always translate to significant improvements in all types of data or settings, especially when compared to simpler models like SVM and Neural Networks, which performed competitively with 94.2% accuracy. While GraphSAGE-JK (94.4%) and DenseNet121-GIN (94.2%) [54] performed well within the GNN and hybrid architecture space, they still slightly underperformed compared to the PSO-LSTM model, which leverages sequential learning capabilities of LSTM alongside PSO's search mechanism. GraphSAGE-JK [54], with its Jumping Knowledge architecture, showed strong performance, particularly in graph-based learning tasks, but its performance is still lower than the PSO-LSTM.

Decision Tree (90.2%) and Naïve Bayes (89.3%) [53] both displayed reasonable accuracy for classical machine learning algorithms but were outperformed by all LSTM, SVM, and hybrid models. When considering Graph-based models, such as GCN (81%) and GraphGPS (90.8%) [54], it's evident that these models excel in graph-related tasks, but they were less effective in this malware detection context when compared to the top performing LSTM models. While GraphGPS GST + EFD (92.5%) [54] showed significant enhancement using graph transformer-based learning, its performance was still lower than PSO-LSTM. Furthermore, Expformer (94.0%) and HyperGCN (79.3%) [54] represent alternative transformer-based and hypergraph models, but their performance is somewhat inconsistent in the context of permission-based Android malware detection. The PSO-LSTM model benefits from LSTM's ability to process sequential data, even though the dataset here is not inherently sequential, indicating that LSTM-based models can still offer competitive performance for static datasets when optimized effectively.

In conclusion, while the PSO-LSTM stands out with its 96.07% accuracy, its performance is competitive but still depends on the specific nature of the dataset. For static data, simpler models such as SVM or Hybrid Models with Neural Networks perform comparably. However, for sequence-based tasks or where hyperparameter optimization (using PSO) can be leveraged effectively, PSO-LSTM presents an appealing choice, showcasing its strength in handling high-dimensional features.

- We compared just with accuracy because it was the only measure available.

3 The third experience (using the Droid det malware dataset [4])

- Parameters of the algorithms used

In the third and the last experience, we are going to use also the LSTM SOLO model with the hyperparameters mentioned before in Table 2 applied on the Droid det malware dataset. For the hybrid model PSO-LSTM, we gone use also the same initial hyperparameters for the PSO mentioned in Table 3, and applied the hybrid model on this new Droid det malware dataset.

Table 9 below contains the results obtained by all the experimentation done on the Droid det malware dataset [4]. We had begun with the LSTM model, after that, we had applied the hybrid model with different parameters of the optimizer.

Table 9. The results obtained by the LSTM solo model and our hybrid model on Droid det malware dataset.

Epoch	Measure	Droid det malware Dataset	
		LSTM SOLO	PSO-LSTM (Swarm Size 3, MaxIter 2)
10	Accuracy	0.9022	0.9125
	Precision	0.9023	0.9125
	Recall	0.9021	0.9125
	F1 Score	0.9022	0.9125
	AUC	0.9416	0.9558
	Entropy	0.1478	0.1319
20	Accuracy	0.9031	0.9165
	Precision	0.9032	0.9168
	Recall	0.9030	0.9165
	F1 Score	0.9031	0.9165
	AUC	0.9449	0.9608
	Entropy	0.1466	0.1257
50	Accuracy	0.9053	0.9155
	Precision	0.9055	0.9156
	Recall	0.9052	0.9155
	F1 Score	0.9052	0.9155
	AUC	0.9547	0.9565
	Entropy	0.1436	0.1279

As shown in Table 9, comparing the results obtained by the two models (the LSTM SOLO model and the PSO-LSTM hybrid model) applied to the Droid det malware Dataset, we used the following PSO parameters in the hybrid model (SwarmSize = 3 and MaxIter = 2). In the first experiment with 10 epochs, the PSO-LSTM hybrid model achieved an

accuracy of 91.25%, and an F1-score of 91.25%, demonstrating balanced performance with modest improvements over the LSTM SOLO model. For the AUC, the model reached a high value of 0.9558, indicating a strong ability to distinguish between classes. The entropy was 0.1319, reflecting a moderate level of uncertainty in predictions. Overall, the hybrid model outperformed the LSTM SOLO in all metrics.

In the second experiment with 20 epochs, the hybrid model showed further improvement, with accuracy increasing to 91.65% and the F1-score rising to 91.65%, indicating effective learning and demonstrating the impact of PSO in optimizing LSTM hyperparameters. However, the AUC increased to 0.9608 compared to the previous epoch. The entropy decreased to 0.1257, showing reduced uncertainty and improved prediction confidence.

In the final experiment with 50 epochs, the hybrid model achieved an accuracy of 91.55% and an F1-score of 91.55%. Notably, the AUC value obtained was 0.9565, highlighting a significant enhancement in the model's ability to separate classes. The entropy has a value of 0.1279, indicating growing model confidence and more deterministic outputs.

In general, the results obtained using the PSO-LSTM hybrid model showed consistent improvements across all epochs when compared to the LSTM SOLO model. This was evident in terms of accuracy, F1-score, AUC, and entropy values. The PSO-LSTM hybrid model, applied to the Droid det malware Dataset. These results highlight the effectiveness of PSO for hyperparameter tuning, enabling efficient model convergence and consistent performance gains over manual tuning. Furthermore, the model's adaptability makes it suitable for real-world applications, providing a balance of scalability, accuracy, and robustness.

The next figure (Fig.6) presents the ROC curves for both models, visualizing their performance by plotting the true positive rate against the false positive rate.

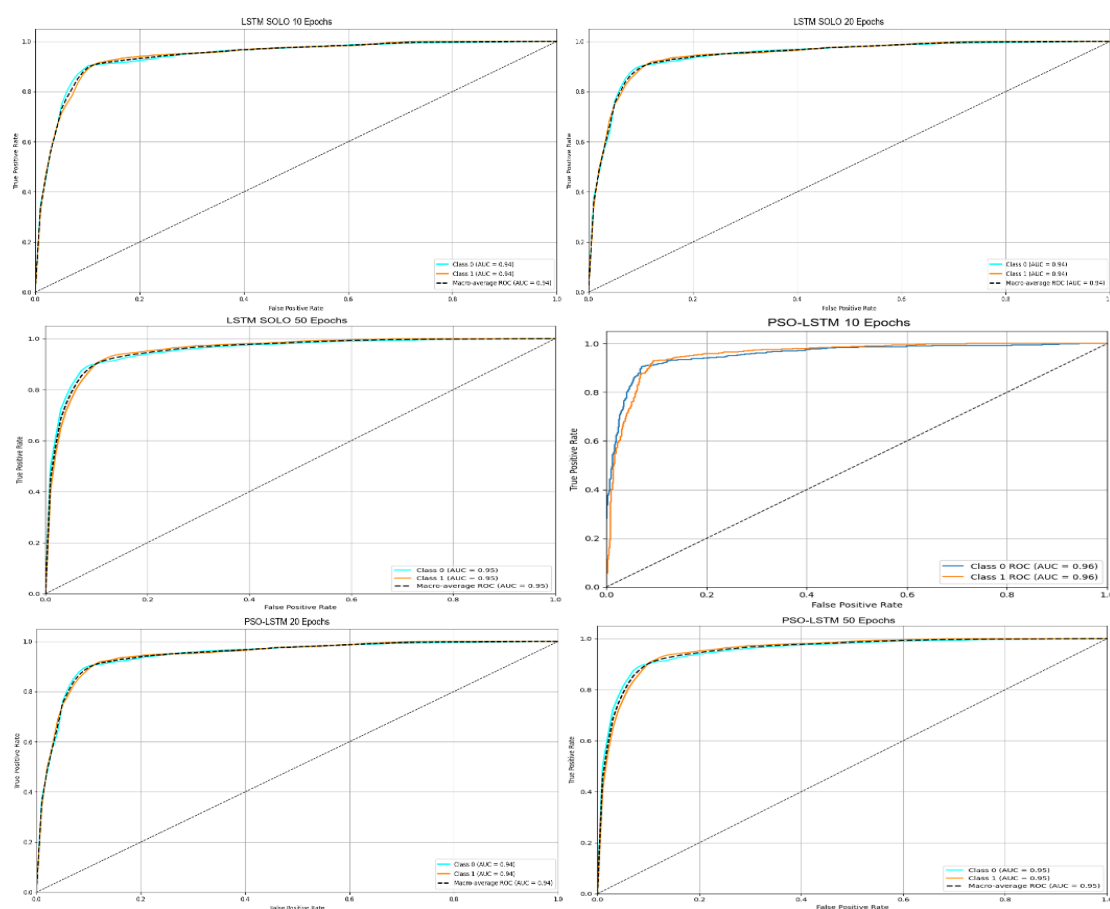


Fig. 6. Comparison of LSTM and PSO-LSTM Performance (ROC curves) with Droid det malware dataset.

Table 10. comparison of the results with other work already done.

Algorithms	Measures		
	Accuracy	Precision	Entropy
Random Forest [55]	90.93 %	90.92 %	0.1371
SVM [55]	89.79 %	89.72 %	0.1527
Rotation Forest [55]	88.10 %	88.03 %	0.1934
Naïve Bayes [55]	76.58 %	68.59 %	0.5430
LSTM SOLO	90.53%	90.55 %	0.1461
PSO-LSTM	91.65%	91.68%	0.1161

As shown in the table, the PSO-LSTM model achieved a strong accuracy of 91.65%, outperforming traditional machine learning algorithms like Random Forest (90.93%) and SVM (89.79%) [55]. The PSO-LSTM hybrid model combines the Particle Swarm Optimization (PSO) for the optimization and the automatic selection of the hyperparameter with the LSTM Model for the sequential learning capabilities. This hybrid model demonstrates superior performance. The LSTM SOLO model performed with an accuracy of 90.53%, which also show that it can offer competitive performance. Moreover, models such as Random Forest (90.93%) and SVM (89.79%) show reasonable accuracy [55]. SVM and LSTM SOLO can perform comparably, particularly when dealing with simpler data. Models as Naïve Bayes (76.58%) and Rotation Forest (88.10%) gives lower accuracy [55], highlighting the superior performance of the hybrid model (PSO-LSTM) and traditional machine learning algorithms like Random Forest.

In conclusion, the PSO-LSTM hybrid model demonstrates superior performance, especially in more complex tasks where hyperparameter optimization and sequential learning can make a significant advantage. LSTM SOLO and models like Random Forest and SVM are still strong contenders, especially in scenarios where computational complexity is a concern or the dataset does not require the advanced capabilities of PSO-LSTM.

DISCUSSION

In this work, we presented a hybrid Android malware detection approach that leverages permission features, LSTM neural networks, and hyperparameter optimization via PSO. In response to the rising volume and sophistication of Android-based threats, this research contributes a comparative framework for understanding the efficacy of various classification models in malware detection. The experimental results, which are applied to three different datasets, demonstrated that PSO significantly enhances the LSTM's performance by selecting optimal hyperparameters, achieving an accuracy of 96.07% on the Drebin dataset and an accuracy of 91.65% for the Droid det malware dataset, outperforming traditional methods. Future work will investigate the integration of API call sequences and network traffic features to further improve detection accuracy and reduce false positives in real-world deployments.

REFERENCES

- [1] C.Urcuqui, and A.Navarro (2016), Machine learning classifiers for android malware analysis, In Communications and Computing (COLCOM), 2016 IEEE Colombian Conference on (pp. 1-6). IEEE, April 2016.
- [2] U.Christian, Dataset malware/benign permissions android, <https://www.kaggle.com/xwolf12/datasetandroidpermissions>, 2016, Accessed 05th April 2025, at 22:41.
- [3] Soju, L. D. (n.d.). Drebin dataset [Data set]. Kaggle. Retrieved August 6, 2025, from <https://www.kaggle.com/datasets/likhithadurusoju/drebindataset/data>

- [4] Akbar, F. (2025) Droid det Malware Dataset. Retrieved August 11, 2025, from <https://github.com/fahadakbar24/android-malware-detection-dataset>
- [5] Keras, The python deep learning library, <https://keras.io> . Accessed 06th April 2025, at 11:16.
- [6] Pyswarm, the python pyswarm library, <https://pypi.org/project/pyswarm/>. Accessed 07th march 2025, at 11:02.
- [7] L.Yongfeng, S.Tong, S.Xin, P.Xuerui and M.Bing, "Detection classification and characterization of android malware using api data dependency", Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, China.
- [8] K.Y.Lok and Y.Heng (2012) "Droid scope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis, Syracuse University Air Force Research Laboratory Syracuse, , New York, USA, floyan, heying@syr.edu.
- [9] T.Kimberly, J.K.Salahuddin, F.Aristide, and C.Lorenzo, "Copper droid: automatic reconstruction of android malware behaviors", Systems Security Research Lab and Information Security Group, Royal Holloway University of London, NDSS '15, 8-11 February 2015, San Diego, CA, USA, Copyright 2015 Internet Society, ISBN 1-891562-38-X, <http://dx.doi.org/10.14722/ndss.2015.23145>
- [10] Z.Yajin and J.Xuxian, "Dissecting android malware: characterization and evolution, Department of Computer Science North Carolina State University, 2012, Yajin Zhou. Under license to IEEE Computer Society, DOI 10.1109/SP.2012.16
- [11] D.Gianluca, M.Fabio, S.Andrea and S.Daniele, "A multi-level anomaly detector for android malware", Computer science Department, University di Pisa, Pisa, Italy ,Springer-Verlag Berlin Heidelberg 2012
- [12] Enck, W., Ongtang, M., & McDaniel, P. (2009). Understanding Android security. IEEE Security & Privacy, 7(1), 50–57. <https://doi.org/10.1109/MSP.2009.26>
- [13] Arshad, S., Shahid, A., & Singh, A. (2016). Android malware detection & protection: A survey. International Journal of Advanced Computer Science and Applications, 7(2), 341–355. <https://doi.org/10.14569/IJACSA.2016.070262>
- [14] Ye, Y., Li, T., Adjeroh, D., & Iyengar, S. S. (2017). A survey on malware detection using data mining techniques. ACM Computing Surveys, 50(3), 1–40. <https://doi.org/10.1145/3073559>
- [15] Scikit-learn, Machine learning library in python, <https://scikit-learn.org/stable/> . Accessed 07th April 2025, at 14:23.
- [16] Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android permissions demystified. Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS), 627–638. <https://doi.org/10.1145/2046707.2046779>.
- [17] Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., & Rieck, K. (2014). Drebin: Effective and explainable detection of Android malware in your pocket. Proceedings of the Network and Distributed System Security Symposium (NDSS). <https://doi.org/10.14722/ndss.2014.23247>
- [18] Sahs, J., & Khan, L. (2012). A machine learning approach to Android malware detection. 2012 European Intelligence and Security Informatics Conference (EISIC), 141–147. <https://doi.org/10.1109/EISIC.2012.34>.
- [19] S.Asaf, K.Uri, E.Yuval, G.Chanan and W.Yael, " A behavioral malware detection framework for android devices", Journal of Intelligent Information Systems, February 2012, Volume 38, Issue 1, pp 161–190, Publisher: Springer.
- [20] D.Gianluca, M.Fabio, S.Andrea and S.Daniele, « A multi-level anomaly detector for android malware», International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2012: Computer Network Security pp 240–253, Publisher: Springer.
- [21] Huang, C., Xiao, H., Liu, Y., & Zhao, J. (2018). Permission-based Android malware detection using deep learning. Journal of Information Security and Applications, 41, 80–89. <https://doi.org/10.1016/j.jisa.2018.06.006>
- [22] Sgandurra, D., Zhou, L., & Muñoz, A. (2016). Automated dynamic analysis of Android malware using machine learning. ACM Transactions on Privacy and Security, 19(4), 1–28. <https://doi.org/10.1145/2996189>
- [23] Fan, L., Wang, Y., Liu, Y., & Liu, Q. (2020). Permission-based Android malware detection using hybrid deep learning model. IEEE Access, 8, 136085–136098. <https://doi.org/10.1109/ACCESS.2020.3010977>

- [24] Aafer, Y., Du, W., & Yin, H. (2013). DroidAPIMiner: Mining API-level features for robust malware detection in Android. *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, 86-103. <https://doi.org/10.1145/2484313.2484323>
- [25] Feizollah, A., Anuar, N. B., Salleh, R., & Radzi, S. A. (2015). A review on feature selection in mobile malware detection. *Digital Investigation*, 13, 22-37. <https://doi.org/10.1016/j.diin.2015.02.001>
- [26] <https://developer.android.com/guide/topics/permissions/overview?hl=fr>, Accessed 24th march 2025, at 01:32
- [27] Kaspersky. (2024, February 15). Banking data theft attacks on smartphones triple in 2024, Kaspersky reports. Kaspersky. <https://www.kaspersky.com/about/press-releases/banking-data-theft-attacks-on-smartphones-triple-in-2024-kaspersky-reports>, Accessed 24th march 2025, at 02:54
- [28] Securelist. (2024, January 10). Mobile threat report 2024. Securelist. <https://securelist.com/mobile-threat-report-2024/115494/>, Accessed 24th march 2025, at 02:54
- [29] TechRadar. (2024, March 5). These malicious Android apps were installed over 60 million times – here's how to stay safe. TechRadar. <https://www.techradar.com/pro/security/these-malicious-android-apps-were-installed-over-60-million-times-heres-how-to-stay-safe>, Accessed 24th march 2025, at 02:54
- [30] The Sun. (2024, February 28). Warning issued over Android apps stealing credit card details. The Sun. <https://www.thesun.co.uk/tech/33905327/warning-android-mobile-apps-bank-raid-stealing-credit-card/>, Accessed 24th march 2025, at 02:54
- [31] Chowdhury, M. N.-U.-R., Haque, A., Soliman, H., Hossen, M. S., Fatima, T., & Ahmed, I. (2023). Android malware detection using machine learning: A review. In A. A. Salah, R. Ramzan, & A. Bouras (Eds.), *Intelligent Systems and Applications* (Vol. 3, pp. 507–522). Springer. https://doi.org/10.1007/978-3-031-47715-7_35
- [32] Muzaffar, A., Ragab, H., Lones, M., & Zantout, H. (2022). An in-depth review of machine learning based Android malware detection. *Computers & Security*, 121, 102833. <https://doi.org/10.1016/j.cose.2022.102833>
- [33] Chowdhury, M. N.-U.-R., Haque, A., Soliman, H., Hossen, M. S., Fatima, T., & Ahmed, I. (2023). Android malware detection using machine learning: A review. In A. A. Salah, R. Ramzan, & A. Bouras (Eds.), *Intelligent Systems and Applications* (Vol. 3, pp. 507–522). Springer. https://doi.org/10.1007/978-3-031-47715-7_35
- [34] H.Fereidooni, M.Conti, D.Yao and A.Sperduti, « Android malware detection using static analysis of applications », Published in: 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Publisher: IEEE, E.ISBN: 978-1-5090-2914-3.
- [35] T.Chen, Q.Mao, Y.Yang, M. Lv and J. Zhu, “Tiny droid: a lightweight and efficient model for android malware detection and classification”, *Mobile Information Systems*, Volume 2018, Article ID 4157156, 9 pages, <https://doi.org/10.1155/2018/4157156>.
- [36] Mokri, M. A. E., Hamou, R. M., & Amine, A. (2019). A new bio-inspired technique based on octopods for spam filtering. *Applied Intelligence*, 49(9), 3425–3435. <https://doi.org/10.1007/s10489-019-01463-y>
- [37] Balasubramaniam, S., Kadry, S., Kumar, M. T. K., & Kumar, K. S. (Eds.). (2025). *Bio-inspired algorithms in machine learning and deep learning for disease detection*. CRC Press. <https://doi.org/10.1201/9781003538158>
- [38] Chowdhury, M. N.-U.-R., Haque, A., Soliman, H., Hossen, M. S., Fatima, T., & Ahmed, I. (2023). Android malware detection using machine learning: A review. In A. A. Salah, R. Ramzan, & A. Bouras (Eds.), *Intelligent Systems and Applications* (Vol. 3, pp. 507–522). Springer. https://doi.org/10.1007/978-3-031-47715-7_35
- [39] Shi, Y., & Eberhart, R. C. (1998). A modified particle swarm optimizer. *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, 69–73. <https://doi.org/10.1109/ICEC.1998.699146>
- [40] Sengupta, S., Basak, S., & Peters, R. A. II. (2019). Particle swarm optimization: A survey of historical and recent developments with hybridization perspectives. *Machine Learning and Knowledge Extraction*, 1(1), 157–191. <https://doi.org/10.3390/make1010010>
- [41] Grassi, S., Huang, H., Pareschi, L., & Qiu, J. (2021). Mean-field particle swarm optimization. *arXiv preprint arXiv:2108.00393*. <https://arxiv.org/abs/2108.00393>
- [42] Chiaradonna, S., Di Giandomenico, F., & Murru, N. (2020). On enhancing efficiency and accuracy of particle swarm optimization algorithms. *arXiv preprint arXiv:2004.06326*. <https://arxiv.org/abs/2004.06326>
- [43] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780. <https://doi.org/10.1162/neco.1997.9.8.1735>

- [44] Graves, A. (2013). Generative recurrent neural networks. In Proceedings of the 30th International Conference on Machine Learning (ICML 2013) (pp. 673-680).
- [45] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555. <https://doi.org/10.48550/arXiv.1412.3555>
- [46] Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. In Proceedings of the International Conference on Machine Learning (ICML 2014).
- [47] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10), 2222-2232. <https://doi.org/10.48550/arXiv.1503.04069>
- [48] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555. <https://doi.org/10.48550/arXiv.1412.3555>
- [49] Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.
- [50] Brownlee, J. (2019). *Machine learning mastery: Deep learning with Python*. Machine Learning Mastery.
- [51] Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems* (2nd ed.). O'Reilly Media.
- [52] Pathak, A., Barman, U., & Kumar, T. S. (2024). Static analysis framework for permission-based dataset generation and Android malware detection using machine learning. **EURASIP Journal on Information Security, 2024**(Article 33). <https://doi.org/10.1186/s13635-024-00182-3>
- [53] Malik, V., Goyal, S. K., & Malik, N. (2019). A hybrid model for Android malware detection. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 8(12), 2656-2662. <https://doi.org/10.35940/ijitee.K2250.1081219>
- [54] Arrowsmith, J., Susnjak, T., & Jang-Jaccard, J. (2025). Multimodal deep learning for Android malware classification. *Machine Learning and Knowledge Extraction*, 7(1), 23. <https://doi.org/10.3390/make7010023>
- [55] Akbar, F., Hussain, M., Mumtaz, R., Riaz, Q., Wahid Abdul Wahab, A., & Jung, K.-H. (2022). Permissions-based detection of Android malware using machine learning. *Symmetry*, 14(4), 718. <https://doi.org/10.3390/sym14040718>