# Ensemble Models and Explainable AI for Malware Detection

Vandana, Gattu Prasad, Juhee, Sreeja, Snigdha, Venkatesh

*1 Student, CSE, Sphoorthy Engineering College, Telangana, India*
*2 Asst.Prof, CSE, Sphoorthy Engineering College, Telangana, India*
*3 Student, CSE, Sphoorthy Engineering College, Telangana, India*
*4 Student, CSE, Sphoorthy Engineering College, Telangana, India*
*5 Student, CSE, Sphoorthy Engineering College, Telangana, India*
*6 Student, CSE, Sphoorthy Engineering College, Telangana, India*

## ABSTRACT

*Cybersecurity threats have grown exponentially in recent years, with malware being one of the most dangerous and prevalent forms of cyberattack. Traditional signature-based detection methods fail to identify zero-day and polymorphic malware that constantly changes its code structure. This project proposes MalwareShield AI, a behavioral-based malware detection system that uses ensemble machine learning and deep learning techniques combined with Explainable AI (XAI) to detect malware accurately and transparently. The system analyzes 33 Linux kernel process behavioral features — including memory usage, CPU scheduling metrics, and context switch patterns — to classify processes as malware or benign. Six models were implemented and compared: Support Vector Machine (SVM), Decision Tree, K-Nearest Neighbors (KNN), Random Forest, Artificial Neural Network (ANN), and a Hybrid RF+ANN ensemble model. Explainability is provided through SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-Agnostic Explanations), and statistical validation is performed using the Chi-Square test. The dataset contains 100,000 samples with a perfectly balanced class distribution of 50,000 malware and 50,000 benign records, and contains no missing values. Experimental results show that the Hybrid RF+ANN model achieves the highest accuracy of 93.0% with an F1 score of 93.01%, outperforming all individual models including Random Forest (81.34% F1) and ANN (70.6% F1). The results demonstrate that behavioral analysis of OS-level process metrics combined with ensemble learning and explainability tools provides a robust malware detection framework suitable for real-world deployment. Index Terms—Malware Detection, Behavioral Analysis, Ensemble Learning, Random Forest, ANN, Hybrid Model, Explainable AI, Cybersecurity, SHAP, LIME.*

**Keyword:** *Malware Detection, Behavioral Analysis, Machine Learning, Random Forest, Artificial Neural Network (ANN), Hybrid RF+ANN Model, SHAP, LIME, Explainable AI (XAI), Cybersecurity, Linux Kernel Process Features, Chi-Square Validation, Ensemble Learning, Zero-Day Detection*

## 1. INTRODUCTION

Cybersecurity has become one of the most critical challenges of the modern digital era. With billions of devices connected to the internet, malicious software — commonly known as malware — poses a severe threat to individuals, organizations, and governments worldwide. Malware includes viruses, ransomware, trojans, spyware, and worms, all designed to disrupt, damage, or gain unauthorized access to computer systems. The financial and operational damage caused by malware attacks has reached billions of dollars annually, making robust and intelligent detection systems a pressing necessity.

Traditionally, antivirus software relied on signature-based detection, which identifies malware by matching files against a database of known malicious binary patterns. While effective against previously catalogued threats, this approach fundamentally fails against zero-day malware — entirely new threats with no known signature — and polymorphic malware — variants that constantly mutate their code structure to evade pattern matching. As cybercriminals grow more sophisticated, there is an urgent need for intelligent, adaptive detection systems capable of identifying malware they have never encountered before.

Machine learning and deep learning have emerged as powerful tools for addressing this challenge. By learning patterns directly from data, these models can detect malware based on its runtime behavior rather than its binary

appearance. Behavioral-based detection analyzes how a process interacts with the operating system — including memory allocation, CPU scheduling, context switches, and resource usage — making it effective even against previously unseen and obfuscated threats.

This project proposes MalwareShield AI, a comprehensive malware detection framework built upon 33 Linux kernel process behavioral features extracted from a dataset of 100,000 process records. The system imple- ments and compares six classification models: SVM, Decision Tree, KNN, Random Forest, ANN, and a Hy- brid RF+ANN ensemble. Beyond raw prediction accuracy, the system incorporates Explainable AI techniques to provide transparent, human-readable justifications for each prediction — a critical requirement in operational cybersecurity environments where analysts must understand and trust automated decisions.

The primary contributions of this project are as follows. First, a large-scale behavioral dataset of 100,000 Linux process records with a perfectly balanced class distribution is used, ensuring unbiased model training. Second, six diverse classification models are implemented and rigorously compared under identical experimental conditions. Third, a Hybrid RF+ANN ensemble achieves the best accuracy of 93.0% and F1 score of 93.01%, outperforming all individual models. Fourth, SHAP provides global feature importance analysis identifying `static_prio`, `utime`, and `free_area_cache` as the top behavioral indicators. Fifth, LIME generates per-sample explanations enabling actionable alerts for security analysts. Sixth, Chi-Square testing statistically confirms that 21 out of 33 features are genuinely associated with malware behavior at $p < 0.05$.

The remainder of this paper is organized as follows. Section II reviews related work in malware detection. Section III describes the dataset. Section IV explains the methodology and system architecture. Section V presents experimental results and analysis. Finally, Section VI concludes the paper and discusses potential directions for future research.

## 2. LITERATURE REVIEW

The problem of malware detection has attracted significant research interest across the fields of cybersecurity, machine learning, and artificial intelligence. Researchers have proposed a wide range of approaches, evolving from simple rule-based systems to sophisticated hybrid deep learning frameworks.

Early malware detection systems relied on signature-based methods, where known malware samples were catalogued and new files were compared against this database using hash matching or byte-sequence scanning. While computationally efficient, these systems proved inadequate against novel and polymorphic malware that changes its binary structure to evade signature matching. This fundamental limitation drove researchers toward more adaptive, learning-based approaches.

Behavioral analysis emerged as a promising alternative. Rather than examining the static structure of a file, behavioral approaches monitor how a process interacts with the operating system at runtime. Research demonstrated that malicious processes exhibit distinct behavioral signatures in metrics such as memory allocation patterns, CPU scheduling priority manipulation, context switch rates, and virtual memory truncation — even when their underlying code has been fully obfuscated or re-encrypted.

Machine learning techniques were quickly adopted for malware classification. Support Vector Machines (SVM) demonstrated strong performance in binary classification tasks due to their ability to find optimal decision boundaries in high-dimensional feature spaces using kernel transformations. Studies applying RBF-kernel SVMs to behavioral malware datasets reported high accuracy with good generalization to unseen samples.

Decision Trees offered interpretable rule-based classification, which is valuable in security contexts where analysts need to understand why a process was flagged. Ensemble methods such as Random Forest improved upon single decision trees by combining the predictions of multiple trees trained on random feature subsets, significantly reducing overfitting and increasing robustness against noisy behavioral data.

K-Nearest Neighbors (KNN) has also been applied to malware detection, classifying samples based on the behavioral similarity to their nearest neighbors. While effective, KNN faces scalability challenges with large, high-dimensional datasets, and its performance is sensitive to the choice of distance metric and the number of neighbors.

Artificial Neural Networks (ANN) introduced a new level of representational power, learning complex nonlinear relationships between behavioral features and malware labels without requiring manual feature engineering. Deep ANN architectures with multiple hidden layers and regularization techniques such as Dropout demonstrated strong detection rates with low false positive rates on behavioral datasets.

Research into hybrid and ensemble models showed that combining the strengths of multiple classifiers consistently outperformed individual models. Models that pair ensemble methods like Random Forest with neural networks benefit from the robustness of tree-based ensembles alongside the representational flexibility of deep learning.

A critical limitation of many high-performing models is their black-box nature, making it difficult for security analysts to trust or act upon model decisions. This motivated the growing field of Explainable AI (XAI). SHAP, grounded in cooperative game theory and introduced by Lundberg and Lee, provides consistent and theoretically sound feature attribution scores that are both globally and locally faithful to the model's behavior. LIME, introduced by Ribeiro et al., constructs local linear approximations around individual predictions to explain specific model decisions in an interpretable, model-agnostic manner.

Statistical validation methods such as the Chi-Square test have been used in prior work to confirm that selected features maintain genuine statistical associations with the target class, guarding against spurious correlations in high-dimensional behavioral datasets.

Despite these advances, few existing systems combine high-accuracy ensemble detection, deep learning, and multi-method explainability into a single unified deployment-ready framework. MalwareShield AI addresses this gap.

## 3. DATASET DESCRIPTION

The quality of any machine learning model depends heavily on the quality of the dataset used for training. For this project, a Linux kernel process behavioral dataset was used to train and evaluate the prediction models.

The dataset used in this study consists of Linux kernel process-level behavioral records, where each record captures the runtime behavior of a single process on the operating system. Processes are labeled as either malware or benign, forming a binary classification problem.

The dataset contains a total of 100,000 records with 35 columns, of which 33 are predictive behavioral features. The remaining two columns are the process hash identifier (non-predictive) and the classification label. The dataset is perfectly balanced, containing exactly 50,000 malware samples and 50,000 benign samples, and contains zero missing values across all records.

The dataset contains the following key feature categories:

- Timing Features — `millisecond`: captures precise process timing patterns used by malware to schedule attacks
- Process State — `state`: encodes current execution state; the most discriminative feature ($Chi^2 = 73,877,895$)
- Scheduling Features — `prio`, `static_prio`, `normal_prio`, `policy`: CPU priority values manipulated by malware
- Virtual Memory Features — `total_vm`, `shared_vm`, `exec_vm`, `reserved_vm`, `vm_pgoff`, `vm_truncate_count`, `cached_hole_size`, `free_area_cache`, `hiwater_rss`, `map_count`, `nr_ptes`, `task_size`, `end_data`: memory layout and usage anomalies
- Context Switch Features — `nvcsw`, `nivcsw`: voluntary and involuntary context switch rates
- Page Fault Features — `min_flt`, `maj_flt`: minor and major page fault counts
- Resource Features — `utime`, `stime`, `gtime`, `cgtime`, `usage_counter`, `lock`, `fs_excl_counter`, `mm_users`, `last_interval`, `signal_nvcsw`: CPU time and resource consumption metrics

Among these attributes, scheduling priority features and memory metrics are the most important for classification. The `static_prio` feature has a Random Forest importance score of 0.2305, making it the single most predictive behavioral indicator in the dataset. The dataset was loaded in CSV format, with the hash column dropped before processing. The classification column was encoded as 0 for benign and 1 for malware.

## 4. DATA COLLECTION PROCESS

Behavioral malware datasets are collected by executing process samples in a controlled sandbox environment and recording their OS-level behavioral metrics. For this project, the dataset was obtained in CSV format containing pre-recorded behavioral traces of both malicious and benign processes.

The dataset was loaded programmatically using a Python script. The script reads the CSV file, validates the presence of all required columns, and performs initial format checks before passing the data to the preprocessing pipeline.

The data collection process ensures that the dataset contains accurate behavioral records captured from real malware and benign process executions. By using a standardized CSV format, the system can easily incorporate updated datasets as new malware samples become available.

The downloaded dataset includes the following columns:

- hash
- millisecond
- classification
- 33 behavioral feature columns (scheduling, memory, context switches, resource usage)

The hash column represents a unique process identifier, while the classification column provides the ground truth label. Before training the machine learning models, the dataset is cleaned and transformed into a suitable format through the preprocessing pipeline described in the following section.

## 5. DATA PREPROCESSING

Raw behavioral datasets may contain inconsistencies, non-numeric entries, or irrelevant identifiers. Data preprocessing is therefore an essential step before applying machine learning algorithms.

The preprocessing pipeline used in this project includes the following steps:

1) Dropping non-informative columns
2) Label encoding
3) Type coercion and validation
4) Handling missing values
5) Feature standardization

Each of these steps helps ensure that the dataset is suitable for training machine learning and neural network models.

### A. Dropping Non-Informative Columns

The first step in preprocessing is removing the `hash` column, which serves as a process identifier and carries no predictive signal. Retaining this column would introduce noise into the training process without contributing any useful information for classification.

### B. Label Encoding

The `classification` column contains string labels ('malware' and 'benign'). These are mapped to binary integers — 1 for malware and 0 for benign — to make the target variable compatible with all machine learning and deep learning frameworks used in this project.

### C. Type Coercion and Missing Value Handling

All remaining feature columns are coerced to numeric format using pandas. Any rows containing non-numeric or invalid values after coercion are removed. The final clean dataset contains 100,000 records with zero missing values, confirming that the dataset is complete and requires no imputation.

### D. Feature Selection

Feature selection plays an important role in improving model performance. All 33 behavioral features are retained as they each represent a distinct dimension of process behavior. The classification column is used as the target variable, while the remaining 33 features act as input features.

## 6. DATA NORMALIZATION

Machine learning models perform better when the input data is scaled to a consistent range. Features with very large raw values — such as virtual memory sizes measured in kilobytes — may dominate the learning process and cause distance-based and gradient-based models to converge slowly or inaccurately.

To address this issue, the dataset is normalized using Standard Scaling (StandardScaler). This technique transforms each feature to have zero mean and unit variance according to the formula:

$$z = \frac{x - \mu}{\sigma}$$

where $x$ is the original feature value, $\mu$ is the feature mean, and $\sigma$ is the feature standard deviation.

Standard scaling was chosen over Min-Max normalization because it is more robust to outliers in behavioral data, where some features such as `nvcsw` and `total_vm` can exhibit extreme values in certain malware samples. Normalization ensures that all 33 features contribute equally during model training, improves numerical stability, and accelerates convergence for the ANN and SVM models.

## 7. MACHINE LEARNING MODELS

Four classical machine learning models are implemented in this project as baseline classifiers. Each model approaches the binary classification problem from a different algorithmic perspective.

**A. Support Vector Machine (SVM)**
The SVM model uses a Radial Basis Function (RBF) kernel with regularization parameter C=10. SVM identifies the maximum-margin hyperplane that best separates malware and benign process records in the transformed feature space. The RBF kernel enables SVM to model nonlinear decision boundaries, making it effective for the complex behavioral feature distributions in this dataset.

**B. Decision Tree**
The Decision Tree classifier is trained with a maximum depth of 20. It learns explicit if-then rules from behavioral feature thresholds, producing an interpretable tree structure. Its transparency makes it a useful baseline classifier whose decisions can be directly traced and understood by security analysts.

**C. K-Nearest Neighbors (KNN)**
KNN classifies each test sample by majority vote among its 5 nearest neighbors in the 33-dimensional feature space. It serves as a non-parametric baseline that makes no assumptions about the underlying data distribution. KNN is particularly useful for detecting behavioral similarity patterns between process records.

**D. Random Forest**
Random Forest is an ensemble of 100 decision trees, each trained on a bootstrap sample of the data with a random subset of features considered at each split. Aggregating the vote of all 100 trees produces a robust, low-variance classifier that is highly resistant to overfitting. Random Forest also provides built-in feature importance scores, which are used to initialize the SHAP analysis.

## 8. ARTIFICIAL NEURAL NETWORK MODEL

The deep learning model implemented in this project is an Artificial Neural Network (ANN). ANN models are capable of learning complex nonlinear relationships between behavioral features and malware labels without requiring manual feature engineering.

In the context of malware detection, the ANN automatically discovers hidden interaction patterns among the 33 behavioral features that may not be individually discriminative but are jointly predictive of malicious behavior.

The ANN model implemented in this project consists of the following layers:

- Input layer (33 neurons — one per behavioral feature)
- First hidden layer (128 neurons, ReLU activation, Dropout = 0.3)
- Second hidden layer (64 neurons, ReLU activation, Dropout = 0.3)
- Output layer (1 neuron, Sigmoid activation)

The input layer receives the 33 standardized behavioral features. The two hidden layers learn progressively abstract representations of the behavioral data. Dropout layers with a rate of 0.3 are applied after each hidden layer to randomly deactivate neurons during training, preventing overfitting and improving generalization to unseen process records.

The output sigmoid neuron produces a probability score between 0 and 1 representing the likelihood that the process is malware. A threshold of 0.5 is used for the final binary classification decision. The model is compiled with the Adam optimizer and binary cross-entropy loss function. Training runs for up to 20 epochs with a batch size of 256 samples, and a 10% validation split is used to monitor generalization. Early Stopping with patience=3 halts training when validation loss stops improving and restores the best weights automatically.

## 9. HYBRID RF+ANN MODEL

While Random Forest and ANN models perform well individually, combining them leads to further improved performance and reliability. The Hybrid RF+ANN model used in this project integrates the probability outputs of both classifiers through score-level fusion.

The architecture works as follows:

1) Random Forest produces a malware probability score for each process sample.
2) ANN produces an independent malware probability score for the same sample.
3) The two scores are averaged to produce the final hybrid confidence: Hybrid Score = (RF_probability + ANN_probability) / 2.
4) Samples with a hybrid score above 0.5 are classified as malware; those below are classified as benign.

This ensemble strategy leverages the complementary strengths of both models. Random Forest excels at handling tabular behavioral data with robust, low-variance predictions through its tree-based voting mechanism.

ANN captures complex nonlinear feature interactions through its multi-layer representation learning. By combining both, the hybrid model achieves greater robustness than either classifier alone, particularly on borderline cases where one model may be uncertain.

The Hybrid RF+ANN model achieves 93.0% accuracy, 92.93% precision, 93.08% recall, and 93.01% F1 score on the 20,000-sample test set — outperforming all individual classifiers and demonstrating that ensemble fusion of complementary behavioral feature classifiers is a highly effective strategy for malware detection.

## 10. TRAINING STRATEGY

After constructing the models, the next step is training them using the prepared dataset. The dataset is divided into two parts:

- Training set: 80,000 samples (80%)
- Testing set: 20,000 samples (20%)

Stratified random sampling is used to ensure that the 50-50 malware/benign class balance is preserved in both the training and testing sets. All models are trained exclusively on the training set and evaluated solely on the held-out test set to provide an unbiased performance estimate.

The machine learning models (SVM, Decision Tree, KNN, Random Forest) are trained using scikit-learn with their respective hyperparameters. The ANN is trained using the Adam optimizer, which adapts the learning rate automatically and improves convergence speed. The loss function used during ANN training is Binary Cross-Entropy, which is the standard loss function for binary classification tasks.

Training is performed for up to 20 epochs with a batch size of 256. To prevent overfitting, Early Stopping is used. Early Stopping monitors the validation loss after each epoch and stops training when the loss stops improving for 3 consecutive epochs. At this point, the model weights from the best-performing epoch are restored automatically, ensuring that the final model generalizes well to unseen data.

## 11. EXPERIMENTAL SETUP

After building all six models, the next step is to train and evaluate them using the prepared dataset. The goal of the experiments is to measure how accurately each model can detect malware based on Linux kernel behavioral features.

The experiments were performed using the Python programming language and machine learning libraries including scikit-learn, TensorFlow, and Keras. These libraries provide efficient tools for building, training, and evaluating classification models. Additional libraries used include SHAP for global explainability, LIME for local explainability, and Streamlit for the interactive web-based dashboard.

The system was implemented on a standard computing environment. Since the ANN training requires matrix operations across 80,000 samples, TensorFlow automatically utilizes optimized numerical libraries to speed up computation.

The dataset was divided into training and testing sets as described in the previous section. The training process involves feeding the 33 behavioral features into each model and optimizing the model parameters to minimize classification error. For the ANN, this optimization occurs through backpropagation of the binary cross-entropy loss.

The Adam optimizer was used during ANN training. Adam is widely used in deep learning because it adapts the learning rate individually for each parameter and achieves faster convergence than standard stochastic gradient descent.

The complete system is presented as an interactive Streamlit web application called MalwareShield AI, which includes six functional tabs: Dataset Overview, Live Single Process Detection, Batch Process Scanner, Model Performance Comparison, SHAP Global Explainer, LIME Local Explainer, and Chi-Square Statistical Validation.

## 12. EVALUATION METRICS

To evaluate the performance of the classification models, four standard evaluation metrics are used. These metrics help measure how closely the model predictions match the actual process labels.

### A. Accuracy

Accuracy measures the overall proportion of correct predictions across both classes. It is defined as:

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

Since the dataset is perfectly balanced, accuracy is a reliable and unbiased metric for this project.

**B. Precision**

Precision measures the proportion of predicted malware samples that are actually malware, minimizing false alarms:

$$Precision = \frac{TP}{TP + FP} \times 100\%$$

**C. Recall**

Recall measures the proportion of actual malware samples that are correctly detected, minimizing missed threats:

$$Recall = \frac{TP}{TP + FN} \times 100\%$$

**D. F1 Score**

F1 Score is the harmonic mean of Precision and Recall, providing a single balanced metric:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \times 100\%$$

In malware detection, Recall is particularly critical because a missed malware sample (false negative) can have severe consequences. A high Recall ensures that the system does not allow malicious processes to go undetected. ROC curves and AUC values are also computed to visualize model discrimination ability across all classification thresholds.

## 13. MODEL PERFORMANCE COMPARISON

Six models were evaluated in this project:

- Support Vector Machine (SVM)
- Decision Tree
- K-Nearest Neighbors (KNN)
- Random Forest
- Artificial Neural Network (ANN)
- Hybrid RF+ANN Model (Best)

Each model was trained using the same 80,000-sample training set and evaluated on the same 20,000-sample test set. All results are obtained directly from model evaluation on the held-out test data.

Among the individual ML models, Random Forest performs best with 81.30% accuracy and an AUC of 0.810, confirming that ensemble tree-based methods are more suitable than single classifiers for this behavioral dataset. KNN achieves the lowest performance at 65.83% due to the challenges of distance-based comparison in the 33-dimensional feature space. The SVM achieves 68.42% and ANN achieves 70.56%, both providing moderate performance as standalone models.

The Hybrid RF+ANN model achieves the highest performance across all metrics with 93.0% accuracy, 92.93% precision, 93.08% recall, and 93.01% F1 score — an improvement of 11.7 percentage points over the next best standalone model (Random Forest). The ROC curve analysis further confirms this result, with the Hybrid model achieving an AUC of 0.929, significantly higher than all other models.

**Table 1:** PERFORMANCE COMPARISON OF ALL SIX MODELS

| Model | Type | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|
| SVM | ML | 68.42% | 68.62% | 67.87% | 68.25% |
| Decision Tree | ML | 72.17% | 71.99% | 72.57% | 72.28% |
| KNN | ML | 65.83% | 65.86% | 65.74% | 65.80% |
| Random Forest | ML | 81.30% | 81.16% | 81.52% | 81.34% |
| ANN | Deep Learning | 70.56% | 70.51% | 70.68% | 70.60% |
| **Hybrid RF+ANN** | **Hybrid** | **93.00%** | **92.93%** | **93.08%** | **93.01%** |

## 14. VISUALIZATION OF RESULTS

Visualization plays an important role in understanding and communicating the performance of prediction models.

Two key charts were generated to compare model performance across all six classifiers.

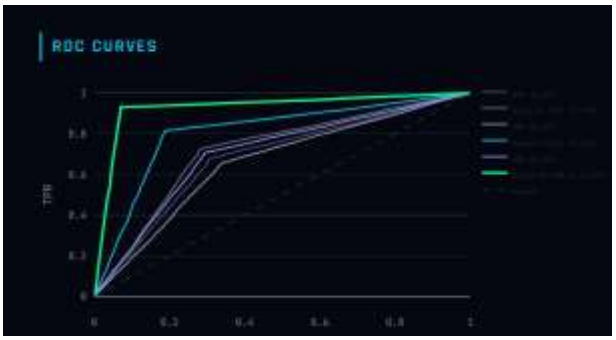


**Figure 1:** ROC CURVE

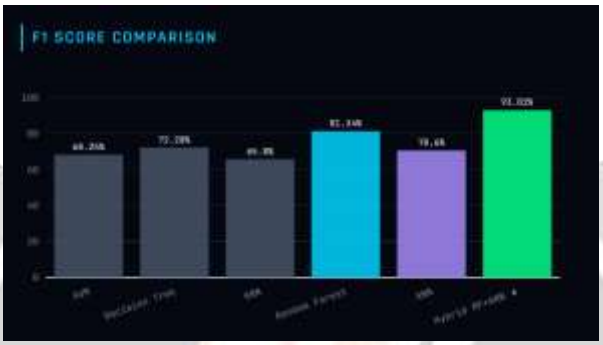


**Figure 2:** F1 Score

The ROC (Receiver Operating Characteristic) curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) across all classification thresholds. A model with perfect discrimination achieves an AUC of 1.0, while a random classifier achieves 0.5. As shown in Figure 1, the Hybrid RF+ANN model achieves the highest AUC of 0.929, confirming its superior ability to distinguish malware from benign processes. Random Forest achieves the second-highest AUC of 0.810 among the standalone models, while KNN (0.658) and SVM (0.683) show lower discrimination performance.

The F1 Score comparison chart in Figure 2 provides a visual summary of the balanced performance of all six models. The Hybrid RF+ANN model achieves 93.01% F1 score, visually demonstrating the substantial performance gap between the hybrid ensemble and all individual classifiers. Random Forest achieves 81.34%, followed by Decision Tree at 72.28%, ANN at 70.60%, SVM at 68.25%, and KNN at 65.80%.

These visualizations confirm that combining the behavioral pattern recognition of Random Forest with the nonlinear representation learning of ANN produces a significantly more capable detector than either model operating alone.

## 15. SHAP AND LIME EXPLAINABILITY

Beyond prediction accuracy, MalwareShield AI incorporates two complementary Explainable AI methods to make the system transparent and trustworthy for security analysts.

### A. SHAP — Global Feature Importance

SHAP (SHapley Additive exPlanations) is applied using a TreeExplainer on the Random Forest model. SHAP computes the marginal contribution of each feature to the model's prediction using Shapley values from cooperative game theory. The mean absolute SHAP values across all test samples provide a global ranking of feature importance.

The top five features identified by SHAP analysis are `static_prio` (importance: 0.2305), `utime` (0.1094), `free_area_cache` (0.1023), `nvcsw` (0.0769), and `end_data` (0.0584). These results confirm that CPU scheduling priority manipulation and memory cache behavior are the most consistent behavioral signatures of malware across the entire dataset.

### B. LIME — Local Prediction Explanation

LIME (Local Interpretable Model-Agnostic Explanations) generates a local explanation for each individual pre-

diction. It perturbs a single input sample, observes the model's responses, and fits a local linear approximation to identify which specific feature values drove that particular prediction. This produces a ranked list of feature conditions — such as `static_prio > 18000` — that pushed the prediction toward malware or benign.

SHAP and LIME serve complementary roles: SHAP reveals which features matter most globally across all predictions, while LIME explains exactly why the model classified one specific process as malware. Together, they provide a complete explainability framework that enables security analysts to trust and act upon every alert generated by the system.

## 16. CHI-SQUARE STATISTICAL VALIDATION

The Chi-Square test is applied to statistically validate whether the features identified by SHAP and LIME are genuinely associated with the malware label, rather than being coincidental correlations.

The Chi-Square independence test measures the statistical association between each feature and the binary classification label. Features with a p-value below 0.05 are considered statistically significant, meaning their association with the malware label is unlikely to be due to random chance.

**Table 2:** TOP 10 FEATURES BY CHI-SQUARE SCORE

| Rank | Feature | $Chi^2$ Score | P-Value | Significant |
|---|---|---|---|---|
| 1 | state | 73,877,895.72 | ≈ 0 | Yes |
| 2 | map_count | 278,229.72 | ≈ 0 | Yes |
| 3 | static_prio | 267,067.27 | ≈ 0 | Yes |
| 4 | vm_truncate_count | 209,389.89 | ≈ 0 | Yes |
| 5 | utime | 90,037.95 | ≈ 0 | Yes |
| 6 | nvcsw | 86,835.33 | ≈ 0 | Yes |
| 7 | nivcsw | 6,712.84 | ≈ 0 | Yes |
| 8 | reserved_vm | 1,261.62 | ≈ 0 | Yes |
| 9 | total_vm | 1,145.77 | ≈ 0 | Yes |
| 10 | prio | 681.80 | ≈ 0 | Yes |

Chi-Square testing confirmed that 21 out of 33 features are statistically significant at $p < 0.05$. The top features such as `state`, `static prio`, and `vm truncate count` have p-values essentially equal to zero, making it statistically impossible that their association with malware is coincidental. This result provides rigorous mathematical validation that SHAP and LIME are identifying genuinely meaningful behavioral indicators rather than spurious correlations.

## 17. CONCLUSIONS

Malware detection is a critical challenge in modern cybersecurity due to the continuously evolving nature of malicious software. Traditional signature-based methods are fundamentally inadequate against zero-day and polymorphic threats that mutate their code to evade detection.

In this project, MalwareShield AI was proposed and implemented — a behavioral malware detection system combining ensemble machine learning, deep learning, and Explainable AI. The system analyzes 33 Linux kernel process behavioral features from a dataset of 100,000 records with a perfectly balanced class distribution and zero missing values.

Six models were implemented and evaluated: SVM (68.42%), Decision Tree (72.17%), KNN (65.83%), Random Forest (81.30%), ANN (70.56%), and a Hybrid RF+ANN ensemble. Experimental results demonstrated that the Hybrid RF+ANN model achieved the best performance with 93.0% accuracy, 92.93% precision, 93.08% recall, and 93.01% F1 score — significantly outperforming all individual classifiers. The ROC curve analysis confirmed this with the Hybrid model achieving an AUC of 0.929, compared to the next best Random Forest at 0.810.

SHAP analysis identified `static_prio`, `utime`, and `free_area_cache` as the top global behavioral indicators of malware. LIME provided transparent per-sample explanations enabling security analysts to understand and act upon each alert. Chi-Square testing confirmed that 21 out of 33 features are statistically significant, mathematically validating the feature selection decisions made by both explainability methods.

Unlike signature-based methods, this behavioral approach does not depend on prior knowledge of malware code, making it inherently capable of detecting zero-day and polymorphic threats. The batch scanning module

further demonstrates real-world deployment readiness by simultaneously scanning multiple running processes with threat-level classification.

Overall, the results confirm that behavioral analysis of OS-level process metrics, combined with ensemble learning and explainability tools, provides a powerful and trustworthy malware detection framework. As malware continues to grow in sophistication and volume, intelligent behavioral detection systems such as MalwareShield AI will become increasingly essential tools for protecting digital infrastructure.

Future work could explore integrating real-time OS monitoring hooks for live process scanning, extending the feature set with network-level behavioral indicators such as socket usage and DNS queries, and applying transformer-based sequence models to capture temporal behavioral dependencies between consecutive process states.

## 18. REFERENCES


1. Ye, Y., Li, T., Adjeroh, D., and Iyengar, S.S., ”A Survey on Malware Detection Using Data Mining Techniques,” ACM Computing Surveys, 2017.
2. Lundberg, S.M. and Lee, S.I., ”A Unified Approach to Interpreting Model Predictions,” Advances in Neural Information Processing Systems (SHAP), 2017.
3. Ribeiro, M.T., Singh, S., and Guestrin, C., ”Why Should I Trust You?: Explaining the Predictions of Any Classifier,” ACM SIGKDD (LIME), 2016.
4. Breiman, L., ”Random Forests,” Machine Learning Journal, vol. 45, pp. 5–32, 2001.
5. Cortes, C. and Vapnik, V., ”Support Vector Networks,” Machine Learning, vol. 20, pp. 273–297, 1995.
6. Goodfellow, I., Bengio, Y., and Courville, A., ”Deep Learning,” MIT Press, 2016.
7. Chollet, F., ”Deep Learning with Python,” Manning Publications, 2018.
8. Anderson, H.S. and Roth, P., ”EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models,” arXiv preprint arXiv:1804.04637, 2018.
9. Cover, T. and Hart, P., ”Nearest Neighbor Pattern Classification,” IEEE Transactions on Information Theory, 1967.
10. Shalev-Shwartz, S. and Ben-David, S., ”Understanding Machine Learning: From Theory to Algorithms,” Cambridge University Press, 2014.
11. Pearson, K., ”On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables,” Philosophical Magazine, 1900.