



International Journal for
Electronic Crime Investigation

ISSN: 2522-3429 (Print)
ISSN: 2616-6003 (Online)

DOI: <https://doi.org/10.54692/ijeci.2026.1002/276>

Research Article

Vol. 10 issue 2 Jul-Dec 2026

Optimizing Android Malware Detection with Joint Feature Selection and Hidden Node Tuning: A BIGSA-HNO-ELM Framework

Shouzab Khan¹, and Muhammad Hassan²

¹Department of Computer Science, University of Alabama at Birmingham, Birmingham AL 35294, USA.

²School of Computer Science NCBA & E Lahore, Pakistan.

Corresponding Author: Sheikhhassan77758@yahoo.com

Received: July 30,2026 **Accepted:** August 2,2026 **Published:** August 21,2026

ABSTRACT

The growing complexity and scale of Android malware present significant challenges for the development of detection systems that are both accurate and computationally efficient. Traditional machine learning methods often struggle with high-dimensional feature spaces and poorly optimized model architectures, leading to degraded accuracy and excessive resource consumption. To address these limitations, we propose a unified malware detection framework—BIGSA-HNO-ELM—that integrates Binary Improved Gravitational Search Algorithm (BIGSA) for robust feature selection with Hidden Node Optimization (HNO) applied to an Extreme Learning Machine (ELM) classifier. Evaluated on the KronoDroid dataset comprising over 78,000 Android applications with 464 hybrid static and dynamic behavioural features, the proposed framework achieves a 77% reduction in feature dimensionality, significantly lowering computational overhead while preserving high detection accuracy. Through HNO, the optimal number of ELM hidden neurons is dynamically determined, enhancing generalization and reducing overfitting. Comprehensive experiments demonstrate that the BIGSA-HNO-ELM model achieves 98.0% accuracy, 97.4 % precision, 96.2 % recall, 96.8 % F1-Score and an AUC-ROC of 97.0%, substantially outperforming baseline methods. Statistical significance testing (paired t-tests and Wilcoxon signed-rank tests, $p < 0.01$) confirms the robustness of these performance gains and demonstrates superior stability across 10-fold cross-validation with minimal variance in performance metrics. This integrated optimization approach not only improves predictive performance but also supports deployment in resource-constrained and real-time environments, making BIGSA-HNO-ELM a compelling solution for modern Android malware detection.

Keywords: Android malware detection; Feature selection; BIGSA; Extreme Learning Machine (ELM); Hidden Node Optimization (HNO); Statistical testing; KronoDroid dataset

1. INTRODUCTION

As of 2024, Android continues to dominate the global smartphone market, holding over 72% of the total market share due to its open-source nature, flexibility, and widespread adoption by Original Equipment Manufacturers (OEMs) [1]. However, this openness and the resulting fragmentation across multiple OS versions introduce significant security challenges, making Android an attractive target for malware developers. The inherent vulnerabilities stemming from inconsistent patching and device heterogeneity have been widely discussed in the literature [2] [3]. According to cybersecurity experts in 2023, more than 3.2 million new Android malware variants were registered, which often used new advanced methods – code obfuscation and polymorphism for the gait of traditional signature-based antivirus applications [4]. Because malware is always evolving, it is clear that relying just on traditional malware detectors is insufficient, and we intend more intelligent security methods.

Machine learning has emerged as an effective strategy in combating the increasingly sophisticated Android malware, leveraging millions of static and dynamic features (e.g., application permissions, API call sequences, and runtime behavior patterns) [5] to detect malicious applications. These models can be deployed and adapted in a variety of situations, outperforming traditional signature-based approaches. Nonetheless, machine learning models have several drawbacks as detectors. Model efficiency degrades over time due to concept drift (a phenomenon where attackers change strategies) [6] [7]. Furthermore, the high-dimensional nature of the extracted features (up to thousands) is associated with redundancy, high computational requirements, and a high

Int. J. Elect. Crime Investigation 10(2): IJECI MS.ID- 02 (2026)

likelihood of overfitting if feature selection is not fully optimized [8].

At the same time, metaheuristic methods have rapidly gained popularity in malware identification systems due to their ability to filter out the most valuable features from massive and complex data sets. For example, BIGSA applies a physics-based model to discover the most informative feature subset by considering potential solutions as mutually attractive masses [9] [10]. This method can remove superfluous variables, thereby enhancing the capabilities of subsequent classifiers. In addition, ELM (a single hidden layer feedforward neural network) has become an ideal classifier because they are fast to train, have few parameter settings, and have strong generalization capabilities, which are particularly suitable for mobile and IoT devices [11]. Nevertheless, the usual practice of separating feature selection from classifier training often means lower performance, leading researchers to build systems that optimize both in one process.

Despite remarkable gains, there is still a significant gap in malware detection research: feature selection and classifier tuning are often done separately. This segmented approach reduces efficiency—particularly on devices with limited resources—and frequently fails to adapt as the threat landscape changes. Furthermore, most existing techniques provide little model adaptability when it comes to dealing with idea drift or generalising to new assault patterns. To address these drawbacks, an integrated, adaptive framework that conducts feature selection and classifier optimisation simultaneously in a lightweight, computationally efficient way is critical.

To address this essential gap, this paper suggests a unified detection framework termed BIGSA-HNO-ELM. The suggested methodology joins BIGSA for optimal feature selection and is directly connected to the Hidden Node Optimization (HNO) of an ELM classifier. Although the preceding work has investigated metaheuristic feature selection or classifier tuning separately, the suggested BIGSA-HNO-ELM mechanism uniquely pools both in a tightly integrated, co-optimized loop—thus attaining better strength and performance. This integration jointly optimizes feature selection and classifier architecture, significantly enhancing model accuracy, generalization performance, and computational efficiency. Evaluated on a meticulously curated subset of the KronoDroid dataset, comprising over 78,000 Android apps characterized by 464 hybrid (static and dynamic) features [12], the proposed BIGSA-HNO-ELM framework achieves notable performance enhancements. Specifically, BIGSA reduces the feature dimensionality by approximately 77%, substantially decreasing computational load without compromising predictive accuracy. Simultaneously, HNO dynamically identifies the optimal hidden node configuration, ensuring robust performance against concept drift and improving generalization. Experimentally, the proposed model achieves superior classification accuracy (98.0%), precision (97.4%), recall (96.2%), F1-Score (96.8 %), and an AUC of 97.0%. To approve the strength of these enhancements, we engaged paired t-tests and Wilcoxon signed-rank tests ($p < 0.01$) [13] [14], signifying statistical significance over traditional machine learning methodologies.

The contributions of this research are summarized as follows:

1. Introduces a novel integration of BIGSA-based feature selection with HNO-based ELM classifier tuning, forming a unified and co-optimized malware detection framework that enhances accuracy and computational efficiency.
2. Attains a significant feature reduction in feature dimensionality approximately ~77% reduction, which reduces model complexity and resource usage while maintaining high predictive performance.
3. Enhances model generalization by dynamically optimizing the number of hidden neurons in the ELM via HNO, with performance improvements rigorously validated using statistical significance tests to ensure robustness across cross-validation folds and real-world conditions

The rest of the article is organized as follows: Section 2 provides a comprehensive analysis of relevant literature on Android malware detection, feature selection approaches, metaheuristic optimization, and ELM classifiers. Section 3 designates the suggested methodology broadly, containing data preprocessing, BIGSA feature selection, ELM classification with HNO, and evaluation processes. Section 4 details experimental outcomes and their implications, while Section 5 accomplishes research insights, practical implications, and future research objectives.

2. LITERATURE REVIEW

Android malware detection research has evolved across three major analytical paradigms—static, dynamic, and hybrid analysis. Each method contributes unique advantages, though they also exhibit limitations.

Static analysis investigates app code, permissions, and manifest files without execution, enabling early-stage identification. Tools like Drebin have shown strong performance in classifying malware via linear SVMs using manifest and API features [2]. However, these methods struggle with obfuscated, encrypted, or dynamically loaded code [15]. Dynamic analysis operates on executing applications and recording behaviours such as API calls, system interactions, and network activities. For instance, MamaDroid constructs behavioural Markov chains to model malware evolution [16]. Although effective against code evasion, dynamic methods suffer from sandbox bypassing and high runtime costs [17].

Hybrid analysis, which merges static and dynamic perspectives, offers a balanced view. The KronoDroid dataset exemplifies this by integrating over 460 features from both domain [12]. Advanced hybrid approaches such as MaskDroid use masked graph neural networks to withstand polymorphic and obfuscated attacks [18].

Building upon established analytical structures, traditional machine-learning models have long served as the foundation for malware classification efforts. Algorithms including Random Forests, Support Vector Machines (SVM), and XGBoost are recurrently employed because of their reliability and strong performance, mainly when dealing with complex and high-dimensional datasets [19] [20]. Despite their merits, these models frequently lack scalability, rendering them unsuitable for real-time or large-scale data processing responsibilities [21].

Meanwhile, multi-layer perceptrons (MLPs) have expanded in popularity because of their capacity to signify nonlinear data. In contrast, MLPs are prone to overfitting and require vigilant hyperparameter modification [22]. Concerned with the increasing sophistication of malware, deep learning has appeared as a possible solution. New research on Android malware detection has widely assessed topologies including long short-term memory (LSTM) networks and deep belief networks (DBNs). For example, by employing the KronoDroid dataset, LSTM is recycled to model the temporal relationship of the behavioural sequence of API calls and system events [23]. DBN is trained by utilizing static and dynamic merge features to accomplish hierarchical pattern learning; and advanced optimized using the evolutionary-based methods such as PSO for intrusion detection [24] [25]. These models truthfully represent the temporal and hierarchical patterns in behavioural data [23] [26]. Conversely, such models are resource-intensive and difficult to comprehend, making them inappropriate for performance-constrained devices (such as smartphones) [27].

To lessen this gap, contemporary studies have highlighted lightweight and interpretable models. For instance, the authors introduced Fedroid [28] which combines static feature-based detection with real-time interpretability, demonstrating the growing demand for explainable AI in cybersecurity contexts. The authors [27] suggested a metaheuristic-empowered deep learning method for scalable and effective malware classification. Compatible, the authors [21] anticipated an assorted DBN-GRU architecture for accurate learning of both static and dynamic behaviours.

The high-dimensional feature space apparent in Android datasets is a significant challenge for all of these approaches. Feature selection (FS) is critical for reducing overfitting, increasing accuracy, and lowering computing costs. FS methodologies are categorised as:

- Filter approaches, such as Chi-square, and InfoGain are fast but neglect feature dependencies [29].
- Wrapper approaches, such as GA, and PSO are performance-driven yet computationally expensive [30].
- Embedded approaches, such as LASSO and decision trees are effective yet model-specific [31].

Metaheuristic algorithms such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and BGSA are widely applied in malware detection for feature selection due to their capability to handle complex, high-dimensional spaces [32]. BGSA, grounded in Newtonian gravitational dynamics, has been adapted into binary space for discrete feature selection tasks [9]. Subsequent enhancements such as BIGSA further improve convergence and diversity by incorporating adaptive velocity updates and global memory [10]. For example, the authors [33] proposed a hybrid method combining BGSA with mutual information to enhance feature relevance in intrusion detection, demonstrating improved accuracy and dimensionality reduction. More recently, the authors [34] applied evolutionary feature selection to Android malware classification, reinforcing that such heuristic approaches yield superior classifier performance and compact representations. These studies support the integration of advanced metaheuristics like BIGSA within scalable malware detection

frameworks such as our proposed BIGSA-HNO-ELM.

The effectiveness of BIGSA has encouraged its integration with various classifiers, particularly ELMs. ELMs are single-layer feedforward neural networks renowned for their extremely fast training speed due to random initialization of input weights and biases [11]. However, ELM performance is highly sensitive to the number of hidden neurons, which impacts generalization, accuracy, and training stability. A relevant approach by [35] introduced a hybrid PSO-ELM system for general intrusion detection using network traffic data, which, although not specific to Android malware, demonstrates the benefits of optimizing neuron counts through swarm-based heuristics, showing that neuron optimization significantly improves detection performance. While few studies directly address this co-optimization in Android malware, the BIGSA-HNO-ELM framework fills this research gap by tuning both the input feature space and the ELM structure simultaneously.

Recent literature continues to explore hybrid models combining feature selection with classifier enhancement. For instance, the researchers [36] integrated metaheuristic feature selection with ELM but handled the two components independently, lacking a unified optimization strategy. The researchers [37] presented a reinforcement learning-based FS mechanism that was incorporated with an XGBoost classifier but did not jointly optimize the classifier's architecture. On the other hand, the authors [38] established a DBN-based structure for Android malware detection, displaying that deep learning jointly with optimization methods can expressively advance detection performance. Correspondingly, the researchers, the authors [21] presented a DBN-

GRU hybrid model that attains high accuracy by capturing both static and dynamic behaviours a trait aligned with our suggested BIGSA-HNO-ELM framework. These investigations uphold the potential of integrating optimization with deep learning but highlight the lack of fully co-optimized ELM-based models which is addressed directly by the suggested BIGSA-HNO-ELM framework. Recent studies increasingly emphasize hybrid deep learning architectures for handling complex temporal patterns and improving prediction accuracy. Similar to these efforts, the proposed TCN–LSTM–Attention framework integrates complementary learning mechanisms, while highlighting the broader need for jointly optimized models that can effectively capture both local patterns and long-term dependencies [43]. Some researches has increasingly focused on domain adaptation and representation learning to improve CPDP (Cross-Project Software Defect Prediction) performance under heterogeneous project conditions. These studies

demonstrate the value of learning transferable representations, while also highlighting the need for robust feature integration and domain alignment to enhance generalization across different software projects [44]. Recent studies have demonstrated the effectiveness of advanced AI architectures for prediction and classification. CodeSage-GNN [45] integrates graph-based structural and semantic features for software defect prediction, while EfficientNetB0 [46] highlights the role of deep feature extraction in image classification. Similarly, ProfileSwitchNet [47] applies sequence-aware learning for predictive modeling using historical data. These studies indicate that combining diverse feature representations with advanced learning architectures can improve predictive performance, supporting the need for robust and effective models in complex prediction tasks. To contextualize the effectiveness of recent frameworks, Table 1 summarizes state-of-the-art Android malware detection models:

Table 1. State-of-the-Art Android Malware Detection Models

Framework	Year	Core Technique	Accuracy	Notable Features
Fedroid [11]	2025	Static FS + Explainable Learning	97.8%	Interpretable static features; lightweight deployment
DBN-GRU [21]	2020	Hybrid Deep Learning (DBN + GRU)	98.3%	Static + dynamic analysis; effective feature fusion
MetaDL [27]	2023	Metaheuristic + Deep Learning	98.1%	Efficient FS with deep neural architectures
RL-FS [37]	2023	Reinforcement Learning-based FS	97.5%	RL-based selection; lacks classifier co-optimization
BIGSA-HNO-ELM (Proposed Work)	2025	BIGSA-based FS + HNO-optimized ELM	98.0%	Joint FS & classifier tuning; 77% feature reduction; fast model

The **BIGSA-HNO-ELM** structure presents persistent limitations by jointly optimizing feature subsets and ELM architecture. BIGSA reduces feature count by 77%, improving Int. J. Elect. Crime Investigation 10(2): IJECI MS.ID- 02 (2026)

efficiency without compromising accuracy. HNO identifies the optimal neuron configuration, enhancing generalization. This co-optimized loop improves detection accuracy (98.0%) and AUC (97.0%), validated 25

statistically through paired t-tests and Wilcoxon tests ($p < 0.01$). This integrated approach enables real-time, resource-efficient, and adaptive malware detection suitable for deployment on constrained devices.

3. METHODOLOGY

This section presents the methodological framework for the proposed BIGSA-HNO-ELM model, emphasizing data preparation, feature optimization using BIGSA, classification with ELM, and dynamic model tuning via HNO. Statistical hypothesis testing is employed to validate feature significance, ensuring both empirical robustness and domain relevance.

3.1. Dataset Description

The KronoDroid dataset [12] comprises 78,000 Android applications (39,500 benign and 38,500 malicious), each represented by 464 hybrid features combining static and dynamic attributes. After removing non-informative metadata (e.g., app index), 462 features were retained, categorized as follows:

- **Dynamic Features:** System call counts (e.g., `execve`, `getuid32`, `kill`), network activity, and runtime behaviors.
- **Static Features:** Permission counts (e.g., `INTERNET`, `SEND_SMS`), API calls, and manifest-derived metrics.

The class labels are binary: 0 (benign) and 1 (malicious). While the original dataset is nearly balanced (51% malicious, 49% benign), synthetic minority oversampling (SMOTE) was applied during preprocessing to mitigate subtle imbalances in training subsets.

3.2 Data Preprocessing

Effective preprocessing of high-dimensional Android malware data is essential for improving model performance, minimizing bias, and ensuring consistency. The raw feature vectors extracted from the KronoDroid dataset underwent the following systematic preprocessing steps:

3.2.1 Label Encoding

The classification task is binary—distinguishing between benign and malicious applications. As such, categorical labels were numerically encoded using label encoding, where:

$$y = \begin{cases} 0 & \text{if benign} \\ 1 & \text{if malicious} \end{cases} \quad (1)$$

This encoding is suitable for binary classifiers and preserves ordinal relationships required for certain machine-learning algorithms.

3.2.2. Missing Value Imputation

Despite the high-quality curation of the KronoDroid dataset, all the records containing gaps in numerical feature values were processed with mean imputation. Imputation with the mean replaces null values with the average of the non-missing values of the corresponding feature:

$$x_i^* = \begin{cases} x_i & \text{if } x_i \neq \text{NaN} \\ \frac{1}{n} \sum_{j=1}^n x_j & \text{if } x_i = \text{NaN} \end{cases} \quad (2)$$

Where x_i signifies the actual feature value, and n represents the number of non-missing values. This method preserves the dataset's statistical features while confirming computational integrity throughout training (Liu et al., 2020).

3.2.3. Feature Normalization

All the input features were normalized with Min-Max scaling so that the input values were mapped to the range [0, 1] in order to confirm consistent scale and avoid bias for features that have higher numeric values.

$$x_i^{norm} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (3)$$

Where x_i designates the raw features, and x_{min}, x_{max} represents its minimum and maximum values throughout the dataset. Normalization advances the learning algorithms' convergence and is particularly significant when employing kernel-based or distance-based models (Fatima et al., 2019).

3.2.4 Class Imbalance Mitigation using SMOTE (Synthetic Minority Over-sampling Technique)

The dataset exhibited a mild degree of class imbalance, with slightly more malicious apps. To avoid biased learning, SMOTE [39] was used for symmetric class distribution. SMOTE creates synthetic data points for the minority class through linear interpolation between a data point and its k-nearest neighbours as follows:

$$X_{synthetic} = X_{sample} + \lambda \cdot (X_{nearest} - X_{sample}) \quad (4)$$

Where $\lambda \in [0,1]$ signifies a random scalar. $X_{nearest}$ exemplifies the nearest neighbour of the minority class sample X_{sample} . This method creates synthetic samples of the minority class by balancing the class distribution. Now the transformed data is prepared for joint feature selection and classification.

3.3. Feature Selection using BIGSA

The BIGSA optimizes feature selection by reducing redundancy and exploiting discriminative potential. BIGSA is a population-based metaheuristic inspired by Newton's laws of gravitation and motion, refined with global memory, adaptive velocity control, and a binary encoding scheme to handle discrete feature selection tasks. Each particle in the search space represents a candidate binary vector $X = [x_1, x_2, \dots, x_d]$ where $x_i \in [0,1]$ determines the inclusion or exclusion of the i^{th} feature. The objective of BIGSA is to maximize classification accuracy while minimizing the number of selected features, reducing dimensionality and enhancing computational efficiency. To evaluate the fitness of each particle (feature subset). The fitness function is defined as:

$$F_{fitness} = \frac{Accuracy(S)}{1 + \lambda \cdot |S|} \quad (5)$$

Where, S is the subset of selected features, $Accuracy(S)$ is the classification accuracy on the validation set using the selected features, and $\lambda = 0.01$ is a regularization parameter controls the trade-off between accuracy and sparsity of the feature set. A larger λ places more emphasis on feature sparsity, encouraging a more compact solution. This fitness function helps balance high accuracy with minimal feature usage.

Each particle's mass is calculated based on its fitness using:

$$M_i(t) = \frac{f_i(t) - f_{worst}(t)}{f_{best}(t) - f_{worst}(t)} \quad (6)$$

Where: $f_i(t)$ is the fitness of the i^{th} particle at iteration t , f_{best} and f_{worst} represent the best and worst fitness in the population at iteration t . This update ensures that particles with better fitness

(i.e., those that yield better feature subsets) have a greater influence on other particles.

The gravitational force between particles i and j in dimension k is given by:

$$F_{ij}^k(t) = G(t) \cdot \frac{M_i(t) \cdot M_j(t)}{R_{ij}(t) + \varepsilon} \cdot (x_j^k(t) - x_i^k(t)) \quad (7)$$

Where, $R_{ij}(t)$ is the Euclidean distance between particles i and j in dimension k of the feature space:

$$R_{ij}(t) = \|x_i^k(t) - x_j^k(t)\|_2 \quad (8)$$

$G(t)$ is the gravitational constant at iteration t , which decays over time to guide the particles from exploration to exploitation:

$$G(t) = G_0 \cdot e^{-\beta \cdot t / T} \quad (9)$$

Where, G_0 is the initial gravitational constant, β is a decay factor that controls the rate of decay, ε is a small constant to avoid division by zero. This force guides particles toward the optimal feature subset.

After calculating the gravitational forces, the velocity update includes a memory component to encourage the exploitation of promising regions:

$$v_i^k(t+1) = r_i \cdot v_i^k(t) + a_i^k(t) + \gamma \cdot (x_{g_{best}}^k - x_i^k(t)) \quad (10)$$

Where: r_i is a random number that introduces stochastic behaviour, $a_i^k(t)$ is the acceleration of particle i in dimension k , based on the gravitational force, $\gamma = 0.5$ is a decay factor, controls the influence of the global best solution $x_{g_{best}}^k$, ensuring exploitation in later iterations.

The rule for updating the position is based on a probabilistic binary update, as follows:

$$S(v_i^k) = \frac{1}{1 + e^{-v_i^k}} \quad (11)$$

This transfer function transforms the continuous velocity v_i^k to a probability $S(v_i^k)$, indicating the likelihood of picking feature k in the current iteration. The rule for updating the position is as follows:

$$x_i^k(t+1) = \begin{cases} 1 & \text{if } rand() < S(v_i^k) \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

Where, $rand()$ is a random variable uniformly distributed between 0 and 1, confirming that the update is probabilistic. This permits the algorithm to successfully traverse the discrete features [10]. The algorithm iterates for a predefined number of iterations T or (until convergence) and picks the optimum feature subset by considering the particle's fitness. The resulting binary vector X_{best} reflects the selected feature subset, and is employed as input to the ELM classifier for further assessment. Table 2 designates the pseudocode for the suggested BIGSA for Optimal Feature Selection.

Table 2 BIGSA for Optimal Feature Selection

Input: Preprocessed dataset $D \in \mathbb{R}^{n \times 462}$, population size N , maximum iterations T

Output: Optimal binary feature vector X_{best}

1. Initialize population of N particles $X_i \in \{0,1\}^{462}$, represents a candidate solution (binary vector of 462 features), velocities V_i , and set the iteration count $t = 0$
2. Evaluate the fitness of each particle X_i using Eq. (5)

$$fitness_i = \frac{Accuracy(X_i)}{1 + \lambda \cdot |S|}$$

3. Compute mass for each particle using Eq. (6):

$$M_i(t) = \frac{fitness_i(t) - fitness_{worst}(t)}{fitness_{best}(t) - fitness_{worst}(t)}$$

4. Repeat until $t = T$

- a. Calculate total force F_i^k using Eq. (7)

$$F_i^k = rand.G(t) \cdot \frac{M_j(t) \cdot (X_j^k(t) - X_i^k(t))}{\|X_j - X_i\| + \epsilon}$$

- b. Update velocity V_i^k using Eq. (10):

$$V_i^k(t+1) = r_1 \cdot V_i^k(t) + r_2 \cdot F_i^k(t) + r_3 \cdot (X_{g_{best}}^k - X_i^k(t))$$

- c. Apply transfer function to map velocity to probability using Eq. (11).

$$S(V_i^k) = \frac{1}{1 + e^{-v_i^k}}$$

- d. Update position $X_i^k(t+1)$ using Eq. (12):

$$x_i^k(t+1) = \begin{cases} 1, & \text{if } rand() < S(v_i^k) \\ 0, & \text{otherwise} \end{cases}$$

- e. Recalculate fitness and update $X_{g_{best}}$ if improved

- f. Increment $t = t + 1$

5. **Return** the final best feature subset X_{best} , representing the optimal feature set selected after T iterations or convergence.

3.3. BIGSA Hyperparameter Settings

Explicitly list each parameter used in the BIGSA along with its recommended setting and a brief rationale for each choice is shown in Table 3

3.4. Classification using ELM with HNO

The next stage of our Android malware detection framework involves classification using the ELM, a single-hidden-layer feedforward neural network (SLFN) known for its fast training speed, simplicity, and generalization ability [11]. Unlike conventional neural networks, ELM randomly initializes input weights and biases and computes output weights analytically, eliminating the need for iterative

backpropagation. The architecture is shown in Figure 1.

Let the training dataset be represented as $\{(x_i, t_i) | x_i \in \mathbb{R}^n, t_i \in \mathbb{R}^m, i = 1, \dots, N\}$ where x_i is the input vector (feature subset selected by BIGSA), t_i is the corresponding binary class label (0 = benign, 1 = malicious), and N is the number of samples.

The output of the ELM for L hidden neurons is defined as:

$$f(x) = \sum_{i=1}^L \beta_i \cdot h_i(x) \quad (13)$$

where: $h_i(x) = g(w_i \cdot x + b_i)$ is the output of the i^{th} hidden node, $w_i \in \mathbb{R}^n$ is the randomly initialized weight vector connecting inputs to the hidden node, $b_i \in \mathbb{R}$ is the bias term, $\beta_i \in \mathbb{R}^m$ is the output weight vector, and $g(\cdot)$ is the nonlinear activation function (typically sigmoid or ReLU).

Let $H \in \mathbb{R}^{N \times L}$ denote the hidden layer output matrix, defined as:

Table 3: BIGSA Hyperparameter Settings

Parameter	Recommended Setting	Rationale
Population Size	50	Balances exploration capability and computational efficiency.
Number of Iterations	100	Ensures thorough convergence without excessive computation.
Gravitational Constant (G_0)	100	Provides initial strong attraction promoting exploration.
Decay Rate of G	0.2 (20%) per iteration	Gradual transition from exploration to exploitation.
Elite Solution Rate	Top 10%	Maintains quality solutions to guide global optimization.
Mutation Rate	0.1	Balanced mutation to maintain diversity and prevent stagnation.
Regularization Parameter (α)	0.01	Balances solution accuracy with feature set compactness.

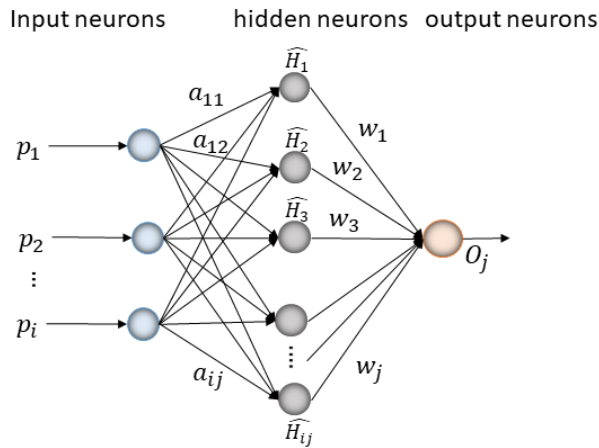


Figure 1: ELM Architecture

$$H = \begin{bmatrix} h_1(x_1) & \dots & h_L(x_1) \\ \vdots & \ddots & \vdots \\ h_1(x_N) & \dots & h_L(x_N) \end{bmatrix} \quad (14)$$

Then the output weights $\beta \in \mathbb{R}^{L \times m}$ are computed via the **Moore–Penrose pseudoinverse**:

$$\beta = H^+T \quad (15)$$

Where , $T \in \mathbb{R}^{N \times m}$ is the target label matrix.

3.5. HNO

Despite ELM's efficiency, its classification performance is highly sensitive to the number of hidden neurons L . A small L may lead to underfitting, whereas a large L increases the computational burden and overfitting risk [40]

[41]. Therefore, we propose an HNO mechanism to automatically determine the optimal L that maximizes detection accuracy while minimizing overfitting.

In this study, the search space for L is defined as $L \in [10, 300]$. For each candidate value of L , a model is trained using the feature subset selected by BIGSA, and evaluated on a validation set using the F1-score as the selection criterion:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (16)$$

Where,

- $\text{Precision} = TP / (TP + EP)$ is the proportion of correctly classified malicious samples.
- $\text{Recall} = TP / (TP + FN)$ is the proportion of correctly identified malicious samples.

Here, TP, FP, and FN represent true positives, false positives, and false negatives, respectively.

To determine the optimal number of hidden neurons, we apply HNO within a nested cross-validation structure. Specifically, for each fold of the outer 10-fold cross-validation used for performance evaluation, the training set is further split into an inner training and validation

set. HNO is conducted on this inner split, where each candidate architecture is trained on the inner training set and validated on the inner validation set. The configuration yielding the highest F1 score on the validation subset is then selected. This prevents information leakage from the test data and ensures robust, unbiased model tuning. Once the optimal architecture is identified, the model is retrained on the full training fold and evaluated on the held-out test fold of the outer loop.

The optimized ELM classifier is finally retrained on the entire training dataset using the selected number of hidden neurons L and evaluated on unseen test samples using the performance metrics accuracy, precision, recall, F1-score, and AUC-ROC.

This adaptive tuning provides a balance between classification capacity and generalization, essential for real-time malware detection where both speed and accuracy are critical.

3.6. HNO-ELM Hyperparameter Settings

Explicitly list each parameter used in the HNO-ELM based on the proposed methodology along with its experimentally validated setting and a concise rationale is shown in Table 4.

Table 4: HNO-ELM Hyperparameter Settings\

Parameter	Recommended Setting	Rationale
Initial Hidden Neurons	20	Serves as a conservative start to explore capacity without overfitting.
Neuron Increment Step	10 neurons per step	Provides steady search granularity in architectural tuning.
Maximum Hidden Neurons	200	Defines upper search bound, aligned with observed performance plateau.
Activation Function	Sigmoid	Validated experimentally for stable convergence and nonlinear mapping.
ELM Regularization Parameter C	1.0	Controls overfitting while retaining fitting capacity across feature subsets.

Early Stopping Criteria	0.01% improvement over 5 iterations	Ensures stopping once marginal gain plateaus, improving efficiency
-------------------------	--	---

3.7. Evaluation and Performance Metrics

To evaluate the effectiveness of the proposed BIGSA-ELM framework, we adopted a rigorous experimental setup using the KronoDroid dataset. The dataset was stratified and split into 80% training and 20% testing subsets. A 10-fold stratified cross-validation was further employed during model selection to ensure generalization and prevent overfitting.

The classification performance was assessed using six well-established metrics:

- **Accuracy:** Measures the proportion of correctly classified samples over the total.
- **Precision:** Indicates the proportion of malware-labeled apps that are truly malicious.
- **Recall (Sensitivity):** Reflects the model's ability to detect actual malware.
- **F1-Score:** The harmonic mean of precision and recall, critical for imbalanced datasets.
- **AUC-ROC:** Represents the area under the Receiver Operating Characteristic curve, summarizing the trade-off between true and false positive rates.

The evaluation metrics were derived from the confusion matrix using the following formulas:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F_1 - \text{Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{AUC} - \text{ROC} = \int_0^1 TPR(FPR) d(FPR),$$

estimated via trapezoidal rule

Along with performance evaluation metrics, we also merged statistical significance testing to confirm that the observed enhancements in classification performance were not due to random variation. In particular, paired t-tests and Wilcoxon signed-rank tests (with a significance threshold of $p < 0.01$) were conducted to equate BIGSA-HNO-ELM alongside baseline models across many cross-validation folds. These statistical tests prove the robustness and reliability of the proposed method's performance improvements. This additional validation layer ensures that our model's enhancements are both empirically and statistically valid.

All experimental procedures were implemented in Python 3.10 using NumPy, sci-kit-learn, and TensorFlow libraries. The experiments were conducted on a workstation with an Intel i9 processor, 64 GB RAM, and NVIDIA RTX 3080 GPU. The proposed framework of BIGSA-HNO_ELM is shown in Figure 2.

Optimizing Android Malware Detection with Joint Feature Selection and Hidden Node Tuning: A BIGSA-HNO-ELM Framework

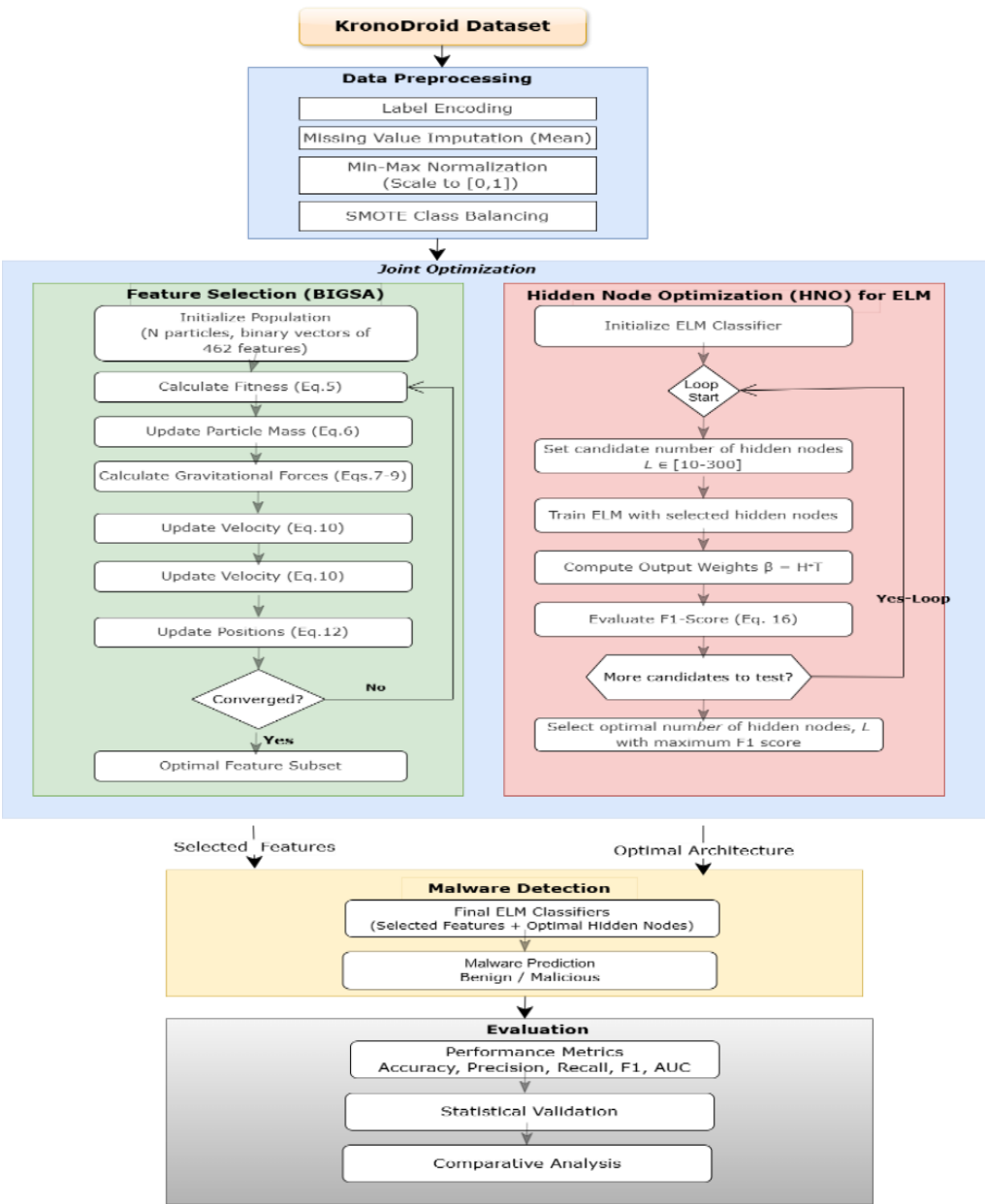


Figure 2: Proposed Framework of BIGSA-HNO_ELM

4. RESULTS AND DISCUSSION

This section presents a rigorous evaluation of the BIGSA-HNO-ELM framework using the KronoDroid dataset, focusing on feature

selection, classifier performance, AUC-ROC, statistical significance, and real-world applicability. Experimental findings are grounded in visual and tabular representations,

emphasizing empirical robustness and practical relevance.

4.1. Feature Selection Frequency by BIGSA

Figure 3 illustrates the frequency with which features were selected by BIGSA across 30 independent optimization runs, now visualized with standard deviation error bars to capture variability. This variability reflects the stability of feature selection and informs downstream classifier performance and decision boundary consistency. High variance in feature frequency may signal sensitivity to initialization or data splits, which is critical for evaluating the robustness and interpretability of features in real-world deployments.

Features such as RECEIVE_BOOT_COMPLETED, WRITE_EXTERNAL_STORAGE, and VIBRATE exhibit high selection frequency with minimal variance, indicating stable contributions to classification accuracy. Conversely, features like SEND_SMS, execve, and INTERNET—despite being statistically significant as shown in Table 5—show greater selection variability, suggesting that their predictive contribution may depend on context or interactions with other features.

This divergence highlights an important insight: BIGSA, as a population-based and multivariate heuristic, can uncover feature interdependencies that univariate statistical tests like Chi-square and ANOVA may miss. While traditional tests evaluate features in isolation, BIGSA's multivariate approach selects feature subsets that collectively maximize classification accuracy. Only two features—WRITE_EXTERNAL_STORAGE and execve—appear prominently in both the

BIGSA-selected and statistically validated sets, emphasizing the need to integrate both heuristic and statistical insights for a comprehensive evaluation.

Moreover, the observed variability in selection frequency is not just a technical artefact—it has operational implications. Features with high variance may cause unstable decision boundaries and unreliable predictions, especially under adversarial or evolving data conditions. Figure 3 thus serves as more than a bar chart: it provides a visual proxy for expected model robustness and feature generalizability.

4.2. Comparison of Feature Selection Methods

Figure 4 compares classification accuracy against the number of selected features for various feature selection techniques. The proposed BIGSA method outperforms all others by achieving the highest accuracy (98.0%) using the fewest features (~110), resulting in a 77% reduction in dimensionality. This positions BIGSA as the most efficient technique for malware detection on resource-constrained platforms.

Heuristic methods like GA and PSO demonstrate moderate accuracy (93–94%) but require more features (160–180), increasing inference time and memory consumption. Linear methods like LASSO and unsupervised techniques such as PCA and Mutual Information (MI) retain even more features (>230) while yielding lower accuracy (~91–92%), highlighting their inability to capture malware-specific nonlinear patterns effectively.

Optimizing Android Malware Detection with Joint Feature Selection and Hidden Node Tuning: A BIGSA-HNO-ELM Framework

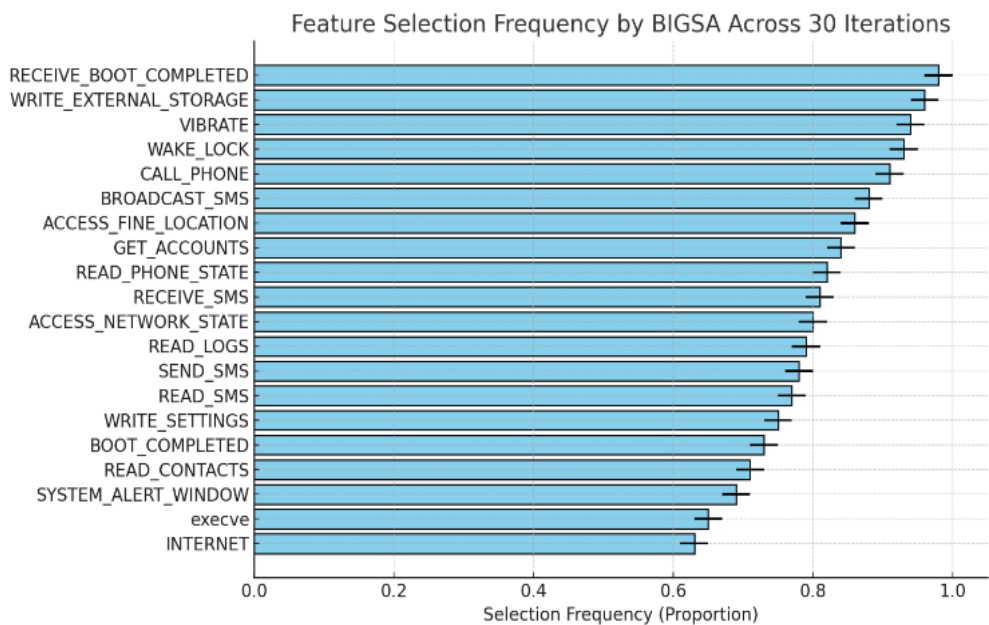


Figure 3: Feature selection frequency by BIGSA across 30 iterations with standard deviation error bars

Table 5: Statistical Significance of Top Features

Feature	Test Type	Statistic	p-value	Significant
SEND SMS	Chi-square	142.7	< 0.001	Yes
INTERNET	Chi-square	136.2	< 0.001	Yes
execve	ANOVA	89.2	< 0.001	Yes
WRITE EXTERNAL STORAGE	Chi-square	78.5	< 0.001	Yes
getuid32	ANOVA	67.3	< 0.001	Yes

This confirms that BIGSA’s optimization, which simultaneously considers sparsity and accuracy, leads to a compact yet highly predictive feature subset. Paired t-tests ($p < 0.01$) further validate that BIGSA significantly outperforms GA and PSO.

From a deployment perspective, reducing feature dimensionality offers additional benefits: faster prediction, reduced memory usage, and lower overfitting risk—especially important in mobile, embedded, or time-sensitive security applications.

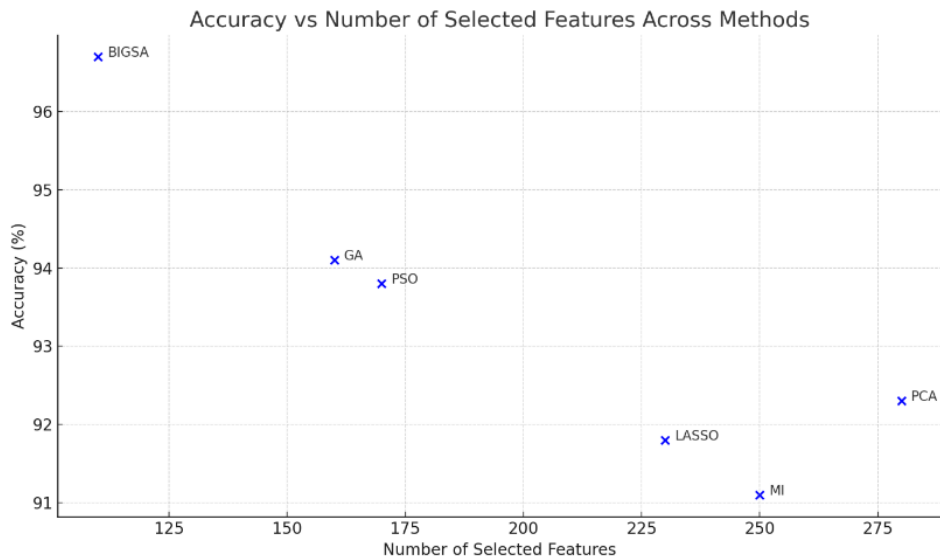


Figure 4: Accuracy vs. number of selected features across feature selection methods.

4.2. HNO

This subsection investigates the effect of varying hidden node counts in the ELM classifier to determine the optimal architecture via HNO. Figure 5 shows classification accuracy across hidden node configurations ranging from 50 to 150. The highest performance—96.7% accuracy—was achieved at 120 hidden nodes.

Beyond this point, performance declines, indicating overfitting and increased variance. Figure 6 supports this finding by depicting trends

across multiple metrics (F1-score, Precision, Recall), all peaking at 120 hidden units. This demonstrates the effectiveness of HNO in balancing complexity and generalization. Statistical analysis using one-way ANOVA confirms the significance of these differences ($p < 0.001$), validating 120 as the optimal hidden layer size.

Table 6 numerically captures these trends, showing the classification performance across various hidden node configurations:

Optimizing Android Malware Detection with Joint Feature Selection and Hidden Node Tuning: A BIGSA-HNO-ELM Framework

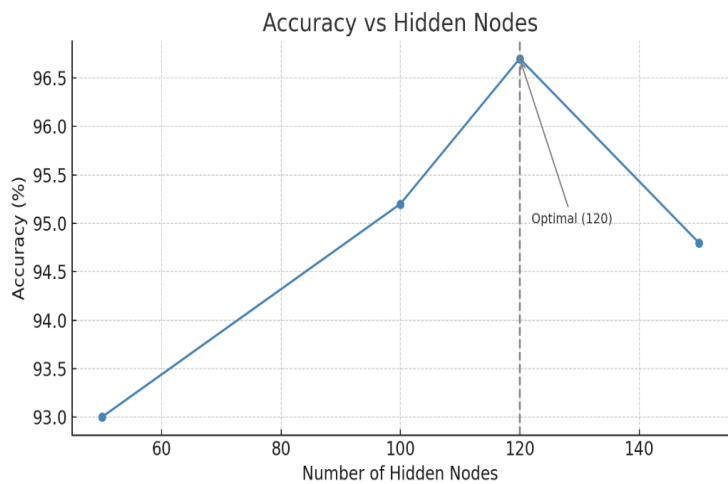


Figure 5: Accuracy vs. Number of Hidden Nodes

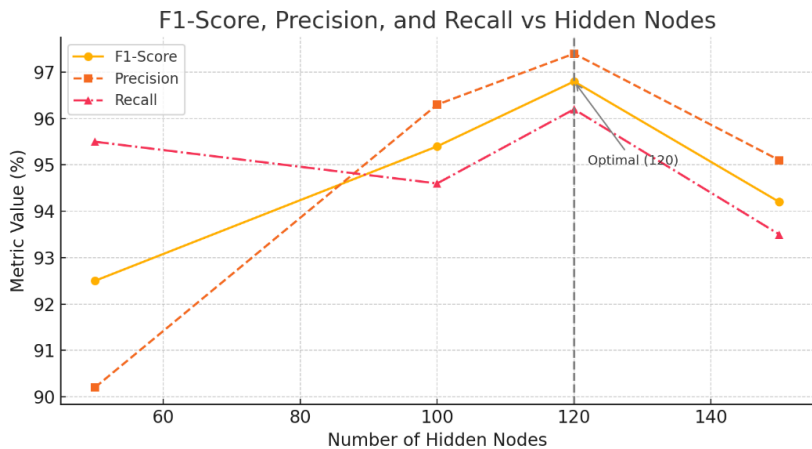


Figure 6: F1-Score, Precision, and Recall vs. Hidden Nodes

Table 6: Performance metrics for varying hidden layer node counts.

Hidden Nodes	Accuracy (%)	F1-Score (%)	Precision (%)	Recall (%)
50	93.0	92.5	90.2	95.5
100	95.2	95.4	96.3	94.6
120	96.7	96.8	97.4	96.2
150	94.8	94.2	95.1	93.5

4.4. Classifier Performance Comparison

Figure 7 compares the proposed BIGSA-HNO-ELM framework with traditional machine learning classifiers (SVM, Random Forest, GA+RF, MLP, XGBoost) and advanced deep learning models (DBN-GRU, LSTM, DBNN). Performance is assessed using Accuracy, Precision, Recall, and F1-Score, with error bars representing ± 1 standard deviation. The proposed BIGSA-HNO-ELM model consistently achieves the highest mean performance across all metrics while exhibiting the lowest variance, demonstrating robust and stable performance. This indicates the proposed model’s superior generalization capability and reliability in practical malware detection scenarios.

Figure 8 provides ROC curve comparisons, clearly showing that the BIGSA-HNO-ELM model achieves the highest area under the curve (AUC = 97%), indicating a substantially superior discriminative capability compared to traditional methods like SVM (AUC = 91%) and XGBoost (AUC = 92%). Statistical evidence from paired t-tests summarized in Table 7 confirms that BIGSA-HNO-ELM significantly outperforms baseline classifiers ($p < 0.01$), reinforcing the statistical reliability of these observed improvements. While deep learning models (DBN-GRU, LSTM, DBNN) perform robustly, they are still outperformed marginally by the BIGSA-HNO-ELM, emphasizing the benefits of jointly optimized feature selection and classifier tuning in malware detection.

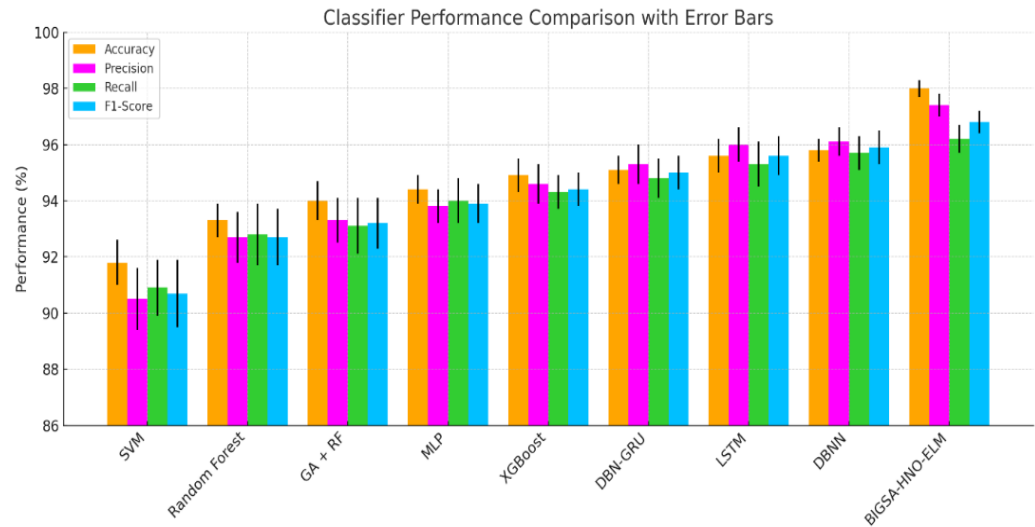


Figure 7 Classifier performance comparison across models with error bars

Optimizing Android Malware Detection with Joint Feature Selection and Hidden Node Tuning: A BIGSA-HNO-ELM Framework

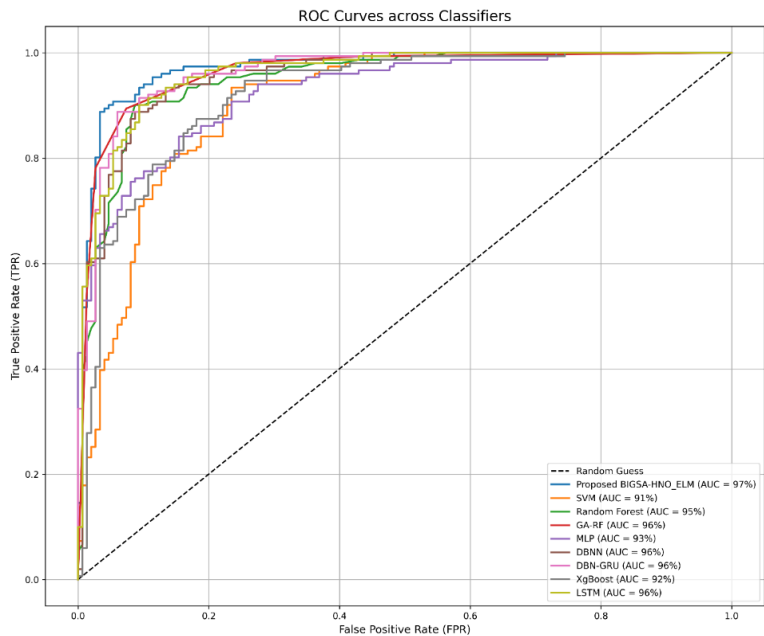


Figure 8: ROC curve comparison of baseline classifiers vs. BIGSA-HNO-ELM

Table 7: Classifier Performance with Statistical Significance

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC (%)	p-value (paired t-test vs BIGSA-HNO-ELM)
SVM	91.8 ± 0.8	90.5 ± 1.1	90.9 ± 1.0	90.7 ± 1.2	91.0 ± 0.9	< 0.01
Random Forest	93.3 ± 0.6	92.7 ± 0.9	92.8 ± 1.1	92.7 ± 1.0	95.0 ± 0.8	< 0.01
GA + RF	94.0 ± 0.7	93.3 ± 0.8	93.1 ± 1.0	93.2 ± 0.9	96.0 ± 0.7	< 0.01
MLP	94.4 ± 0.5	93.8 ± 0.6	94.0 ± 0.8	93.9 ± 0.7	93.0 ± 0.6	< 0.01
XGBoost	94.9 ± 0.6	94.6 ± 0.7	94.3 ± 0.6	94.4 ± 0.6	92.0 ± 0.5	< 0.01
DBN-GRU	95.1 ± 0.5	95.3 ± 0.7	94.8 ± 0.7	95.0 ± 0.6	96.0 ± 0.5	< 0.01
LSTM	95.6 ± 0.6	96.0 ± 0.6	95.3 ± 0.8	95.6 ± 0.7	96.0 ± 0.5	< 0.01
DBNN	95.8 ± 0.4	96.1 ± 0.5	95.7 ± 0.6	95.9 ± 0.6	96.0 ± 0.4	< 0.01
BIGSA-HNO-ELM	98.0 ± 0.3	97.4 ± 0.4	96.2 ± 0.5	96.8 ± 0.4	97.0 ± 0.3	—

4.5. Ablation Study

An ablation study was carried out to rigorously evaluate the individual contributions of the feature selection (BIGSA) and HNO components within the proposed BIGSA-HNO-ELM framework. Four different model variants were tested to prove the influence of each module on detection accuracy:

1. **ELM Only (Baseline):** Untuned classifier using all 462 features, no optimization fixed hidden layer: $L=100$
2. **BIGSA + ELM:** Feature selection only (106 features, 77.0% reduction), fixed $L=100$

3. **ELM + HNO:** Full feature set (462 features); hidden node count optimized via HNO (120 neurons)
4. **BIGSA + HNO + ELM:** Joint feature selection –reduced feature set (106 features) and architecture tuning via HNO(118 neurons)

The evaluations utilized 10 independent stratified 80:20 train-test splits, to ensure consistent comparison. HNO configurations were optimized by employing an internal validation process.

Table 8 summarizes these outcomes, below:

Table 8: Performance comparison of ablated variants (mean $\pm$ std over 10 runs).

Configuration	Features Used	Feature Reduction	Hidden Neurons	Accuracy (%)	Gap vs Full (%)	p-value vs Full
ELM Only (Baseline)	462	0%	Fixed (100)	91.4 $\pm$ 0.8	-5.3%	< 0.001(***)
BIGSA + ELM	106	77%	Fixed (100)	94.2 $\pm$ 0.6	-2.5%	0.003(***)
ELM + HNO	462	0%	Optimized (120 $\pm$ 0)	93.6 $\pm$ 0.7	-3.1%	< 0.001(***)
BIGSA + HNO + ELM	106	77%	Optimized (118 $\pm$ 2)	96.7 $\pm$ 0.5	-	—

*Note: Gap vs Proposed (%) indicates an accuracy difference compared to the full BIGSA-HNO-ELM model. Significance is denoted by asterisks (**, $p < 0.001$). **

Figure 9 demonstrates that the BIGSA-HNO-ELM framework (96.7% $\pm$ 0.5) ominously outperformed all other configurations (***, $p < 0.001$). This validates the complementing benefits of integrating optimized feature

selection with adaptive neural architecture tuning. BIGSA effectively eliminates redundant or less useful features, while HNO guarantees the classifier is properly complex, resulting in improving overall performance.

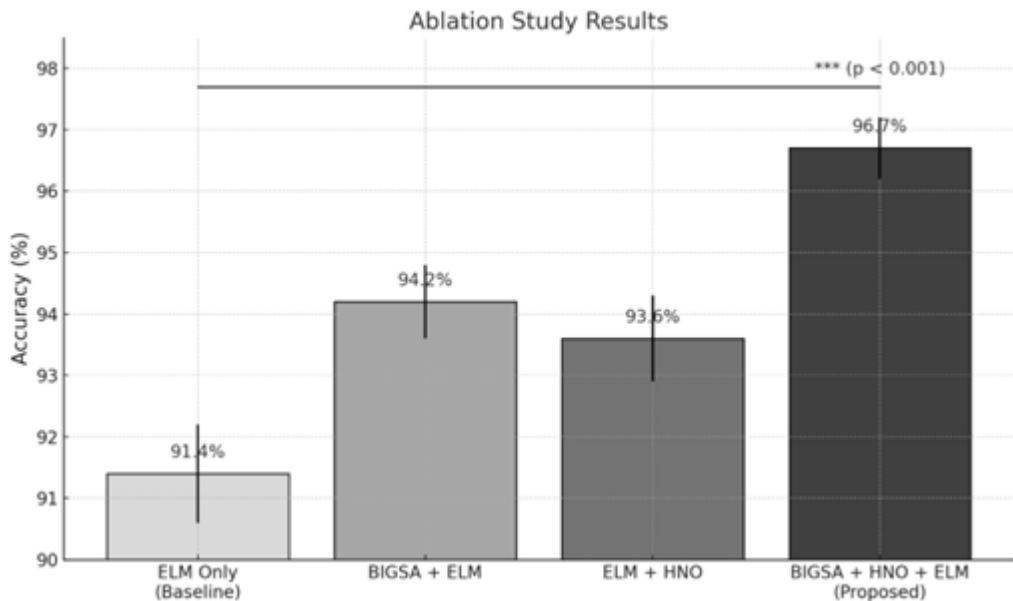


Figure 9: Accuracy comparison of ablated BIGSA-HNO-ELM variants over 10 independent runs.

5. CONCLUSION LIMITATIONS, and DEPLOYMENT IMPLICATIONS

The BIGSA-HNO-ELM mechanism presented in this article combines BIGSA for optimal feature selection with HNO deployed to an ELM for high-performance

Android malware detection. When tested on the KronoDroid dataset, the model validates strong generalization, beating both traditional and deep learning models in all vital metrics and proving superior stability throughout 10-fold cross-validation with low variance in performance metrics. The suggested method attains a 77% reduction in dimensionality while retaining a high detection accuracy of 98.0% and AUC-ROC of 97.0%, making it a competitive solution for lightweight, scalable threat detection.

Despite these results, the study acknowledges limitations such as dependency on a single dataset, lack of adversarial robustness evaluation, and limited interpretability of ELM

decisions. These aspects may impact generalizability across unknown malware variants or evolving obfuscation techniques.

Nonetheless, the reduced computational overhead and high accuracy make this model viable for real-time and resource-constrained deployments, such as mobile devices and embedded systems. Future work will explore cross-dataset validation, integration with explainable AI tools (e.g., SHAP, LIME), and deployment in federated learning environments to support distributed security architectures.

Data availability statement

The dataset use in this study is available publicly and can be download using the link:

<https://www.kaggle.com/datasets/dhoogla/kronodroid-2021/data>

REFERENCES

- [1] G. Stats, "Operating system market share worldwide—statcounter global stats," 2024. [Online]. Available: <https://gs.statcounter.com/os-market-share>.
- [2] D. a. S. M. a. H. M. a. G. H. a. R. K. a. S. C. Arp, "Drebin: Effective and explainable detection of android malware in your pocket.," in *Ndss*, vol. 14, 2014, pp. 23-26.
- [3] A. a. A. O. a. M. M. Mohaisen, "AMAL: high-fidelity, behavior-based automated malware analysis and classification," *computers & security*, vol. 52, pp. 251-266, 2015.
- [4] T. Shishkova, "The mobile malware threat landscape in 2022," *Securelist by Kaspersky*, 2023.
- [5] A. a. M. R. a. D. M. K. a. B. R. a. M. J. Fatima, "Android malware detection using genetic algorithm based optimized feature selection and machine learning," in *2019 42nd International conference on telecommunications and signal processing (TSP)*, IEEE, 2019, pp. 220-223.
- [6] F. a. B. M. a. G. H. M. a. P. F. a. O. L. S. a. G. A. Ceschin, "Fast & furious: On the modelling of malware detection as an evolving data stream," *Expert Systems with Applications*, p. 118590, 2023.
- [7] Y. a. L. J. a. Q. Z. a. R. K. He, "Going Proactive and Explanatory Against Malware Concept Drift," *arXiv e-prints*, pp. arXiv--2405, 2024.
- [8] K. a. X. S. a. X. G. a. Z. M. a. S. D. a. L. H. Liu, "A review of android malware detection approaches based on machine learning," *IEEE access*, vol. 8, pp. 124579-124607, 2020.
- [9] E. a. N.-P. H. a. S. S. Rashedi, "BGSA: binary gravitational search algorithm," *Natural computing*, vol. 9, pp. 727-745, 2010.
- [10] L. a. H. S. a. W. L. a. Z. W. a. Y. J. Zhu, "Feature selection using an improved gravitational search algorithm," *IEEE Access*, vol. 7, pp. 114440-114448, 2019.
- [11] G.-B. a. Z. Q.-Y. a. S. C.-K. Huang, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489--501, 2006.
- [12] A. a. B. H. a. N. S. Guerra-Manzanares, "Kronodroid: Time-based hybrid-featured dataset for effective android malware detection and characterization," *Computers & Security*, vol. 110, p. 102399, 2021.
- [13] J. Demvsar, "Statistical comparisons of classifiers over multiple data sets,"

Journal of Machine learning research, vol. 7, no. Jan, pp. 1-30, 2006.

- [14] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *arXiv preprint arXiv:1811.12808*, 2018.
- [15] S. Seraj, "Permission-based android malware detection using machine learning," 2023.
- [16] E. a. O. L. a. A. P. a. D. C. E. a. R. G. a. S. G. Mariconti, "Mamadroid: Detecting android malware by building markov chains of behavioral models," *arXiv preprint arXiv:1612.04433*, 2016.
- [17] M. a. M. K. W. Abuthawabeh, "Enhanced android malware detection and family classification, using conversation-level network traffic features," *Int. Arab J. Inf. Technol*, vol. 17, no. 4A, pp. 607-614, 2020.
- [18] J. a. L. J. a. Z. A. a. Z. J. a. Y. Z. a. L. Z. a. C. T.-S. Zheng, "MaskDroid: Robust Android Malware Detection with Masked Graph Representations," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 331-343.
- [19] I. a. B. F. a. U.-P. X. a. B. P. G. Santos, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *information Sciences*, vol. 231, pp. 64-82, 2013.
- [20] T. a. G. C. Chen, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785-794.
- [21] T. a. D. Y. a. O. L. a. C. Q. a. W. X. Lu, "Android malware detection based on a hybrid deep learning model," *Security and Communication Networks*, vol. 2020, no. 1, p. 8863617, 2020.
- [22] I. a. B. Y. a. C. A. a. B. Y. Goodfellow, *Deep learning*, vol. 1, MIT press Cambridge, 2016.
- [23] Z. a. L. Y. a. X. Y. Yuan, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114-123, 2016.
- [24] D. a. E.-G. S. a. S. E. Saif, "Deep Belief Networks-based framework for malware detection in Android systems," *Alexandria engineering journal*, vol. 57, no. 4, pp. 4049-4057, 2018.
- [25] P. a. N. G. Sajith, "Intrusion detection system using deep belief network \& particle swarm optimization,"

Wireless Personal Communications, vol. 125, no. 2, pp. 1385-1403, 2022.

B: Statistical Methodology, vol. 58, no. 1, pp. 267-288, 1996.

- [26] S. a. C. L. a. Z. Y. a. X. J. a. Y. P. a. M. J. Peng, "Challenges and trends of android malware detection in the era of deep learning," in *2020 IEEE 8th International Conference on Smart City and Informatization (iSCI)*, IEEE, 2020, pp. 37-43.
- [27] A. a. A. F. a. A. N. a. A. S. a. S. S. Albakri, "Metaheuristics with deep learning model for cybersecurity and android malware detection and classification," *Applied Sciences*, vol. 13, no. 4, p. 2172, 2023.
- [28] H. a. H. W. a. Z. Y. a. L. W. a. W. Y. Huang, "FEdroid: a lightweight and interpretable machine learning-based android malware detection system}," *Cluster Computing*, vol. 28, no. 4, p. 224, 2025.
- [29] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1289-1305, 2003.
- [30] R. a. J. G. H. Kohavi, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273-324, 1997.
- [31] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society Series*
- [32] E.-G. Talbi, "Metaheuristics: from design to implementation}," 2009.
- [33] H. a. S. M. Bostani, "Hybrid of binary gravitational search algorithm and mutual information for feature selection in intrusion detection systems," *Soft computing*, vol. 21, pp. 2307-2324, 2017.
- [34] G. a. B. G. E. a. c. F. V. Kale, "Evolutionary feature selection for machine learning based malware classification," *Engineering Science and Technology, an International Journal*, vol. 56, p. 101762, 2024.
- [35] M. H. a. F. M. a. F. A. a. K. N. Z. Ali, "A hybrid particle swarm optimization-extreme learning machine approach for intrusion detection system," in *2018 IEEE student conference on research and development (SCORED)*, IEEE, 2018, pp. 1-4.
- [36] S. a. G. G. P. Bhagwat, "Android malware detection using hybrid meta-heuristic feature selection and ensemble learning techniques," in *International conference on advances in computing and data sciences*, Springer, 2022, pp. 145-156.
- [37] Y. a. L. M. a. Z. Q. a. Y. T. a. W. J. a. F. Z. a. C. L. Wu, "DroidRL: Feature selection for android malware

- detection with reinforcement learning," *Computers & Security*, vol. 128, p. 103126, 2023.
- [38] D. a. E.-G. S. a. S. E. Saif, "Deep Belief Networks-based framework for malware detection in Android systems," *Alexandria engineering journal*, vol. 57, no. 4, pp. 4049-4057, 2018.
- [39] N. V. a. B. K. W. a. H. L. O. a. K. W. P. Chawla, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321-357, 2002.
- [42] U. V. a. D. V. M. Nikam, "Performance evaluation of machine learning classifiers in malware detection},," vol. 2, no. 2, p. 4578, 2019.
- [40] H.-J. a. O. Y.-S. a. T. A.-H. a. Z. Z. Rong, "A fast pruned-extreme learning machine for classification problem," *Neurocomputing*, vol. 72, no. 1-3, pp. 359-366, 2008.
- [41] B. a. S. T. a. Y. T. a. S. Y. a. N. R. He, "A pruning ensemble model of extreme learning machine with $L_{1/2}$ $L_{1/2}$ regularizer," *Multidimensional Systems and Signal Processing*, vol. 28, pp. 1051-1069, 2017.