**Malware Family Classification with Machine Learning**

**Team Members: Te-Jou Hou, Meng-Hsuan Ho**

|  |  |
| --- | --- |
| Name | Contribution |
| Te-Jou Hou | import data, data preprocessing (excluding the merging of the target variable), CNN model |
| Meng-Hsuan Ho | merge target value, Decision Tree/ Random Forest model, existing solution investigation |

# Introduction

Different malware families exhibit distinct behavioral patterns, such as variations in API usage, obfuscation techniques, payload delivery methods, and persistence mechanisms. Accurately classifying these malware families is the central goal of our research, as it enables security systems to move beyond generic detection and toward a deeper understanding of the threat landscape. By learning to recognize family-specific traits, machine learning models can not only improve detection accuracy but also assist in attributing attacks, generating targeted signatures, and designing more effective remediation strategies. This classification process plays a critical role in modern cybersecurity, where timely and precise identification of threats is essential for incident response, malware triage, and proactive threat hunting. For instance, if a malware family is characterized by distinct API usage patterns or code structures, accurately assigning a new sample to that family can enable automated systems to generate corresponding detection signatures, such as YARA or Snort rules. This automated signature generation facilitates broader and more consistent detection across diverse security platforms, while simultaneously reducing the need for manual analyst intervention. As a result, it streamlines the security workflow and contributes to improved detection coverage and accuracy.

# Dataset Description

## Data Overview:

This study leverages the EMBER dataset [EMBER, <https://github.com/elastic/ember>], a comprehensive open-source malware classification benchmark containing approximately 3.28 GB of binary files. The dataset includes 158,000 labeled training samples (63,000 malicious, 52,000 benign, and 42,000 unlabeled) and 200,000 test samples evenly split between malicious and benign classes.

Each sample is associated with various static features that support robust malware analysis. These include cryptographic hashes (SHA256/MD5) for unique identification; a timestamp feature (appeared) indicating the year the sample first surfaced; and a “label” field denoting whether the sample is malicious (1), benign (0), or unlabeled (-1). Additionally, the “avclass” attribute provides a malware family name derived from AVClass, such as installmonster, zusy, or fareit, which facilitates family-level classification.

Feature representations include low-level structural indicators such as “histogram” (byte frequency distributions) and “byteentropy” (entropy metrics used to detect packing or obfuscation). Higher-level static attributes are also included, such as “strings” (extracted ASCII and Unicode strings), “general” metadata (e.g., file size, executable type), and “header” information (e.g., compile timestamp, machine type). Further structural components include section information (e.g., .text, .data, .rsrc), imports (e.g., API calls like CreateFile from “kernel32.dll”), exports (for DLL-based files), and “datadirectories” (e.g., import/export/resource tables).

The target variable in this study involves mapping AVClass-generated family labels to broader malware categories, including Trojan, Ransomware, Worm, Spyware, Remote Access Trojan (RAT), Downloader, Adware, Botnet, Potentially Unwanted Applications (PUA), Virus, Backdoor, and Generic malware. This hierarchical classification enables a deeper behavioral understanding of malware and provides a foundation for family-level detection and response strategies.

## Preprocessing Steps:

We applied a series of data preprocessing steps to ensure model compatibility, reduce noise, and improve generalization in malware family classification.

### Feature Selection

### To address issues related to high dimensionality, such as increased computational costs and potential overfitting, we evaluated the relevance of each feature with assistance from ChatGPT and retained only those beneficial for malware classification. The selected features include:

* + **General**: Basic file attributes (e.g., file size, packing). Low-dimensional yet highly informative.
  + **Header**: PE header fields such as machine type and compile timestamp, reflecting malware origin.
  + **Histogram**: Byte frequency distribution (first 50–100 of 256 dimensions), preserving structure while reducing memory.
  + **Strings**: Summary statistics (e.g., number, average length, entropy, and counts of URLs/paths/registry).
  + **Imports**: Only DLL import counts, avoiding API-level feature explosion.

### Flattening Nested Structures

### Since tree-based models and CNNs do not support nested structures, we flattened only the selected nested columns.

### Malicious Data Filtering

### Only samples labeled as 1 (Malicious) were retained. Benign (0) and unlabeled (-1) samples were excluded, as our task focuses solely on malware family identification.

After filtering, our training and validation set contains only 63,713 samples in total, while the test set has 100,000 samples remaining.

### Class Merging

### To reduce the number of highly fragmented malware family classes, we consulted ChatGPT, Gemini, and external sources including VirusTotal, Kaspersky, Malpedia, and Microsoft Security Intelligence. Based on this, we consolidated avclass into broader categories such as Trojan, Ransomware, Downloader, Worm, Spyware, etc. Classes with very few samples were grouped into an "Others" category. Samples with missing avclass were labeled as "Unknown" and removed (5.68% of training data).

### **Mitigating** Data Shift

### We compared class distributions across training and testing sets. Categories like "Ransomware", "Worm", and "Generic" were underrepresented in training but prevalent in testing. These were removed to minimize distributional shift and ensure reliable evaluation.

### Separating Features and Target

### As part of the feature selection and data cleaning process, several columns were excluded from the input feature set. sha256 and md5, which are file hash identifiers, were excluded due to their lack of relevance in malware family classification. Similarly, avclass and avclass\_grouped, which serve as the target labels (y), were excluded from the input features (X). Since the dataset was filtered to include only malicious samples (i.e., label = 1), the label column became constant and thus provided no additional information; it was also discarded. Additionally, based on prior feature selection, we excluded appeared, byteentropy, section, datadirectories, and exports due to their limited predictive value.

### Label Encoding

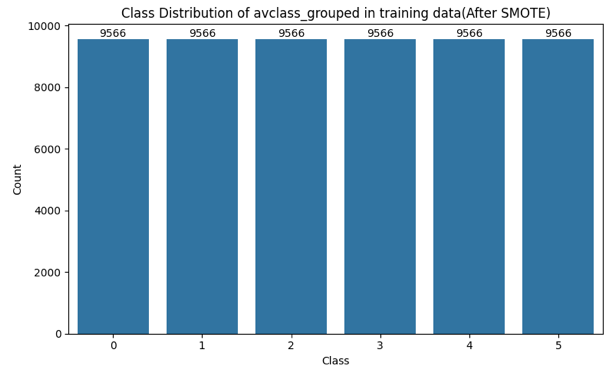
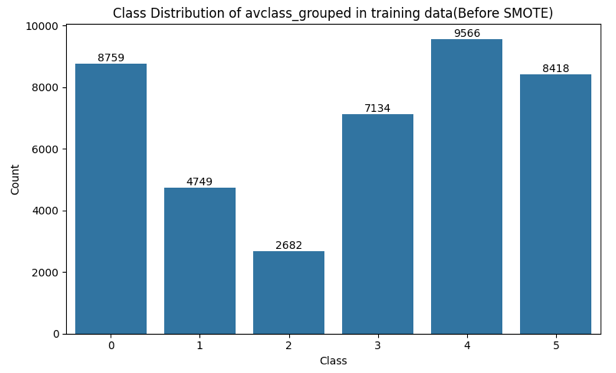
### The target variable avclass\_grouped was label-encoded into numerical format, as models like CNNs do not handle string inputs.

### Train-**Validation** Split

### The training data was split into training and validation sets to monitor generalization and avoid overfitting.

### **Addressing** Class Imbalance

### The class distribution ranged from 2,682 to 9,566 samples. To mitigate bias toward dominant classes, we applied SMOTE to synthetically oversample minority classes.



### **Feature** **Scaling**

### Input features vary widely in scale (e.g., general\_size in millions vs. binary features). We applied standard scaling to normalize feature ranges and prevent disproportionate influence from high-magnitude variables.

# Related Work Section

We identified that Ahmadi et al.'s work shares a common goal with ours: to classify malware samples into their respective families rather than simply detecting their presence. Their study introduced a novel classification framework capable of accurately assigning malware variants to known family groups. While they used the Microsoft Malware Classification Challenge dataset, which comprises approximately 20,000 labeled malware samples, our study utilizes a significantly larger corpus with 163,000 malicious samples from the EMBER dataset to train our models.

The paper emphasizes behavior-based static feature extraction and employs a feature fusion strategy to optimize classification performance. Notably, the authors deliberately avoid unpacking due to its computational expense and unreliability when handling obfuscated binaries. Instead, they adopt flattened, summary-level features that capture the essence of malware structure and behavior. Our research similarly avoids high-dimensional representations such as full API lists and byte histograms, as well as complex nested features like opcode sequences or control flow graphs. This design choice helps reduce memory consumption and supports efficient model training on large-scale data.

While their system, based on an XGBoost classifier, achieved near-perfect accuracy (≈99.8%), our tree-based models, Decision Tree and Random Forest, achieved only 40% accuracy. We also experimented with a deep learning model for comparison; however, its classification performance was also notably lower, with an accuracy of approximately 30%.

# Description of Your Approach

## Decision Tree/Random Forest:

In our study, we employed Decision Tree and Random Forest classifiers as the models for malware family classification. These tree-based algorithms were chosen for their interpretability and low computational cost. They also work well with structured static malware data where each sample is described by measurable features such as file size, section count, or string entropy. Random Forest, as an ensemble method, improves generalization by combining multiple decision trees. Each tree is trained on a random subset of the data and features. This ensemble strategy helps reduce overfitting which is common in single decision trees.

Our approach differs from that of Ahmadi et al., who used the XGBoost classifier with extensive feature fusion. In contrast, we emphasize feature selection over fusion to maintain model simplicity and scalability. Their system was trained on a dataset of approximately 20,000 malware samples. Our models used a significantly larger dataset, with 163,000 malicious samples from the EMBER dataset. This allowed us to evaluate performance under more realistic, large-scale conditions. Both methods focus solely on static analysis and deliberately exclude unpacking or runtime behavior monitoring. This design choice prioritizes efficiency and scalability, enabling classification without the high overhead or unreliability often associated with dynamic or unpacking-based techniques. Instead, they rely on flattened summary features to represent malware samples. However, our method is designed for higher efficiency. We excluded high-dimensional features, such as full byte histograms and complete API call sets. We also avoided complex nested structures, such as opcode sequences and control flow graphs. This strategy reduces memory usage and training time, making the system more practical for deployment in constrained environments.

## Convolutional Neural Network (CNN):

Among the models introduced after the midterm, we selected the one-dimensional Convolutional Neural Network (CNN) for our multi-class classification task. Autoencoders, while effective for anomaly detection, require training a separate model per class and evaluating each test instance across all models—a process that is complex and inefficient for our use case. Similarly, Generative Adversarial Networks (GANs) are primarily designed for data generation rather than classification, and thus were not considered suitable for our objective.

We implemented a CNN model to classify malware samples into six categories, using a SMOTE-balanced training set to address class imbalance. The architecture includes a Conv1D layer for local feature extraction, followed by a GlobalMaxPooling1D layer and three fully connected (Dense) layers. A softmax output layer produces class probabilities. The model was trained on standardized input features with early stopping to prevent overfitting and a model checkpoint mechanism to save the best-performing model (best\_cnn\_model.keras) based on validation accuracy.

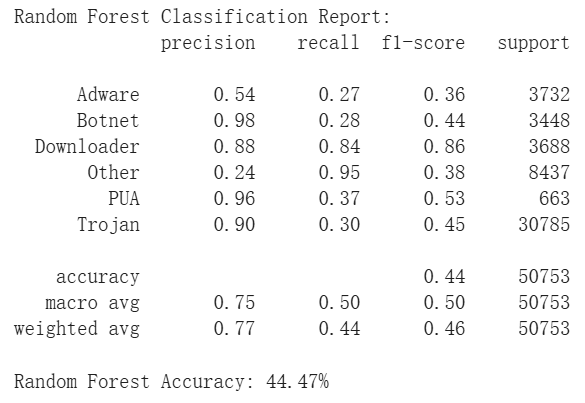
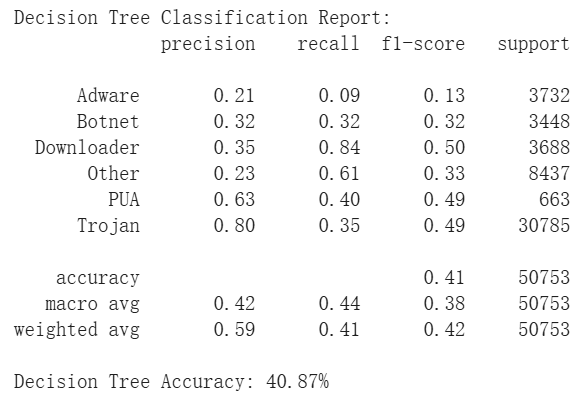
Interestingly, during training on the SMOTE-augmented data, we observed that validation accuracy consistently exceeded training accuracy—a counterintuitive result. We hypothesized that this may be due to synthetic patterns introduced by SMOTE. To verify this, we trained an additional CNN model without SMOTE for comparison.

# Results, Analysis and Discussion

## Decision Tree/Random Forest:

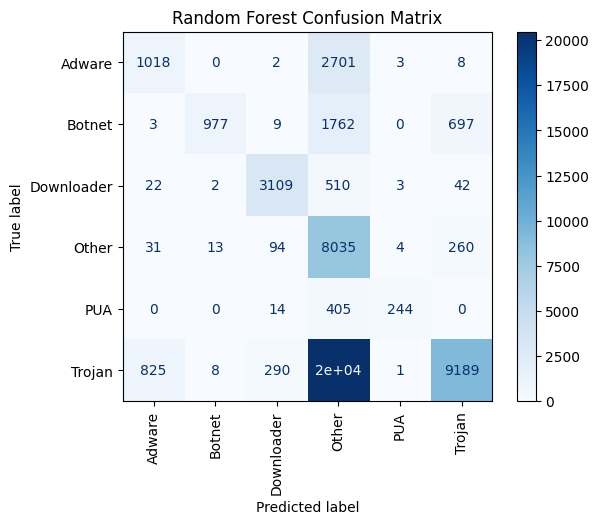
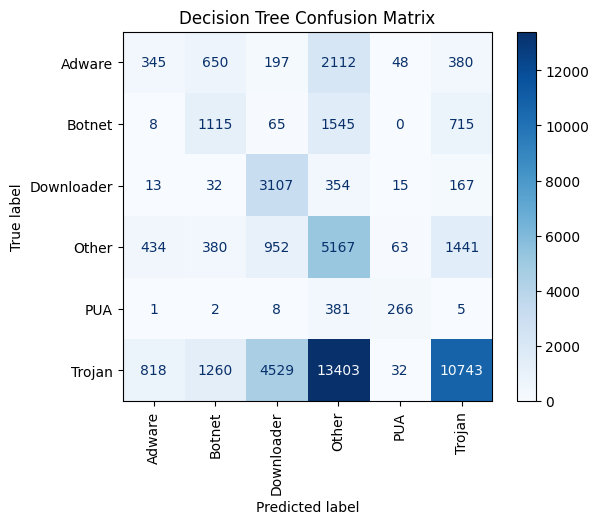
The main objective of this study was to build an accurate and interpretable model for malware family classification. We hypothesized that tree-based models, such as Decision Tree and Random Forest, would offer strong performance. These models were expected to scale efficiently and generalize well on large datasets. However, the results of our experiments didn’t support this hypothesis.

### Classification Report



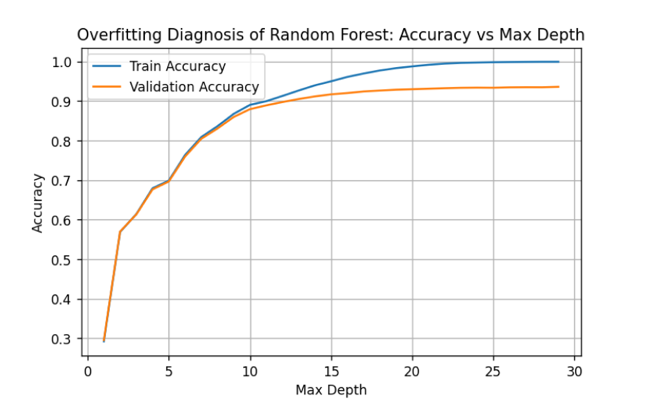
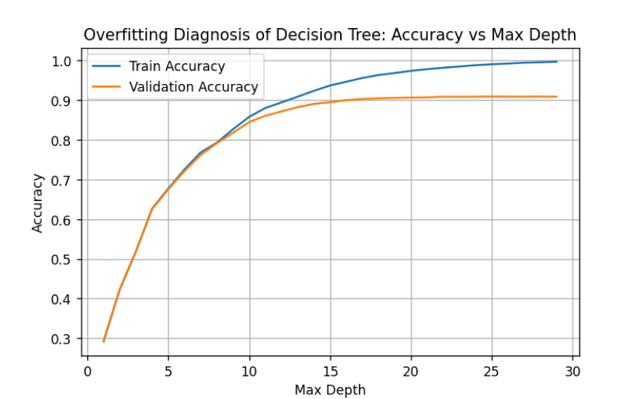
### Both models achieved strong cross-validation accuracies, but Random Forest demonstrated slightly higher stability and better overall generalization. In particular, the Random Forest model performed notably well on the Downloader class, achieving a high precision of 0.88 and a recall of 0.84. In contrast, both models struggled with classes such as Adware, Botnet, and Trojan. For these families, precision and recall scores were much lower and more imbalanced. This imbalance points to challenges in distinguishing complex or overlapping feature patterns among major malware families. The macro average F1-score further highlights the model's difficulty in maintaining balanced performance across all classes as well.

### Confusion Matrix



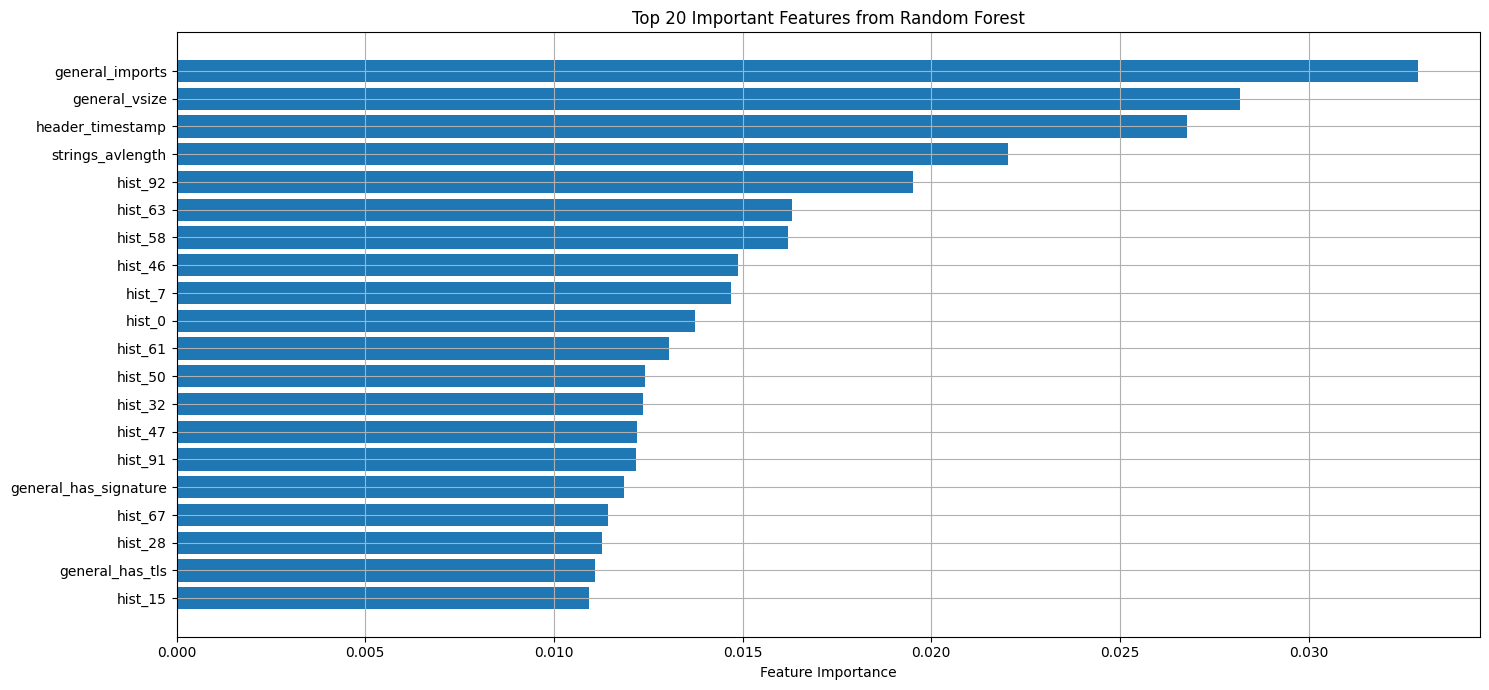
### A comparison of the confusion matrices reveals that the Random Forest model improves classification performance across several malware families compared to the Decision Tree. Specifically, correct classification of Adware increased from 345 to 1,018 samples, and Other improved from 5,167 to 8,035 correctly predicted samples. PUA detection also improved slightly. Despite these improvements, both models continued to misclassify a large number of samples as Other which indicates that family overlap and feature similarity remain challenging.

### The impact of depth on Accuracy



### Overfitting analysis showed that both models began to overfit beyond a maximum depth of approximately 15. Random Forest exhibited milder overfitting with validation accuracy plateauing around 93%. In contrast, the Decision Tree showed sharper overfitting and less consistent validation performance. Overall, Random Forest produced more stable results across malware families and demonstrated greater resilience against overfitting. These findings suggest that ensemble methods are more suitable than single decision trees for malware family classification using static features.

### Feature Importance



### The feature importance analysis revealed that the Random Forest model relied on a diverse set of static characteristics to classify malware families. API related information, particularly “general\_imports”, ranked highly, reflecting differences in functional dependencies across malware types. Metadata features such as “header\_timestamp” and “general\_vsize” were also highly influential. It suggests that compile time and file size provide strong family-specific signals. String based features, including “strings\_avlength” and “strings\_entropy”, captured obfuscation patterns commonly used by certain families. Additionally, byte histogram features such as “hist\_92” and “hist\_58” were prominent which indicates that structural byte-level patterns are effective for classification. Finally, security indicators like “general\_has\_signature” and “general\_has\_tls” contributed to detecting whether samples possess attributes typically associated with legitimate software. Together, these findings highlight the strength of flattened static features in distinguishing malware families without relying on dynamic analysis.

## Convolutional Neural Network (CNN):

### Comparison Between SMOTE and Non-SMOTE Models

1. **Impact of SMOTE on Model Validation Performance**

Upon comparative analysis, it was observed that models trained without the application of SMOTE consistently exhibited lower validation accuracy compared to their corresponding training accuracy. This behavior aligns with typical expectations in supervised learning, where models generally perform better on training data than on unseen validation data. In contrast, the unusual phenomenon of validation accuracy exceeding training accuracy was predominantly observed in models utilizing SMOTE. This suggests that SMOTE may be a contributing factor to this counterintuitive outcome.

|  |  |
| --- | --- |
| SMOTE model | Non-SMOTE model |
| Inserting image... |  |

1. **Overall Performance Comparison**

The SMOTE-based model achieved a validation accuracy of 0.7945, reflecting stronger performance during training and validation. However, its test accuracy dropped significantly to 0.2751, indicating poor generalization to unseen data. In contrast, the non-SMOTE model attained a slightly lower validation accuracy of 0.7211 but showed marginally better generalization with a test accuracy of 0.2447.

Regarding F1-score, the SMOTE model outperformed in terms of macro F1 (0.25), which treats all classes equally without accounting for class imbalance. This suggests improved performance across minority classes.

|  |  |
| --- | --- |
| SMOTE model | Non-SMOTE model |
|  |  |

1. **Differences in Minority and Trojan Class Performance**

The SMOTE model was able to effectively recognize minority classes such as Downloader, achieving a recall of up to 0.81. This suggests that synthetic oversampling helped the model learn patterns from underrepresented classes. However, this also led to the model becoming overly reliant on the characteristics of synthetic samples, resulting in poor generalization on real-world Trojan data, where recall dropped to only 0.16.

Conversely, the non-SMOTE model exhibited almost no ability to identify minority classes such as Botnet and Downloader, with recall as low as 0.03. Nevertheless, it achieved a much higher recall of 0.21 for the Trojan class, demonstrating better generalization for high-frequency classes in real-world distributions.

1. **Observations from the Confusion Matrix**

|  |  |
| --- | --- |
| SMOTE model | Non-SMOTE model |
|  |  |

Regardless of whether SMOTE was applied, both models showed a clear tendency to predict samples as Trojan or Other. This behavior led to many samples from classes such as Adware and Downloader being misclassified into these dominant categories. While the SMOTE model provided more balanced predictions across minority classes, this came at the cost of a significant decline in Trojan class performance. In contrast, the non-SMOTE model maintained more stable performance on Trojan, albeit at the expense of minority class recognition.

1. **Conclusion**

In summary, applying SMOTE improved recall for minority classes but also introduced overfitting to synthetic data, reducing the model’s ability to generalize to real-world distributions. The non-SMOTE model, while unable to effectively learn from minority classes, maintained consistent performance on the major class (Trojan), making it more robust in real-world applications where data is inherently imbalanced.

### Comparison of Decision Tree, Random Forest, and CNN Models

### We compared the performance of three different models—Decision Tree, Random Forest, and Convolutional Neural Network (CNN)—for our malware family classification task. Among these, the Random Forest model achieved the highest accuracy, followed by the Decision Tree, with the CNN exhibiting the lowest performance. The superior performance of the Random Forest likely stems from its ensemble-based architecture, which aggregates predictions from multiple decision trees, effectively reducing variance and improving generalization, particularly in structured tabular data scenarios. The Decision Tree, while simpler and faster, exhibited lower accuracy due to its susceptibility to overfitting on complex data. The CNN model, despite its powerful feature extraction capabilities, underperformed possibly due to the inherent characteristics of our dataset, which consists predominantly of structured numerical features rather than spatially or sequentially correlated data—contexts where CNNs typically excel.

# Conclusion

## Summary of Findings

This study evaluated the performance of three models—Decision Tree, Random Forest, and Convolutional Neural Network (CNN)—for malware family classification using structured static features. The results showed that Random Forest achieved the highest classification accuracy, followed by Decision Tree, while CNN performed relatively poorly, indicating its limitations in handling tabular data. Through feature importance analysis, the study identified metadata, import counts, string entropy, and byte patterns as key factors contributing to classification. Additionally, although the use of SMOTE improved validation performance, it led to decreased generalization on the test set, suggesting that oversampling may distort real-world data distributions. Overall, the study highlights the advantages of traditional machine learning models, such as Random Forest, in structured feature domains and emphasizes the need for cautious evaluation of data balancing techniques in cybersecurity applications to ensure model reliability and practical usability.

## Significance

The findings of this study have important implications for the field of malware detection. First, the superior performance of Random Forest demonstrates that traditional tree-based models remain highly effective for structured, static feature domains. Their accuracy, scalability, and interpretability make them ideal candidates for integration into real-world cybersecurity pipelines where transparency and performance are both critical.

Second, the feature importance analysis offers actionable insights for security practitioners and researchers by identifying which binary attributes are most relevant for malware family classification. This contributes to the development of more focused, efficient, and explainable detection systems, reducing noise and improving response times in operational environments.

Finally, the observed limitations of SMOTE reveal potential risks associated with synthetic oversampling in cybersecurity contexts. By highlighting how it can inflate validation results while harming generalization, our study provides an important cautionary perspective for future work on imbalanced datasets. Overall, this research not only advances practical malware classification techniques but also informs the design of more robust and trustworthy machine learning models in security applications.

## Limitations and Challenges

One major limitation of this study lies in the computational constraints that restricted our ability to fully utilize the available dataset. Due to hardware and memory limitations, we significantly reduced the size of the training set and simplified the target variable by merging multiple malware families into broader categories. While this approach allowed the model to be trained more efficiently, it inevitably limited the granularity of classification and may have impacted the model’s ability to capture subtle distinctions between malware families. Additionally, the exclusion of a substantial portion of the original data likely contributed to lower overall accuracy and reduced generalizability. Future work with greater computational resources could incorporate the complete dataset and finer-grained class labels to build a more robust and comprehensive model.

## Future Research

Future work may focus on refining the label merging strategy and exploring alternative data augmentation techniques. The classification report indicates that the model overpredicts the "Others" category, exhibiting high recall but low precision. This suggests that the broad consolidation of numerous distinct malware families into the "Others" class may have hindered the model’s ability to learn discriminative features for that group. Future studies could explore more nuanced merging methods to improve classification performance. Additionally, for low-frequency classes, the use of Generative Adversarial Networks (GANs) may be explored as a more sophisticated alternative to SMOTE, potentially generating more realistic and diverse synthetic samples that enhance the model's ability to generalize across rare classes.

# References

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., & Giacinto, G. (2016). Novel feature extraction, selection and fusion for effective malware family classification. *CODASPY '16: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*,183–194. <https://doi.org/10.1145/2857705.2857713>

Virustotal. (n.d.). <https://www.virustotal.com/gui/home>

Kaspersky. (n.d.). <https://threats.kaspersky.com/>

Fraunhofer FKIE. (n.d.). <https://malpedia.caad.fkie.fraunhofer.de/>

Microsoft. (n.d.). <https://www.microsoft.com/en-us/wdsi/threats>