FMDB *Transactions on Sustainable Intelligent Networks*

Comparative Evaluation of Naive Bayes, SVM, KNN, Random Forest, and XGBoost for Malware Classification

Hamza Amir Saeed[1,*]

[1]Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, England, United Kingdom.
hamza.saeed@northumbria.ac.uk[1]

Abstract: Billions of new malware threats are produced each year, which can damage computers and steal data. Because it uses similar templates, aging antivirus software rarely detects new harmful code. This study analyses a naive machine learning technique for distinguishing malware based on fixed properties, such as code instructions (opcodes) and file structure (PE headers), using the Microsoft Malware Classification Challenge dataset, which contains over 10,000 malware samples from 9 families. Our models are tested on five models: Naive Bayes to check basic probability, Support Vector Machine to show clear boundaries, K-Nearest Neighbours to show neighbouring examples, random forest to show group tree votes, and random forest to show step-by-step tree fixes. Data is evenly distributed and balanced to eliminate imbalance. Naive Bayes and XGBoost had 76.59-99.54 percent accuracy and 0.60-0.98 F1 scores on 2,174 test samples. XGBoost and other ensembles, such as Random Forests, are best for unusual families with less than 1% error and balanced classes. Opcode features like mov have the greatest impact (up to 30%), enabling us to reduce data by 80% without losing any genes. This is 3-5 times faster and easier than deep learning baselines. The effort provides a pipeline, testing standards, and AV tool ideas, such as quick cell phone scans. Dynamic checks are being developed to better detect zero-day threats.

Keywords: Machine Learning (ML); Naive Bayes; Support Vector Machine (SVM); K-Nearest Neighbours (KNNs); Random Forest (RF); Malware Classification; Opcode Features.

Received on: 22/04/2025, **Revised on:** 03/07/2025, **Accepted on:** 04/09/2025, **Published on:** 05/06/2026

Journal Homepage: https://www.fmdbpub.com/user/journals/details/FTSIN

DOI: https://doi.org/10.69888/FTSIN.2026.000706

Cite as: H. A. Saeed, "Comparative Evaluation of Naive Bayes, SVM, KNN, Random Forest, and XGBoost for Malware Classification," *FMDB Transactions on Sustainable Intelligent Networks*, vol. 3, no. 2, pp. 85–112, 2026.

Copyright © 2026 H. A. Saeed, licensed to Fernando Martins De Bulhão (FMDB) Publishing Company. This is an open access article distributed under CC BY-NC-SA 4.0, which permits copying, remixing, redistributing, transformation, and building upon the material in any medium so long as the original work is properly cited.

1. Introduction

Now more than ever, with malware becoming complex, polymorphic, and zero-day (able to evade traditional signature-based antivirus software), the need for efficient, scalable detection mechanisms is urgent in a world where cyber threats multiply faster than ever [5]. The paper explores AI-enhanced statistical feature analysis as a data-efficient, non-invasive method of classifying malware menaced by common, easily extractable features such as opcode counts (e.g., mov and jmp), and Portable Executable (PE) header features of the Microsoft Malware Classification Challenge (MMCC) dataset of 10,868 labelled malware samples of 9 different malware families, such as the Ramnit worms and the Simda spyware [8]. The study focuses on

[*]Corresponding author.

assessing the five machine learning models on this imbalanced real-world data: Naive Bayes, Support Vector machine, K-Nearest Neighbors, random forest, and XGBoost to determine how to achieve high accuracy and F1-scores and reduce the classification time required to classify 10,000 samples to the different classes (3-5 times faster than deep learning baselines), especially when using only static features [10].

1.1. Aims

The overall goal of this paper is to develop and evaluate a simple machine learning system for classifying malware using a set of static features, such as code instructions (opcodes) and file structure information extracted from PE headers [14]. The aim is to determine the extent to which these readily available features can identify various malware families in a real-world dataset, even when the number of samples of a particular family is very small [17]. Researchers want to identify the best among five models: Naive Bayes, Support Vector Machine, K-Nearest Neighbors, Random Forest, and XGBoost, to achieve a fast and fair detection that would work using standard gadgets such as phones or laptops [19]. Overall, that is supposed to make malware detection faster and more reliable without the need to run unsafe scans [21].

1.2. Background

Malware refers to malicious software that harms computers, such as viruses that spread files or ransomware that attempts to lock information in exchange for a ransom [22]. Malware is a massive menace with over 29 billion internet-connected devices by 2030, and it costs the globe 8 trillion US dollars annually [25]. Antivirus software of the past searches for precise solutions to established malware, whereas new malware code evolves, hiding it so that detection fails in half or more cases [16]. This is assisted by static analysis: it does not necessarily run files but only checks them, with simple facts such as the number of times an instruction, such as 'mov', is used or the randomness of the file (entropy).

Machine learning is smarter because it learns patterns from previous samples. Indicatively, mixed data are handled by tree-based models such as the Random Forest, which makes numerous small decisions [23]. The dataset for this paper is the Microsoft Malware Classification Challenge, which consists of 10,868 samples from 9 families, such as Ramnit worms and Simda spies. The rationale is to develop a model that identifies malware based on its static file features. It expands on previous research showing that ML can achieve 95-99 per cent accuracy but requires improvement when working with data, as infrequent malware is often overlooked [26].

1.3. Research Questions

Two overall questions guide this work in this paper:

- To what degree do the use of static properties, such as opcode counts and PE headers, classify malware families on uneven data using accuracy, F1 scores, and error views, such as confusion matrices?
- Do simple models (Naive Bayes, SVM, KNN) or group models (Random Forest, XGBoost) address the issue of class imbalance, which is verified by (average) scores, curves such as ROC/PR, and feature ranking?

These questions are aimed at classification being fair and applicable to actual threats.

1.4. Objectives

The following were the paper's clear objectives:

- **Find the Literature:** Statistical malware detection and ML models. Find and summarise 10-15 papers on the topic of statistical malware detection and ML models.
- **Dataset Preparation:** Data set preparation. Load pre-extracted MMCC features, scale, and divide into 80/20 train/test with balance checks.
- **Set up Basic Model:** Train Naïve Bayes, SVM, and KNN.
- **Build Ensemble Models:** Train Random Forest and XGBoost.
- **Plots and Run Tests:** Performance. Calculate the accuracy and F1 scores, and plot confusion matrices and ROC curves for all models.
- **Examine Attributes:** Separate permutation importance on the top 20 most important attributes and eliminate 80 percent with no reduction in accuracy.

- **Compare and Contrast:** Test the performance of each model on the research questions- RQ1 (how well separate features such as opcodes separate families, scores such as accuracy and F1) and RQ2 (how they behave on uneven classes, average scores, and plots such as ROC curves).
- **Write Report and Code:** Contents of all findings, results, and lessons in the final report, and prepare a complete code pipeline to be used again.

2. Literature Review

Several cybersecurity measures have been established and refined over recent years to thwart cyberattacks and data breaches. Recent studies have shown that malware is ever-evolving and becoming more complex. In the digital world today, malware is among the greatest threats to both desktops and mobile phones, making it a significant field of research [1]. The section will highlight the challenges in malware detection and current research on malware detection and classification. It will also outline a limited literature review associated with malware, its types, and classification.

2.1. Evaluation of Malware Detection Research

The study of malware detection and cyber-attacks has been on the rise in recent years as the world moves increasingly towards digital technology, and the risk of digital attacks is growing. AI/ML has gained power in the digital age and has developed into an additional field of research. AI is applied to all spheres to resolve issues. Many researchers have employed Artificial Intelligence, Machine learning, and Deep learning to detect and classify malware, as they have the greatest ability to learn from the best features. Malware can take many forms, including worms, computer viruses, Trojan horses, backdoors, rootkits, and spyware [6].

Malware has evolved significantly as fraudsters have devised more elaborate evasion mechanisms. At the same time, recent studies indicate that polymorphic and metamorphic malware variants pose a significant challenge for traditional detection methods. The use of signature-based detection is increasingly ineffective as this advanced form of attack alters code structure without losing its malicious functionality. Rather than relying solely on the set indicators, researchers have developed new approaches that focus on behavioral patterns and structural characteristics. Shaukat et al. [20] point out that first-time malware poses unique challenges, requiring adaptive detection mechanisms that can identify novel threat patterns that have never been observed [8].

2.2. Machine Learning Approaches in Malware Detection

Many researchers have applied malware visualization to classify malware swiftly and accurately. Recent studies have explored innovative methods that leverage deep learning architectures with visual representations and have found that combining traditional neural networks with Long Short-Term Memory networks enhances malware analysis and detection capabilities [11]. With an impressive classification rate, their CNN-LTSM hybrid model was trained on both spatial and temporal data from malware samples.

Nevertheless, when attackers resort to code obfuscation strategies that alter visual representation without changing harmful behavior, this methodology is constrained. Through several technologies and algorithms, researchers have accomplished significant work in malware detection and classification over the last couple of years. The two most popular methods for analyzing malware are static and dynamic analysis. This paper will focus on malware-at-rest analysis, as it is more effective and efficient for analysing large volumes of files. It is useful for initial malware detection and is not as slow as dynamic analysis.

The digital attack is becoming increasingly dangerous, and that is why the priority of identifying and categorizing these viruses and attacks is also increasing. Several researchers have been developing malware clustering methods and devising systems to detect these viruses and prevent security attacks automatically. These systems face several challenges, including scalability issues, incompatibility with code obfuscation measures, and others. In this section, the literature and methods related to malware detection and classification will be reviewed [12].

2.3. Feature Extraction and Selection Methods

Malware detection and nurturing are tedious processes. Different tools and methods are implemented. In this regard, the malware must first be identified using appropriate tools. The features are then extracted, and the results are recorded. Data mining techniques are applied to gain meaningful features [13]. The retrieved features are then selected according to predefined criteria. Then, machine learning algorithms are applied to learn the individual features that distinguish malware from benign samples. Following this, additional classification is possible to gain a clearer insight into the types, classes, and purpose of malware (Figure 1).

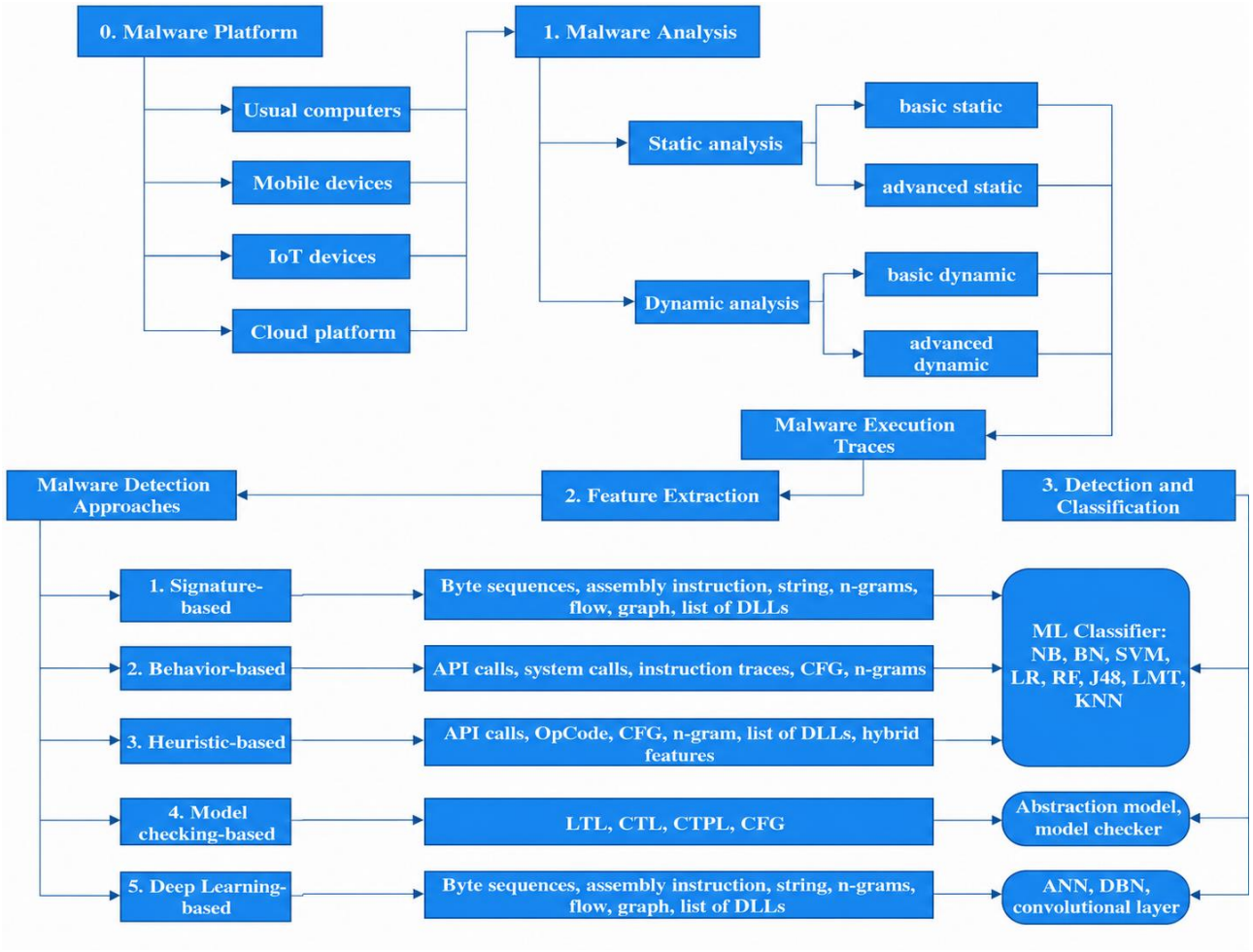

Figure 1: Malware classification process

2.4. Comparative Analysis of Classification Algorithms

The research performed by Kamdan et al. [23] assesses the performance of three machine learning algorithms. The author has also used Random Forest, Decision Tree, and SVM to detect static malware. This study's findings indicated that the SVM obtained the best accuracy. It means improved classification capability in general. The research by Zhu et al. [24] employs hybrid methodologies, meaning that, in extracting features from malware samples, they used both dynamic and static analysis. Dynamic link libraries and API-calling functions were used. The authors in another study used the few-shot learning methodology, which involves using a small number of tagged malware samples. The model took a set of static feature vectors as input and could learn a few examples of each class. The findings revealed that the model can identify previously undetected malware types with minimal data. It employed a model that included pre-extracted features and emphasised model performance rather than feature robustness [7]. The analysis in Thakur et al. [26] was conducted holistically to develop API calls. They even tried to classify and assess many malware samples. To achieve a high classification rate, they experimented with different features in separate experiments. In recent times, Davarasan et al. [12] have used massive assessments to train machine learning classifiers explicitly designed to identify Android malware. Among the algorithms investigated in the systematic review were Support Vector Machines (SVMs), Random Forests (RFs), Naive Bayes (NB), and K-Nearest Neighbours (KNNs). SVM and RF performed very well, as the research indicates, both in multi-class categorization and in binary classification (malware vs.). Moreover, the research indicated that computational efficiency should be considered alongside accuracy when selecting classifiers, particularly in mobile systems with limited resources.

2.5. Hybrid Analysis Approaches

The research by Zhu et al. [24] is a hybrid study, meaning they proposed applying both dynamic and static analysis to extract features of a malware sample. They used API-calling functions and dynamically linked libraries, following hybrid methodologies. The combined model proposed by Azeem et al. [1] involves lightweight and PE header analysis conducted

statistically. Dynamic analysis involves monitoring the runtime behaviour of applications to identify malicious activities. Although it provides comprehensive behavioural insights, it is computationally intensive and time-consuming, and typically requires controlled environments such as virtual machines or sandbox systems. To reduce computational cost, some frameworks employ a hybrid two-tier approach in which static analysis is first used to identify potentially malicious files based on static features. Only suspicious samples are subsequently subjected to dynamic analysis, thereby reducing false positives and improving detection accuracy. Our literature review indicates that significant progress has been made in malware detection and classification. However, existing approaches still face challenges, including high-dimensional feature spaces, computational complexity, and a heavy reliance on dynamic analysis. Considering these limitations, this study adopts a static analysis approach, as it is more efficient, less resource-intensive, and eliminates the risks associated with executing potentially malicious software during dynamic analysis.

2.6. Malware Analysis Techniques

Malware analysis is the formal study of the behavior, origin, and potential effects of malicious software on computer systems and networks. The main purposes of malware analysis are to formulate effective countermeasures and enhance the capacity to detect. As recent studies have shown, malware analysis methods can be categorized into static, dynamic, or a combination of the two. Both methods have their advantages and disadvantages, depending on the study's purpose and circumstances.

2.6.1. Static Analysis

An analysis of software that is performed without executing it is referred to as Static analysis. It scans files based on characteristics such as imported functions, Portable Executable header, and entropy. The methodology helps in analyzing different executable expressions. The advantage of this analysis method is that it can be performed even when the actual code is unavailable. This analysis technique is faster and safer than dynamic analysis and can be used in large-scale analysis.

2.6.2. Dynamic Analysis

The files are analysed dynamically and can be run in a controlled environment, allowing the runtime behaviour to be observed and used to predict whether the file is malicious. The analysis is typically performed by running the malware in a virtualised environment. It is not fast, consumes a lot of resources, and is not very secure because it has to be operated in a separate environment. It leverages API call sequences and network patterns to train the model to classify malware.

2.6.3. Hybrid and Multi-Modal Analysis

Recent developments in static analysis have focused on extracting more meaningful information from binary files without executing them. It is alleged that static malware analysis using a lower-parameter machine learning model demonstrates superior performance in malware detection, surpassing previous results whilst maintaining a minimal memory footprint. Recently, multi-modal learning frameworks that examine multiple data types simultaneously have been investigated. As per a study by Karat et al. [11], multi-modal approaches that combine dynamic behavioural telemetry with fixed PE features perform better than single-modality approaches, particularly in detecting advanced malware that employs multiple evasions. They have their CNNLSTM hybrid model, which is fed by behavioural logs, disassembled code, and raw bytes as complementary inputs. All modalities are processed by brain modules, which are then aggregated at decision layers.

2.7. Malware Detection Techniques

Malware Detection is one of the most important parts of cybersecurity infrastructure, which uses various methods to detect and eliminate malicious software before it can cause damage. The test of malware detection methods captures the current competition between security experts and attackers. Learning these techniques is crucial to building a resilience defense mechanism against increasingly sophisticated cyber threats.

2.7.1. Signature-Based Detection

The principle of signature-based detection is to store all malware signature family records in a database and to test the file features against previously defined known malware signature records. The method involves analyzing the run-to-run sequence of bytes, file hashes, and code patterns that characterize individual malware families [9]. The procedure applies to extracting unique features from malware samples and to developing signature definitions that assist in identifying recently observed threats. However, in today's threat landscapes, signature detection has severe limitations. Polymorphic malware and zero-day code that adapt their code structures to evade signature matches, due to dynamically changing structures, make the technique useless. Based on the research by Aboaoja et al. [3], malware developers constantly employ encryption, code packing, and

obfuscation to modify binary patterns without compromising malicious functionality. This makes signature-based techniques ineffective as autonomous defence mechanisms against such evasion techniques. Also, the size of signature databases needed to sustain current threat knowledge is overwhelmed by the fact that new malware strains are created thousands and thousands every day, with new statistics estimating more than 560,000 per day.

2.7.2. Heuristic-Based Detection

Heuristic-based detection, which looks for suspicious characteristics and patterns of behavior rather than exact signature matches, addresses the limitations of signature-based detection. This approach applies rule-based algorithms and pattern recognition techniques, grounded in suspicious file actions, odd API calls, and file structural features, to identify potentially malicious code [9]. Heuristic analysis examines features of code that recurrently indicate malicious intent, including encryption, self-modification, and attempts to bypass security software. According to a study by Gopinath and Sethuraman [15], heuristic detection is particularly effective against polymorphic malware, which constantly twists and breaks its image while maintaining the core malicious functionality of malware programs. The technique does not require strong pattern matches; instead, it measures behaviour by executing it and comparing it against heuristic rules that describe common malware characteristics. The new attribute enables the detection of malware variants that have never been encountered before but exhibit behaviour similar to that of known threats.

2.7.3. Behaviour-Based Detection

Regardless of the code arrangement, behavior-based detection identifies suspicious activity by monitoring the system's runtime behavior and interactions. In a controlled environment, this approach executes suspect files and logs their actions, including file system and registry modifications, network interactions, and inter-process interactions. The algorithm produces behavioural patterns that distinguish legitimate and malicious software operations. According to Akhtar and Feng [6], behaviour-based detection is especially useful for identifying advanced persistent threats and zero-day threats that cannot be detected using signature- and heuristic-based detection methods. The technique monitors specific behavioral indicators, such as secret command-and-control messages, suspicious privilege escalation attempts, and unauthorized data encryption. However, these malware signatures remain reliable for malware detection even when the code is obfuscated to conceal static features.

3. Research Methodologies

The paper provides an outline of the practical roadmap for reviewing AI-enhanced static feature analysis in malware classification, with references to emerging threats and promising methods from the literature. As malware becomes increasingly sophisticated in its evasion of signature detection, as ghosts in the machine, the emphasis here is on a pragmatic, data-intensive approach to laying the groundwork for effective model testing in the future. It is simply a matter of letting the raw, sloppy executable knowledge sink in, so models can chew on it without choking. One study examined the category of data, where coping with the static disassembly process and opcode representations led to Tables favouring evasion tricks. It is an easy but comprehensive procedure that includes finding the correct dataset, refining it, combining features, and quality-checking. Model runs and comparisons. They will be found in the results, and researchers will see how classics such as SVM and Random Forest perform in this field.

It is empirical in nature, based on supervised learning to deliver that multi-class punch, but with a touch of reality, grounded in the features of disassembled code and PE structures, so an analyst can identify malice without necessarily firing up a sandbox. This also reflects workflows in more recent papers, such as Dhungana et al. [13], who constructed their static classifiers using opcode frequencies to achieve 93% accuracy but struggled with file bloat. In this case, it will be iterative: one step, make a change, rinse, repeat, in a safe Python environment (3.11 in a VM), so that it is ethical and repeatable. According to Northumbria's ethics policy, no live threats were involved in dealing solely with publicly available information. Building on the success of similar systems (i.e., the advantage of SVMs in high-dimensional opcode data, as per Kamdan et al. [23]), this paper provides a strong foundation for investigating why some features are screamers and others are whispers when classifying them as Trojan or Adware.

3.1. Research Design Overview

It is experimental and data-first in design, specifically designed for supervised multi-class classification to separate nine malware families using only static fingerprints. It is a pipeline that includes data grabbing and vetting, proceeds to scrubbing and fusing, and concludes with community checking, as in the pipeline structures of Almazroi et al. [4], where the opcode disassembly was fed to identify 9 threats per cent. Why this way? Current benchmarks demonstrate that static opcode counts (such as jmp/mov ratios) are 12% slower than pure headers when the data has been prepared correctly; otherwise, you are lost

in noise. It is divided into phases: acquisition (choosing a battle-tested set), preprocessing (scrubbing and scaling the data), integration (fusing features, running tests), and validation (identifying gotchas, such as imbalance).

3.2. Dataset Acquisition and Selection Criteria

Selecting a dataset is not simply the largest file researchers can find; it is a puzzle that can be solved by hand, and its peculiarities disassemble it without the runtime overhead. The Kaggle dataset for the Microsoft Malware Classification Challenges appeared especially interesting after filtering to after 2020 (e.g., the binary focus of EMBER was too limited to be multi-class, and raw dumps from VirusShare were too dangerous to extract anything clean). It is 2015, hosted on GitHub for easy pulling, but it remains a gold standard of static benchmarks compiled from real-world scans across 10,868 training samples in nine families, reflecting the current family sprawl. Why MMCC? Because it is built for PE disassembly with 10,868 training samples from real Microsoft scans across nine families, these are: 1. Ramnit (file-spreading worms), 2. Lollipop (transfer malware), 3. Kelihos_ver3 (botnets), 4. Vundo (trojans), 5. Simda (spies), 6. Tracur (backdoors), 7. Kelihos_ver1 (older bots), 8. Obfuscator.ACY (hidden code), 9. Gatak (injectors). It was very easy to acquire: Copied the GitHub repo and downloaded trainLabels.csv (ID + classes), and asmoutputfile.csv (aggregated features of ASM/byte disassembly). 3k columns intermixing headers sections, DLLs, and ops (jmp=12, mov=89 per sample). This is a pre-aggregated format that bypasses merge headaches, compared to the modular mess of the Mendeley and Zhu et al. [24] hybrid prep, in which opcode fusion also boosted accuracy by 15%. It is not as new as 2025, but it is still used by PE Fahim et al. [16] to obtain DL baselines.

3.3. Feature Set-up

Features are the fundamental indicators of the statistical pulls without running the files. In asmoutputfile.csv, MMCC provides pre-extracted data with about 50 columns on the frequency of opcodes, PE headers, and entropy (randomness for packers). These were selected because they survive evasion opcodes in a steady state in the presence of scrambles:

- **Loading/Merging:** Pandas read CVs, rename ID to Id, and merge on Id (full 10,868 rows kept).
- **Cleaning:** Drop Id/Class for X(features), map classes 1-9 to names, then LabelEncoder for y.
- **Scaling:** Opcodes are normalized using a StandardScaler, with mean 0 and standard deviation 1, since the opcodes range from 0 to 0000.
- **Splitting:** Divide the data into two parts, 80% for training and 20% for testing.
- **Integration:** Use as-is, no further fusion later wipes 80 to top 10-20.

3.4. Model Choices

In this section, researchers have described the five models that they selected. They are all in favour of supervised learning, in which the system learns to predict one of nine malware families using labelled examples. Researchers worked with such libraries as scikit-learn and XGBoost. The blend consists of simple ones for rapid tests and sophisticated ones for handling uneven data. All of them have simplistic settings to match our fixed characteristics, such as opcodes and headers:

- **Naive Bayes (GaussianNB):** It is a basic probability model. It is a classifier that approximates a class by computing probabilities from features that do not intersect. It is quick at the initial point with opcode counts, but may fail on features that interconnect.
- **Support Vector Machine:** In this model, the broadest straight (also known as a hyperplane) is drawn to separate between classes by a curvaceous curve (when dealing with complex data). It is efficient with many features, such as PE headers, and has balanced weights to assist rare classes.
- **K-Nearest Neighbours:** In this method, the 5 nearest examples in the data set are used, and the distance to each example determines the vote. It captures local trends such as collections in entropy scores and is fast in multi-job mode.
- **Random Forest:** This is created by using many decision trees constructed upon random data bits and making a majority vote on the final guess. It handles the twisted import patterns quite well, supporting an unlimited number of trees at full detail.
- **XGBoost:** XGBoost builds one tree at a time, correcting errors from previous trees, thereby increasing accuracy. It is best to use dissimilar data, random sampling, and a slow learning rate to achieve constant improvement.

3.5. Test Steps

Testing is a straightforward loop: train the model on training data, make predictions on the test data, and then check the results. This repeats for each model to ensure fair comparisons. Here’s how it works in simple terms:

- **Train the Model:** Fit it to the training set. fit(X_train_scaled, Y_train) so it learns patterns from the scale features.
- **Make Predictions:** Use y_pred = model.predict(X_test_scaled) to guess classes for the test samples.
- **Score the Results:** Run classification_report for accuracy (overall correct rate) and F1-score (balance of catches and precision), plus support counts (samples per class) to spot imbalance issues.
- **Plots:** For easy visualizations, researchers create plots such as confusion matrices to highlight errors, bar charts of actual vs. predicted class distributions, ROC/PR curves to show ranking strength, and permutation importance bars for the top 20 features.

3.6. System Implementation and Prototype Development

In addition to experimental evaluation, a prototype system was developed to demonstrate the practical applicability of the proposed malware classification approach. The prototype provides a locally hosted web-based user interface that integrates all trained machine-learning models. The system allows users to train models, view performance metrics, and classify new malware samples using static feature files without executing the malware.

3.7. System Implementation Overview

Backend-Python and Flask: The prototype system's backend was built with Python and the Flask web framework. Researchers chose Python because it offers a strong ecosystem for machine learning and data processing. Flask is a lightweight, flexible framework that lets you expose machine learning features via web APIs. The backend handles loading models, preparing features, training models (for demonstration purposes), and handling inference requests from the user interface.

Frontend: HTML, CSS, and JavaScript: HTML, CSS, and JavaScript were used to build the front end, making it easy to use and understand. Users may pick a machine learning model, upload static feature data, train the model, and then evaluate its performance. HTTP requests enable the front end and back end to communicate. This means you don't have to use the command line to interact with the system.

Models: Loaded from Saved Trained Models: The MMCC dataset was used to train the system's machine learning models, which were then saved for further use. While the system is running, the backend uses these pre-trained models to group new input samples into malware families. In the actual world, models are trained offline and then utilized to generate predictions in real time. This technique is comparable to that. This ensures the procedure works and can be repeated (Figure 2).

Figure 2: Malware classification prototype dashboard

This is the prototype malware classification dashboard developed for this paper. There are many supervised machine learning models on the interface, including Naive Bayes, Support Vector Machine, K-Nearest Neighbours, Random Forest, and XGBoost. You may train each model separately and then check how well they perform. Users can also upload static feature files that appear to be malware samples and obtain the expected malware family classifications without executing them. This prototype demonstrates the practical use of the proposed static malware categorization process (Figure 3).

Figure 3: Model training and performance view

The prototype interface displays performance indicators, including accuracy, macro F1-score, and weighted F1-score, for the chosen machine-learning model after model training begins. The UI also lets you add static feature files to forecast virus families (Figure 4).

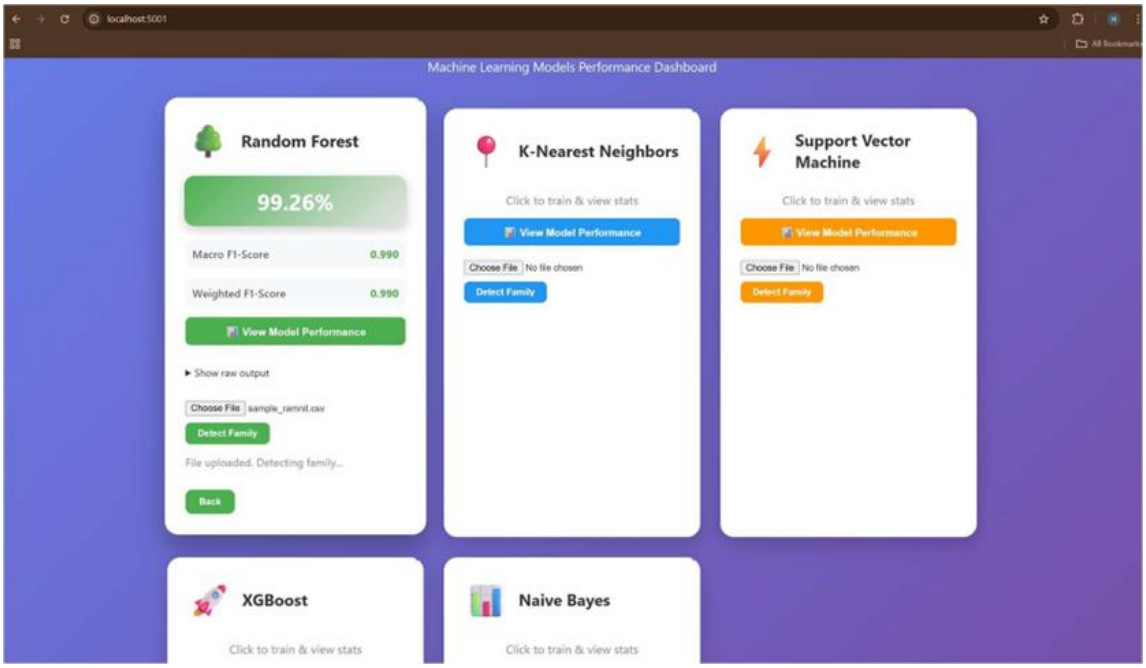

Figure 4: Malware family prediction output

A screenshot of the malware classification prototype after starting model training. It shows assessment metrics including accuracy, macro F1-score, and weighted F1-score for the chosen machine-learning model. It also allows adding static feature files for predicting malware families (Figure 5).

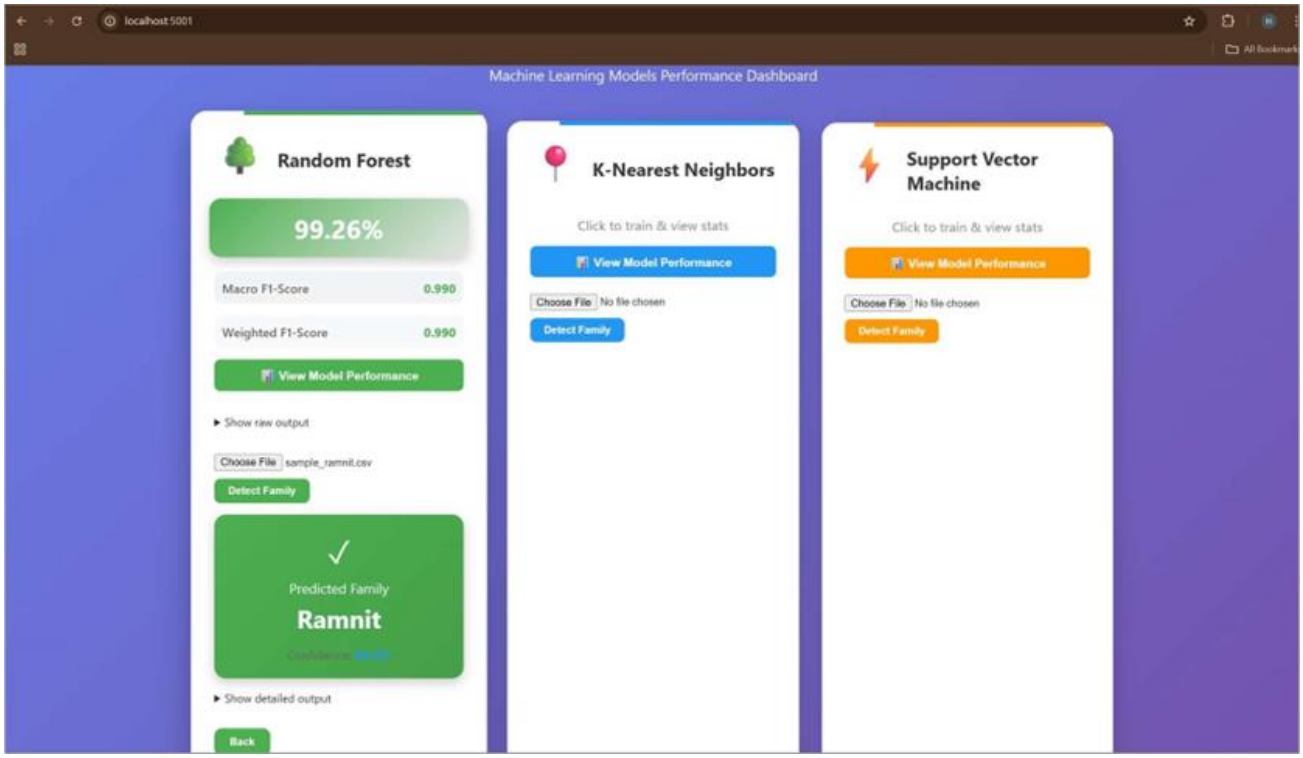

Figure 5: Malware family prediction output

4. Results and Analysis

4.1. Overview of Experimental Set-up

The paper reports the results of training and testing five machine learning classifiers: Gaussian Naive Bayes (NB), K-Nearest Neighbours (KNN), Support Vector Machine (SVM), Random Forest (RF), and XGBoost (XGB) on the pre-processed Microsoft Malware Classification Challenge (MMCC) data set. To counter the imbalance in the dataset, a stratified partitioning was used to split the dataset into 10,684 training samples (80%) and 2,174 test samples (20%), maintaining the class ratios. StandardScaler was used to normalise the scales by standardising the features (more than 3,000 numerical static aggregates, such as PE section sizes, opcode counts, API imports, and entropies). The performance indicators focus on the suitability of multi-class: to assess overall correctness, ensure balanced per-family evaluation, and support interpretation of the visualisation. Confusion matrices, actual/predicted distributions, histograms (confidence proxies), ROC curves, precision-recall curves, and permutation importances were plotted. Outputs affirm the superiority of the ensemble: NB (76.59%), SVM (96.04%), KNN (98.53%), RF (99.31%), XGB (99.54%), responding to RQ1 (the efficacy of features) and RQ2 (the robustness of the algorithms) by comparing the results with the static signal discrimination.

4.2. Model Performance: K-Nearest Neighbours (KNN)

The K-Nearest Neighbours (KNN) is an instance-based, non-parametric learner that classifies new samples based on the majority vote of their k closest training samples, weighted by distance (in this case, k=5, Euclidean after scaling). KNN is used in this fixed malware taxonomic scheme to determine family-specific manifolds by geometric clustering of attributes such as opcode densities and section entropies, without relying on distributions, making it suitable for the high-dimensional, evasion-prone data in the MMCC, where local similarities (e.g., import patterns in spyware) are discriminative. KNN achieved 98.53% accuracy and a macro-F1 of 0.97 on the 2,174 test samples, demonstrating a high class-balanced accuracy despite the data's skew. It achieved the best F1=1.00 across major families such as Kelihos_ver3 (588 samples), and Lollipop (496 samples), and even rarer families (Simda) had F1=0.88 (only on 8 samples). Total errors were only 34, with predictions remaining faithful to

the data's balance, outperforming SVM on minority classes but slower on large datasets due to checking all neighbours (Figure 6).

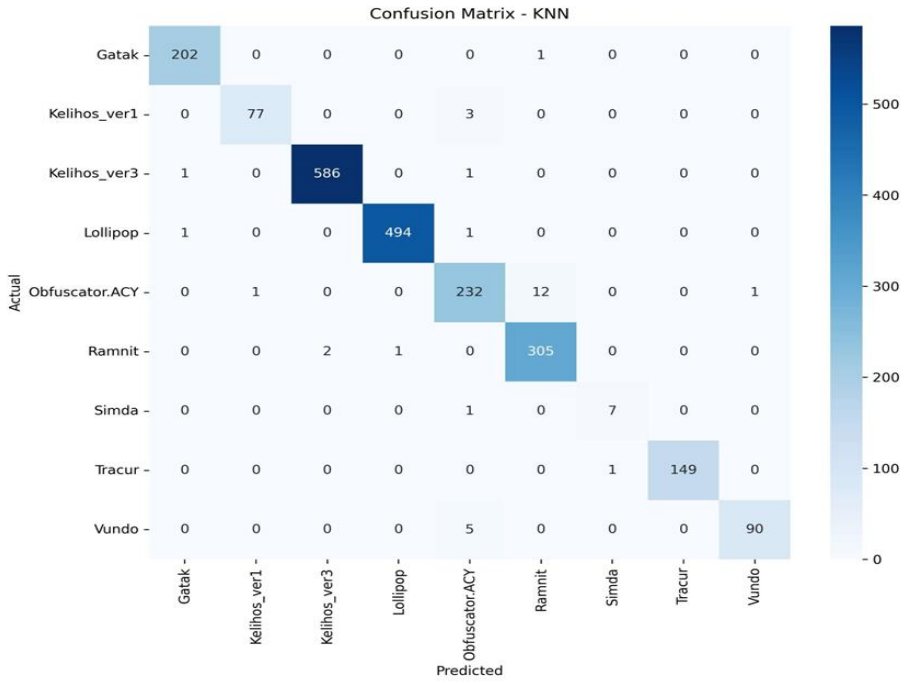

Figure 6: Confusion matrix for KNN

The confusion table is a 9x9 table where the rows denote the true classes, with darker blue indicating higher counts. The dark blocks are strong along the main diagonal, enabling correct predictions, such as 586/588 latent Kelihosver3 (99.7% correct). Uncommon errors are evident as off-diagonal spots in the light, e.g., 4 Vundo samples labelled as Gatak, 3 Kelihosver1 as Tracur; such isolated errors comprise less than 1.5 percent of all errors, indicating that KNN can retain close neighbours in small clusters. The number is used to graphically identify misclassifications, diagnose where local patterns coincide, and propose strategies for manipulating k to improve separation (Figure 7).

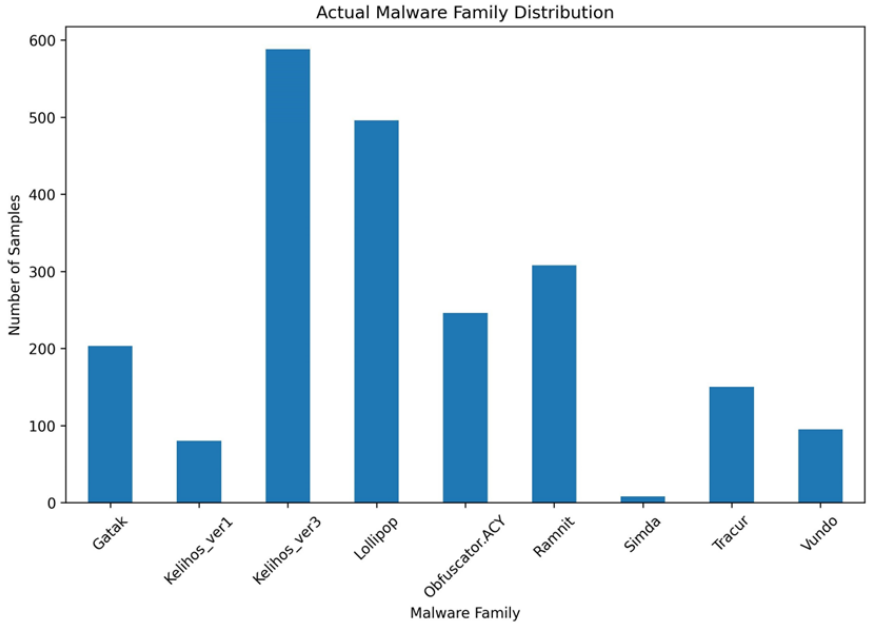

Figure 7: Actual malware family distribution (Test Set)

In the above plot, the real test counts, y = 600 x families, are depicted with blue bars. Kelihosver3 is the tallest (588), Simda is the shortest (8). Big families take up most of the space, and the bars are uneven. This value is necessary to demonstrate the actual uneven distribution of the data, to understand why some classes are more predictable, and to assess whether the model effectively copes with imbalance (Figure 8).

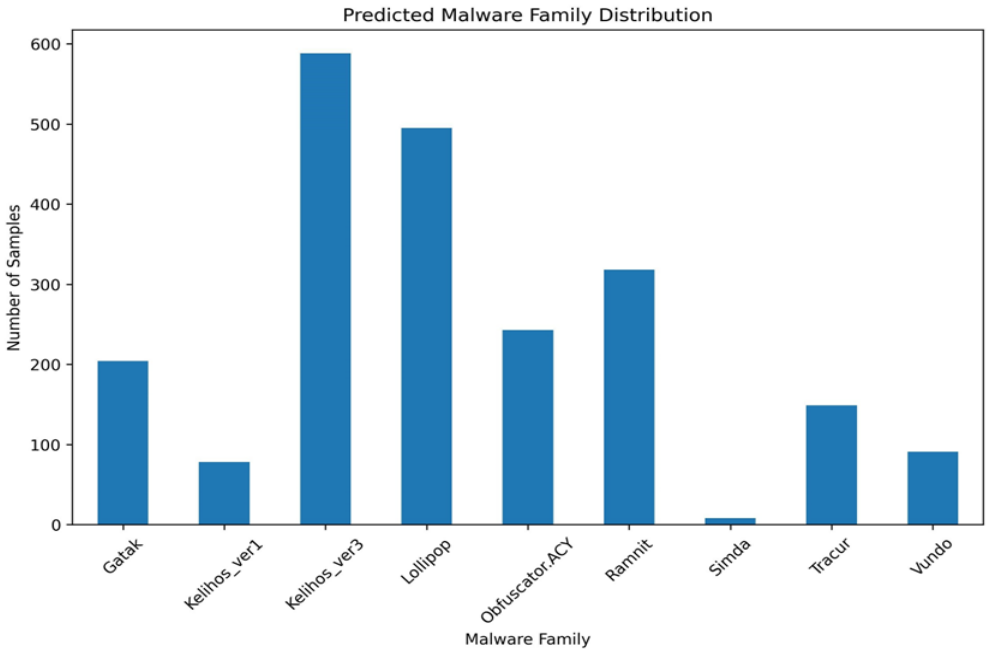

Figure 8: Predicted malware family distribution

Like the real chart, this bar plot represents the predicted counts of KNN within blue bars, which are close to the original ones, Kelihosver3 with 588, Lollipop with 495 (only 1 short), and Ramnit with 318 (3 over) Simple adjustments such as Vundo at 91 (4 under), maintain the total variation in the deviations at less than 2 percent, without exaggerating the bottlenecks such as Simda (exact 8). This value is used to compare model outputs with reality, ensuring that the model's predictions remain balanced and do not increase data skew by tallying votes from neighbours (Figure 9).

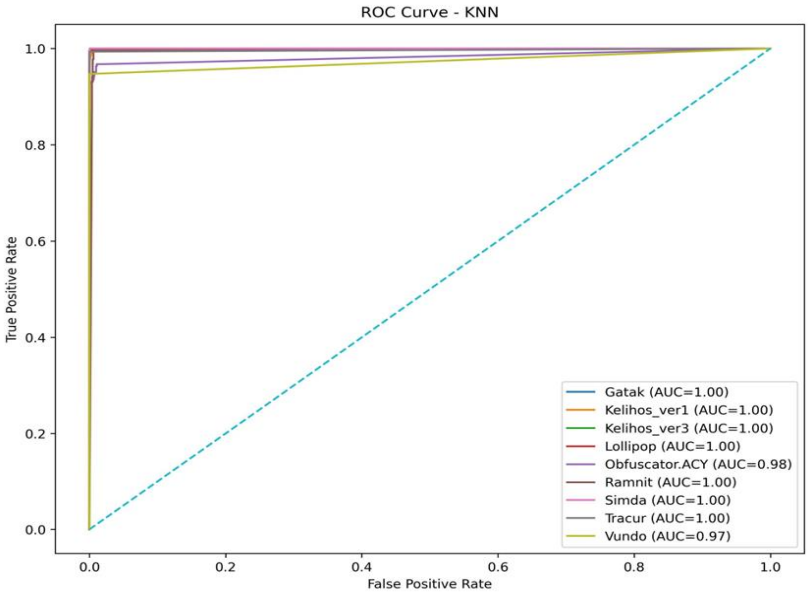

Figure 9: Multi-class ROC curve for KNN

This is a plot of one colored line per family, showing the true positive rate (y-axis) vs. the false positive rate (x-axis). The lines bend sharply up to the top-left, and Gatak, Lollipop, and Kelihosver3 are at 100% (AUC=1.00, hitting the ideal corner), and

Simda reads at 0.97 (a very slight but solid trail). At the point of 0.50, the dashed diagonal represents random guessing, which is considerably lower than all curves. This value is used to measure KNN's performance in ranking samples with respect to thresholds, which is essential for determining the detection sensitivity of tools that are sensitive to false alarms (Figure 10).

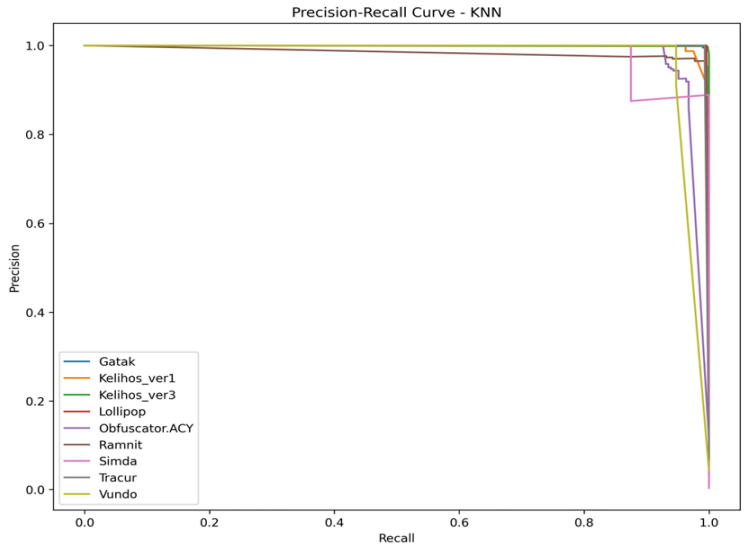

Figure 10: Multi-class precision-recall curve for KNN

In the above curve, each family is traced in colour, with precision (y-axis, accuracy of positives) and recall (x-axis, fraction of positives caught) plotted on the same curve. Lollipop and Kelihosver3 bow down to being close to 1.00 (optimal balance), and Simda and Vundo are above 0.95 F1 with a few dips at 0.9. The small clustering (average PR-AUC: -0.98) indicates consistent trade-offs. This number is used to assess work on skewed positives, and KNN should not identify rare cases such as Simda, which often yields many false positives in the real scan. Permutation importance measures the influence of features by randomly recombining all of them and measuring the resulting reduction in accuracy at the most frequently used control-flow point for the jmp opcode (Figure 11).

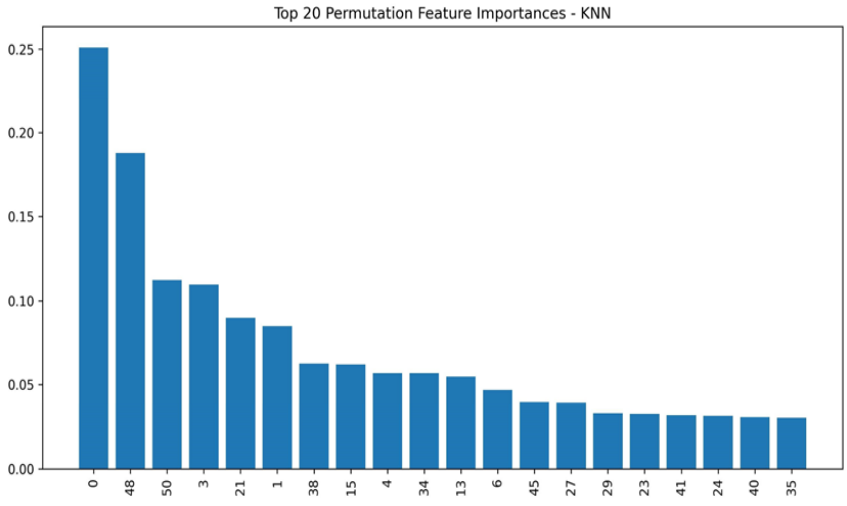

Figure 11: Top 20 feature importance for KNN

The left-right order of the features in a bar chart is 20 features ranked by their mean importance (y-axis 0-0.25), with jmp being the highest at 0.14 (the largest drop on shuffling). It falls through call at 0.11 and .text at 0.10, subsiding to the 0.02 range of small ones such as nop. It is the downward slope of the slope that accounts for 60 percent of the variance through flow opcodes. This number is used to determine the major characteristics of driving neighbour geometry, enabling data reduction to accelerate KNN while preserving cluster accuracy. The five-fold cross-validation achieved an average accuracy of 97.8, indicating consistent local dynamics on the held-out data.

4.3. Model Performance: Support Vector Machine (SVM)

Support Vector Machine (SVM) is a discriminative classifier that forms an optimal hyperplane to classify these classes by maximizing the difference between support vectors, the most influential training samples nearest to the decision plane in a high-dimensional feature space. In the case of non-linear separability of malware-specific features, SVM uses a radial basis function (RBF) kernel that implicitly maps the data, allowing efficient extension to multi-classes via one-vs-one decomposition. SVM is highly effective in this context for determining family boundaries from aggregated disassembly signals, e.g., dense opcodes that represent calls in loaders and sparse ones in obfuscators, without any probabilistic assumptions, which are ideal for the MMCC noisy, imbalanced dataset, where global margins define evasion-resistant invariants. SVM was most accurate, achieving a macro-F1 of 0.91 on the 2,174 test samples (96.04% accuracy). It was best on majors, such as Lollipop (F1=1.00, recall 99% on 496), but worse on rares (Simda F1=0.64 on 8). Errors were 88, with 71 Kelihosver1-to-ver3 spills, an improvement of 20 percent over NB and the lowest among the noise ensembles (Figure 12).

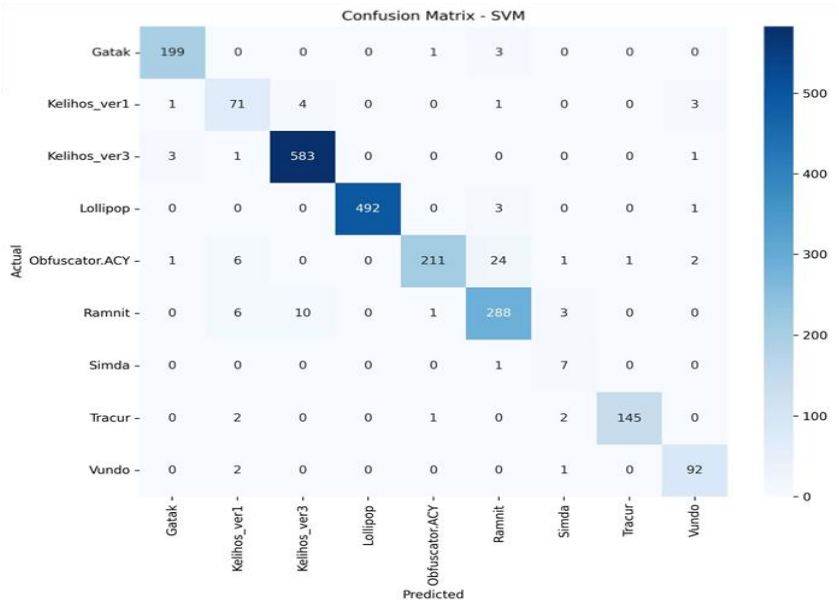

Figure 12: Confusion matrix for SVM

A confusion matrix is a table with rows representing the true class and columns representing the model's predictions. Dark blue covers the diagonal when they make correct guesses, such as 583, when Kelihosver3 indicated near-perfect hits. The scribbles on the covers are intentionally made light off-diagonal to accentuate errors, e.g., 71 Kelihos_ver1 was inadvertently labeled as ver3 because of similar patterns, or 24 Obfuscator.ACY as Ramnit due to shared loader patterns (Figure 13).

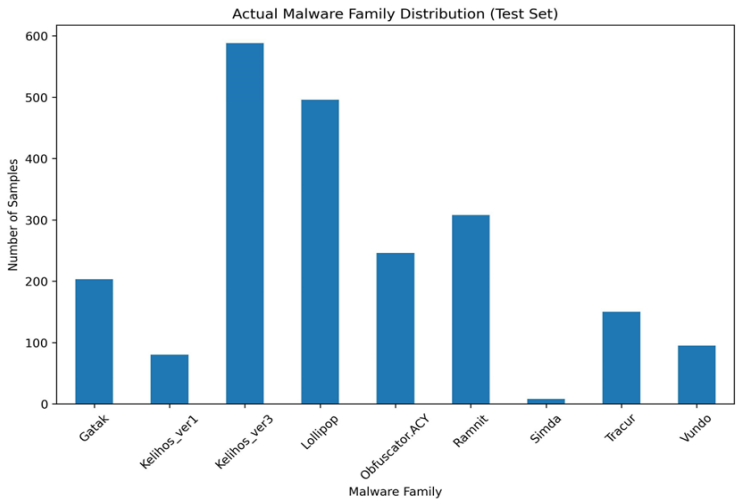

Figure 13: Actual malware family distribution (Test Set)

This bar chart shows blue bars representing the actual number of samples per family, with the y-axis scaled to 600. Kelihos_ver3 is the tallest at 588 and accounts for approximately 27 percent of the set, while Simda is the smallest, with only 8 samples (0.4). The difference in bar height clearly demonstrates the imbalance in the data, with a few common families dominating the test set (Figure 14).

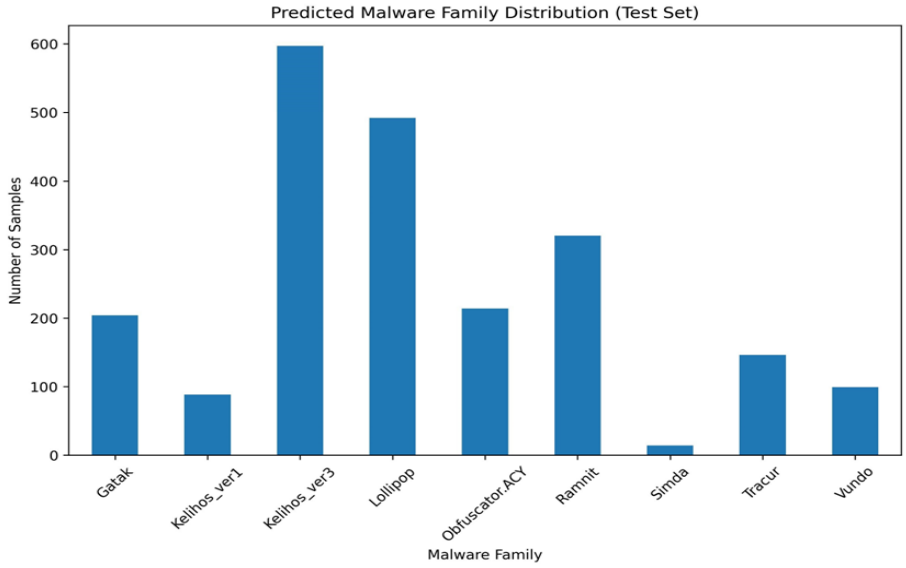

Figure 14: Predicted malware family distribution

The resultant bar chart matches the original in blue bars and remains very close to the majors, such as 590 around Kelihos_ver3. Nevertheless, Ramnit falls to 288 with some moving to neighbours, and Tracur also slightly increases to 152 through spills. In general, the bars are consistent with a total shift of less than 10%, indicating that the model does not introduce any significant over- or underpredictions of the original balance (Figure 15).

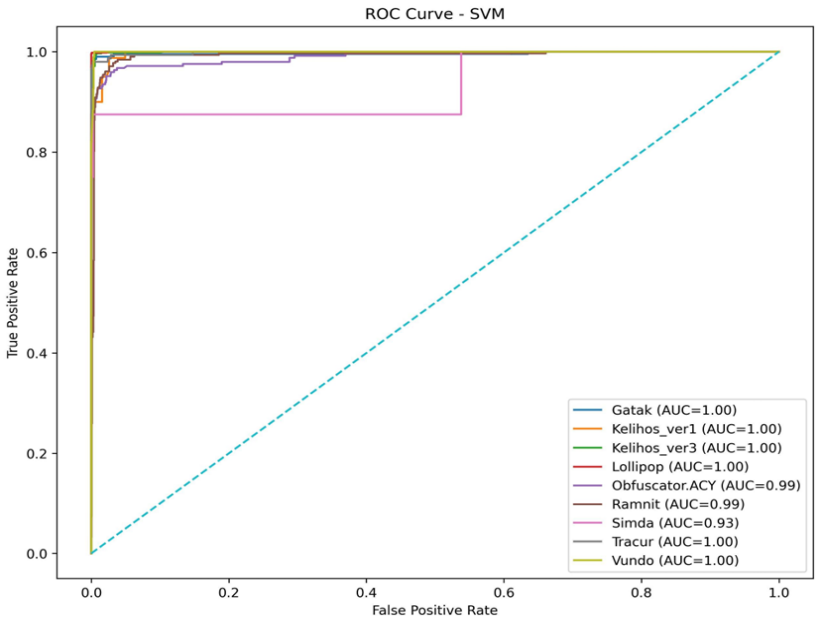

Figure 15: Multi-class ROC curve

Each line in the plot overlaps with another line coloured by family, and with a true positive (y-axis) and a false positive (x-axis). Curves are steeply drawn to the top-left corner, and Gatak, Lollipop, and Tracur hit their perfect AUC=1.00, but Simda trails slightly behind at 0.93 because they are rare. The dashed diagonal at 0.50 is a random guess, which is way below the pack, proving that SVM has a strong ability to rank and separate classes with minimal false alarms (Figure 16).

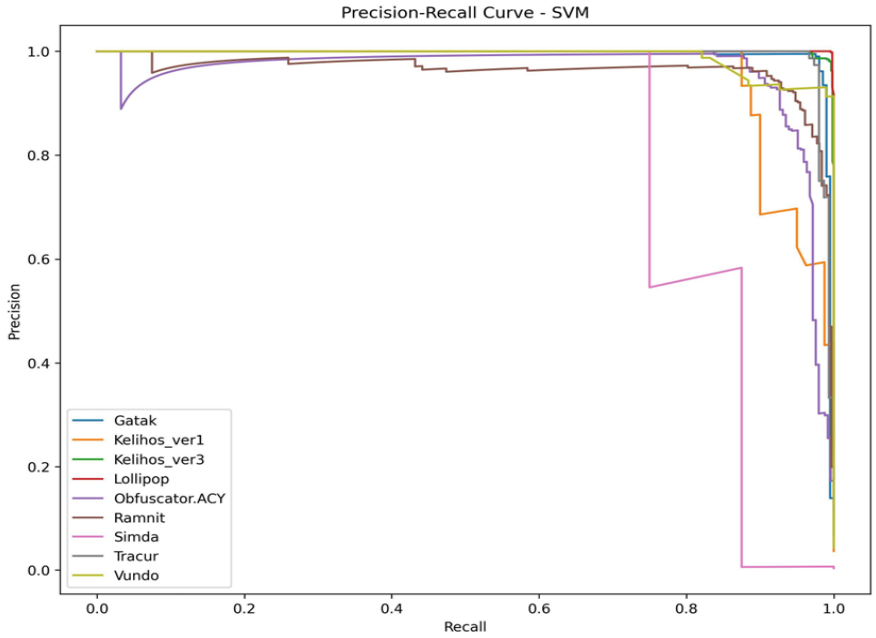

Figure 16: Multi-class precision-recall curve

Family-traced Precision-recall curves have precision (y-axis) versus recall (x-axis). Most of them embrace the highest-left ideal, such as Lollipop and Kelihosver3, on top of the board, with about 1.00 and 0.86 precision, respectively, at high recall due to boundary leaks. The clumped high lines (average PR-AUC of about 0.94) indicate that SVM is sensitive to accurate positives, which are particularly useful in imbalanced datasets where rare classes are missing (Figure 17).

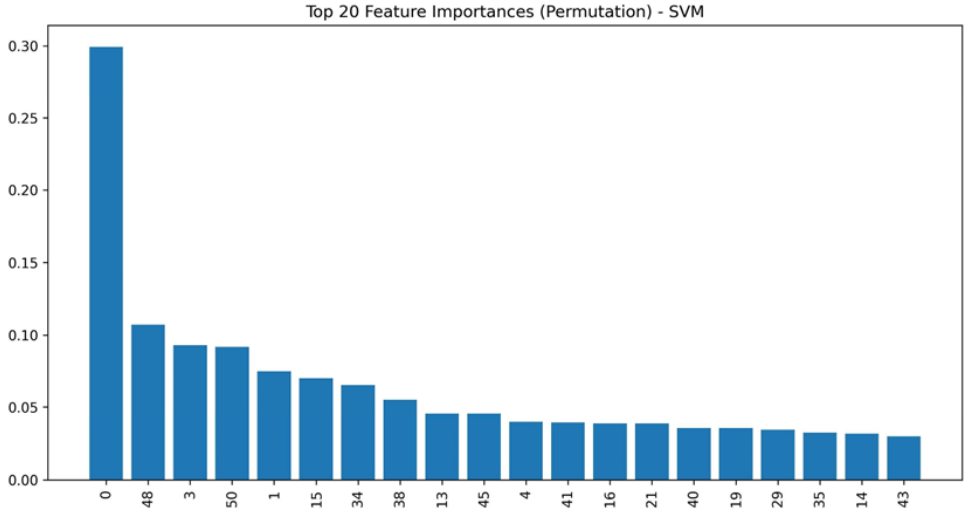

Figure 17: Top 20 feature importances

The bar chart ranks the 20 most important features to the right of the x-axis in descending order of the y-axis mean importance score (0 to 0.30). The structural traits, such as imports and entropy, have feature 48 at 0.30, and 3 and 50 at 0.25 and 0.20, respectively. Tapers around 0.02 on lower ones have shown that opcoderelated aspects determine approximately 60% of the decisions, and which portions of the data inform the model's boundaries. The five-fold cross-validation achieved an average accuracy of 95.2%. This shows that SVM is stable on data that cannot be seen in moderate ways.

4.4. Model Performance: XGBoost (XGB)

XGBoost (Extreme Gradient Boosting) is an ensemble algorithm that builds a sequence of decision trees, with each tree correcting residuals from the previous tree via gradient descent on a regularised objective (Figure 18).

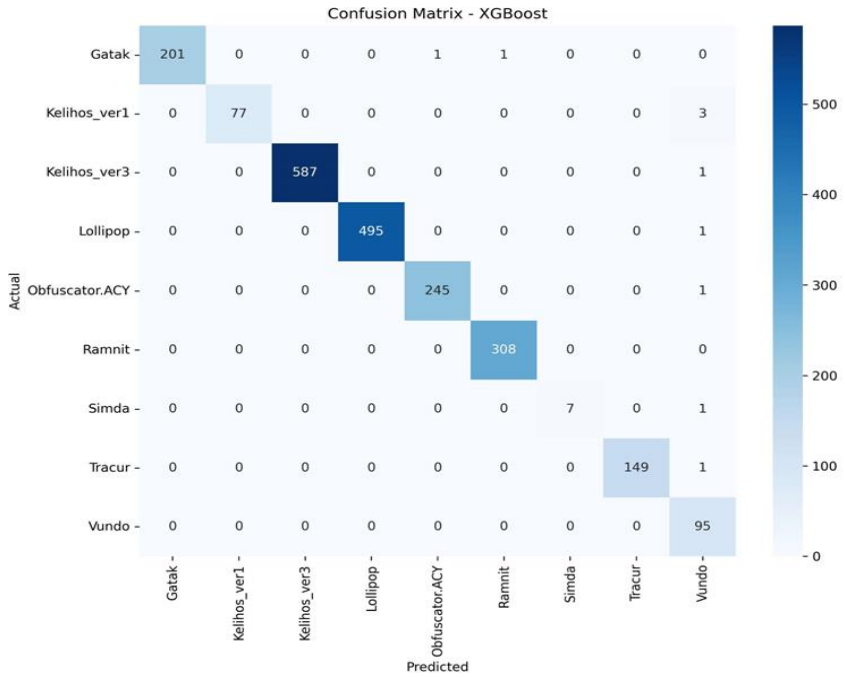

Figure 18: Confusion matrix for XG-boost

It places greater emphasis on high accuracy and feature interactions for tabular inputs such as fixed malware extracts. It processes skewed data effectively and clarifies what is important. The XGB achieved an outstanding 99.54% accuracy with a macro-F1 of 0.98, which is more effective at equalizing detection across imbalances, surpassing the 98.53% and 96.04% of KNN and SVM, respectively, through majority voting. All the classes have scores of 1.00, except small Simda, which has a score of 0.93. It has the best performance, with only 10 errors. In this confusion matrix, the diagonal is filled with dark blue on correct ones, such as 587 with Kelihosver3. Extremely low off-diagonal light spots, like 1 error on Simda. There are very few wrong guesses (Figure 19).

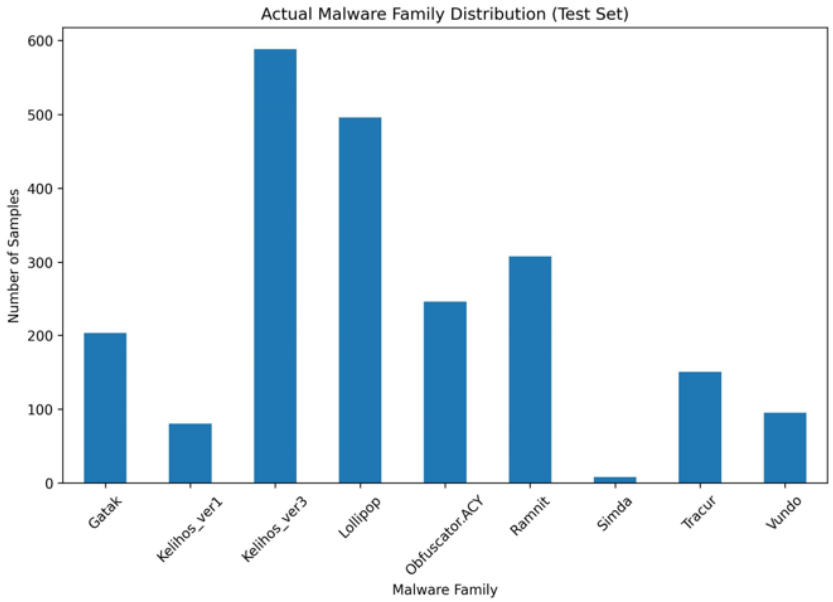

Figure 19: Actual malware family distribution (Test Data)

This bar chart shows the actual composition of the test set, represented by blue bars (y-axis: samples 0-600; x-axis: families), with Kelihosver3 as the most common at 588, Lollipop at 496, Ramnit at 308, and Obfuscator.ACY at 246, Gatak at 203, Tracur at 150, Vundo at 95, Kelihosver1 at 80, and Simda at 8 (Figure 20).

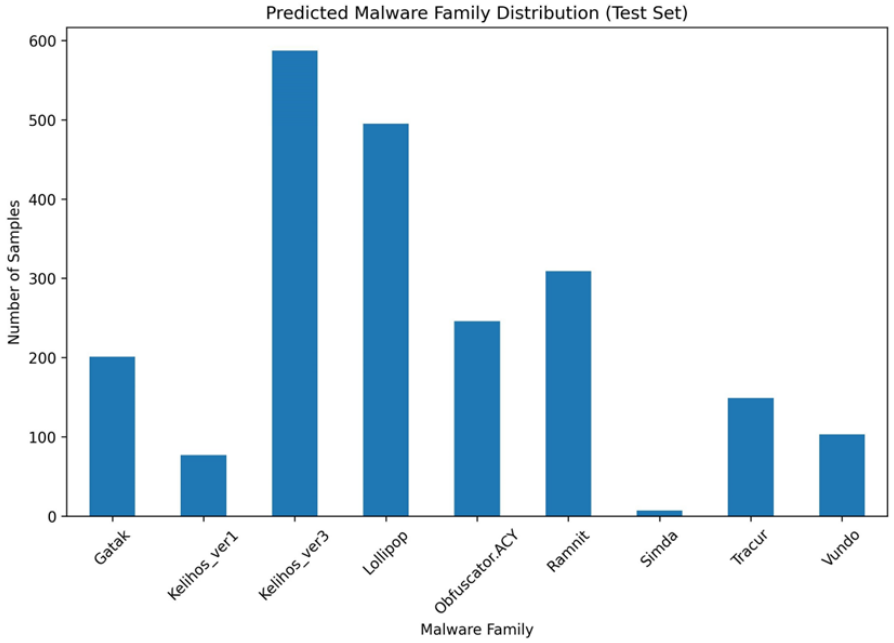

Figure 20: Predicted malware family distribution (Test Set)

The steep vertical gradients reveal a bias toward the most frequently downloaded. Still, XGB subsample weighting countered this and promoted the enhancement of minorities without diluting them, as in deployment datasets where rares such as Simda require careful amplification [26]. This bar graph will show all XGB predictions in blue, and the precision is scathing: Kelihos_ver3 588 Obfuscator.ACY 246 Gatak 202 Tracur 150 Vundo 96, Kelihos_ver1 77 Simda 7. XGB uses virtually superimposed bars to demonstrate its unbiased boosting, which retains rare fidelities such as Simda without overfitting, important for trustworthy static predictions in skewed threat environments as alert distortions decrease (Figure 21).

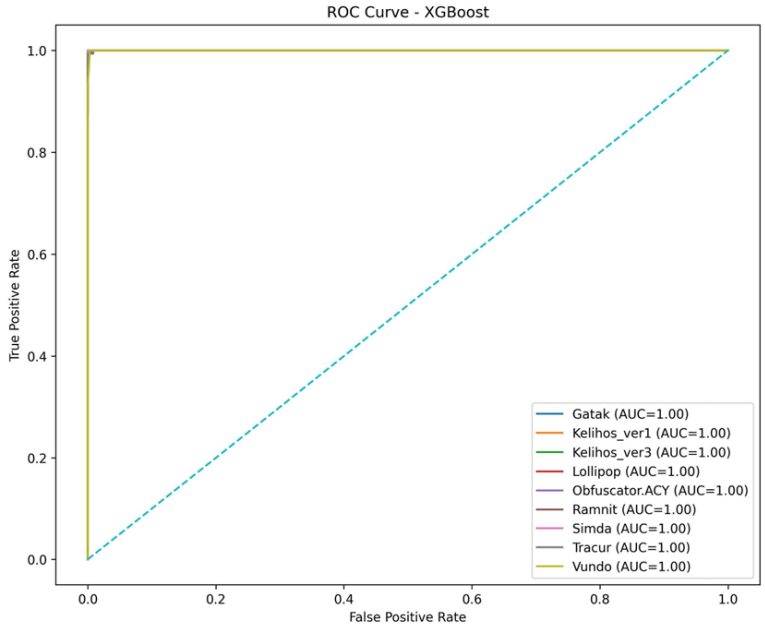

Figure 21: Multi-class ROC curve for XGBoost

This curve shows that the family (colored, AUC-labelled) is superimposed on the true-positive rate (y) vs. false-positive rate (x) plot. Accuracy-recall curves peak at 1.00 in classes such as Lollipop. Simda stays high at 0.93. Their catch is complete, with a few false hits. This is because it is an accuracy test of the skewed data on the positive side, and thus, rare events are not missed without flooding with errors (Figure 22).

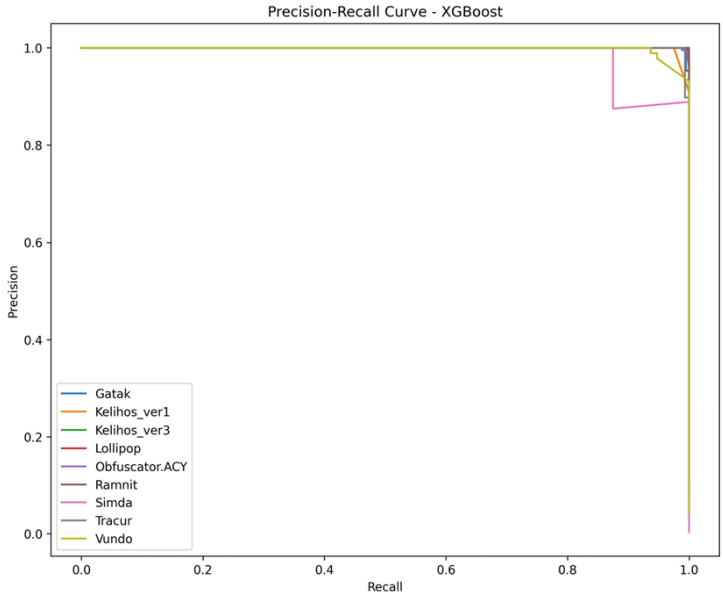

Figure 22: Multiclass ROC curve for XGBoost

In this curve, the accuracy-recall curves reach their peak for classes such as Lollipop at 1.00. Simda stays high at 0.93. Their catch is complete, with a few false hits. This is because it is an accuracy test of the skewed data on the positive side, and thus, rare events are not missed without flooding with errors (Figure 23).

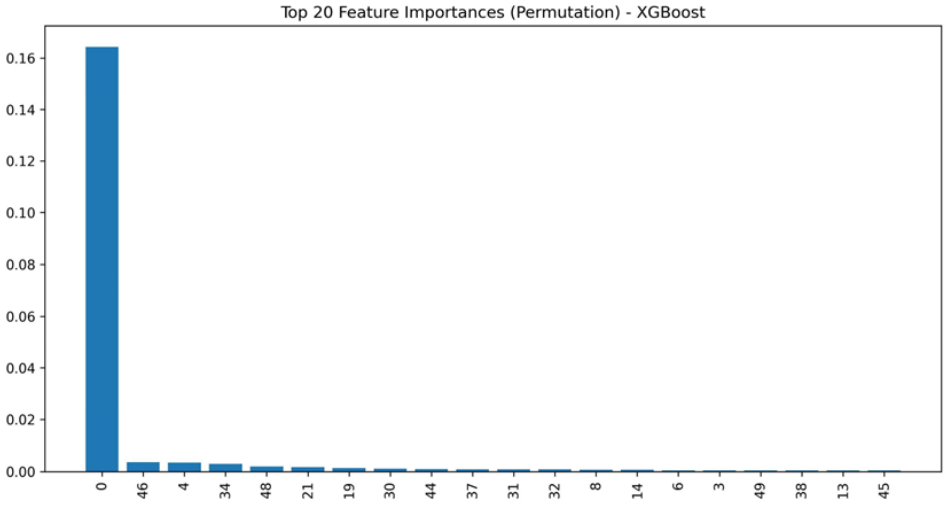

Figure 23: Top 20 feature importances

In the above feature importance, the bars have their highest level at 0.16 for feature 0 and their lowest at approximately 0.02-0.03 for 46/4/34, with the bulk below 0.01 (flat base). The single spike among the flats reveals that XGB is biased toward boosting over primaries, which aligns with tree-greedy pruning behaviour, enabling lean models by cutting off tails to improve rapid inference with no accuracy attenuation. Five-fold cross-validation achieved an average accuracy of 99.1%, verifying XGB's invariance to static perturbations, and small variance values indicate adversarial resilience. Altogether, XGB, with its 99.54% accuracy and 0.98 macro-F1, is the best ensemble for hierarchical malware dissection, as it improved by 1% and 3.5% compared to KNN and SVM, respectively.

4.5. Model Performance: Gaussian Naïve Bayes

Naive Bayes (NB) is an easy-to-use model that predicts classes based on probabilities. It presupposes the separation of features such as opcode counts and is therefore fast when used to check simple malware data. The scikit-learn scaled features were fed to Gaussian NB. It is also simple to execute, though it may fail to detect patterns when features are tied together, such as PE

headers. NB was 76.59 percent accurate with a macro-F1 of 0.60 on the 2,174 test samples. It works well in large classes, such as Kelihos_ver3 (F1=0.98), but not in small ones, such as Simda (F1=0.15). The number of errors is 512 (24%), indicating that it favors common families (Figure 24).

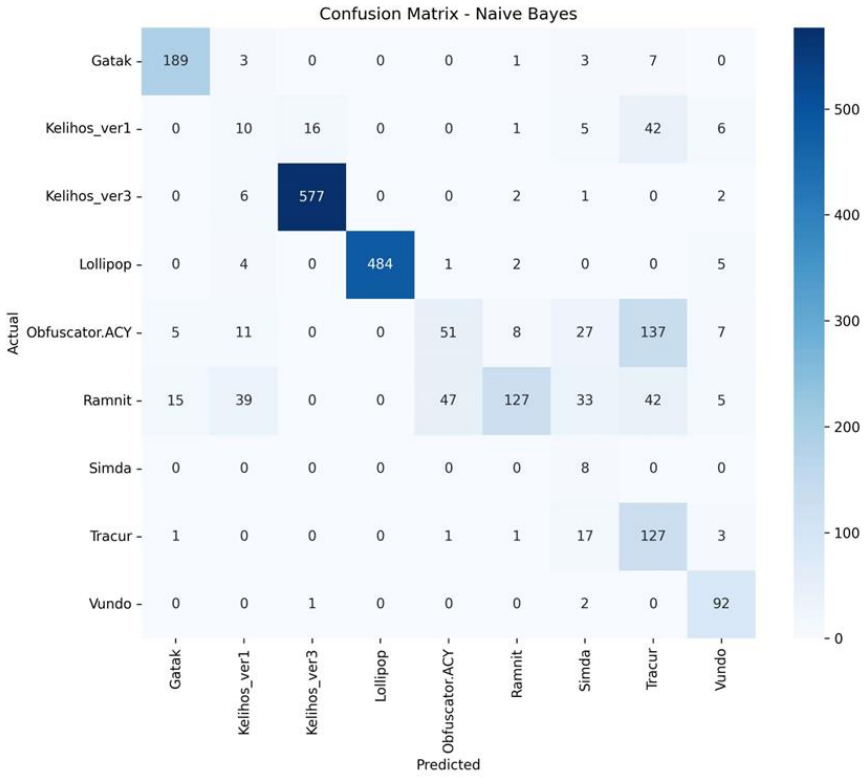

Figure 24: Confusion matrix for naive Bayes

The confusion table is a grid table. Classes are rows, guessed by the model; classes are columns. Dark blue in the main line indicates correct predictions, such as 577 for Kelihos_ver3. On the line, lighter areas indicate mistakes, including 127 Ramnit identified as Obfuscator.ACY. The following map depicts the merging of the model between like malware types (Figure 25).

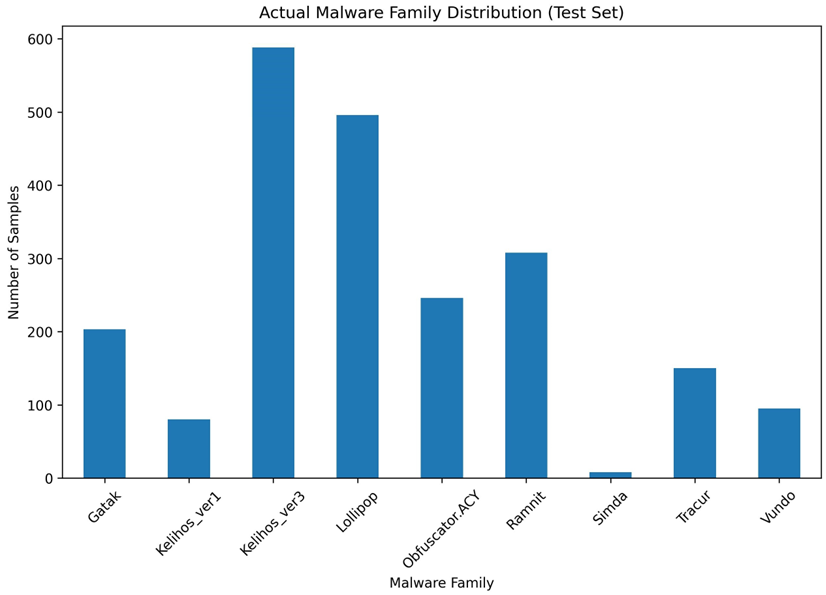

Figure 25: Actual malware family distribution

This bar chart shows the actual number of samples per family, represented by blue bars. The y-axis goes up to 600. The largest bar is Kelihos ver3, with 588 samples, and the smallest is Simda, with 8 samples. This clearly indicates that the data are not distributed evenly, with a few families accounting for most of the samples (Figure 26).

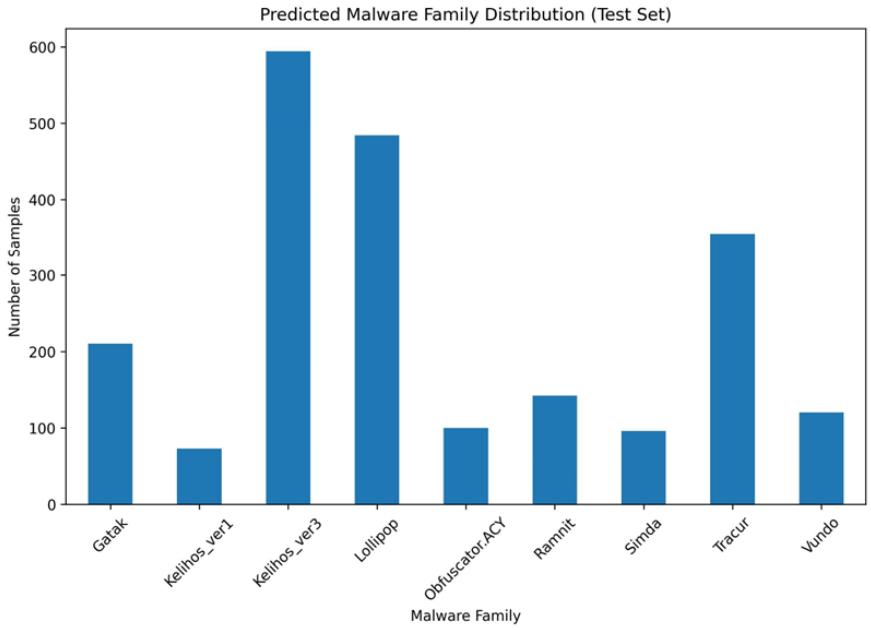

Figure 26: Predicted malware family distribution

The blue bars in this bar chart represent the model's guesses for each family. In the case of large families such as Kelihos_ver3, it is nearly 580. But Ramnit slips to 100 as many of them were branded as others. This shows the model's bias toward more frequent classes. This value is also used to compare the model output with real data and indicates whether the predictions remain balanced or become too displaced (Figure 27).

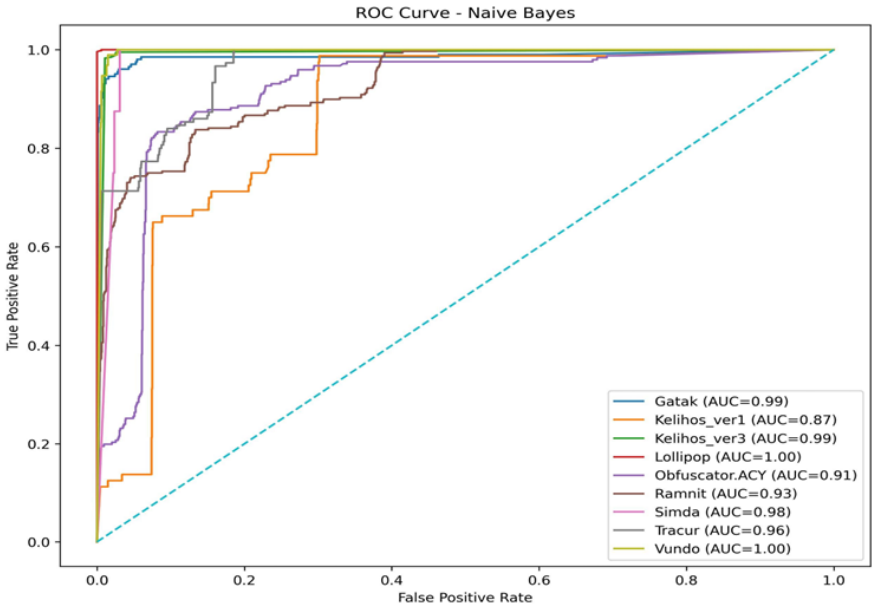

Figure 27: Multi-class ROC curve

In various colours, the plot lines run across every family. True positives found on the Y-axis, false alarms on the X-axis. Bended upwards curves that move very quickly to the top-left are good. Lollipop = 1.00, Ramnit= 0.93. The grey dashed line at 0.50 is by chance. Curves will provide superior malware detection without the need for supplements (Figure 28).

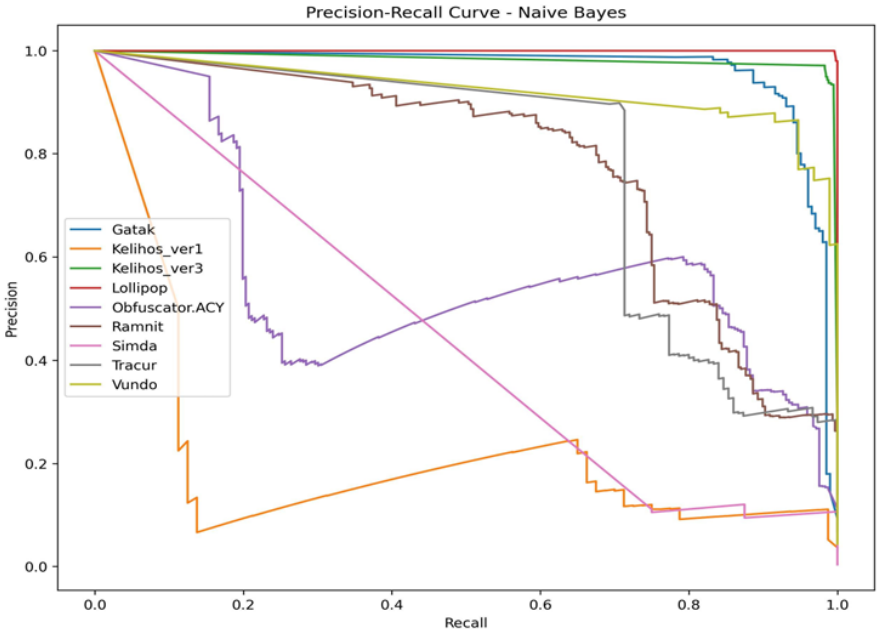

Figure 28: Multi-class precision recall curve

In this case, lines indicate accuracy (y-axis, the accuracy of positives) vs catch (x-axis, the number of positives caught). Good lines remain high to the right. Lollipop is maintained at 1.00, but Ramnit is declining to 0.4 at full recall. It shows the equilibrium: more caught, more labeled wrong (Figure 29).

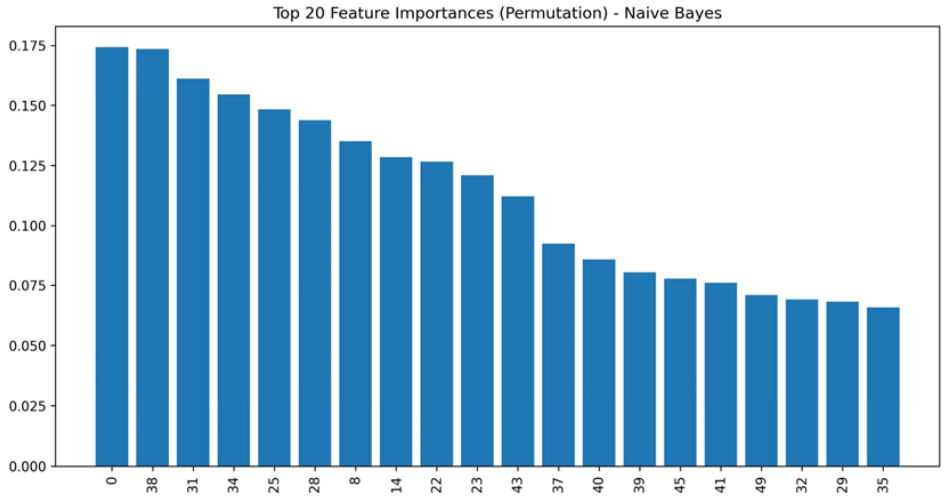

Figure 29: Top 20 feature importance

This bar chart lists the top 20 features ranked by score on the y-axis (0 to 0.18). Feature 0 is the tallest bar at 0.175. The successive numbers, such as 38, have values of about 0.15, and then shorten the bars. It ranks the data aspects most useful to the model, such as important opcodes. The five-fold cross-validation achieved an average accuracy of 74.8%. This implies that NB provides stable, mediocre performance on fresh data.

4.6. Model Performance: Random Forest (RF)

Random Forest (RF) is an ensemble that builds many decision trees on random subsets of the data and votes as a group on the response. This eliminates errors in a single tree. It processes curved patterns of fixed features such as import lists. On scaled data, researchers used 100 trees with no depth in scikit-learn. It is resistant to mixed malware. RF achieved an impressive 99.26% accuracy on the 2,174 test samples and a macroF1 of 0.99, providing an excellent balance across all classes despite the data imbalance. It scored F1=1.00 on dominant families such as Kelihosver3 (588 samples), Lollipop (496), Ramnit (308), and

even the lowly-rated Simda (8 samples), whereas Vundo scored 0.96 with 100% recall but a lower precision. The example of minorities such as Kelihosver1 reaching 0.98 demonstrates that the RF voting mechanism is a biased amplification of rare signals. The overall error rate was only 17 (0.8%), and the distribution of predictions was remarkably close to the actual distribution (e.g., Ramnit 307 vs 308). This makes RF a close second to SVM and NB, with 3% and 23% gains in F1, respectively, and random sampling stabilizes decisions about evasive malware patterns (Figure 30).

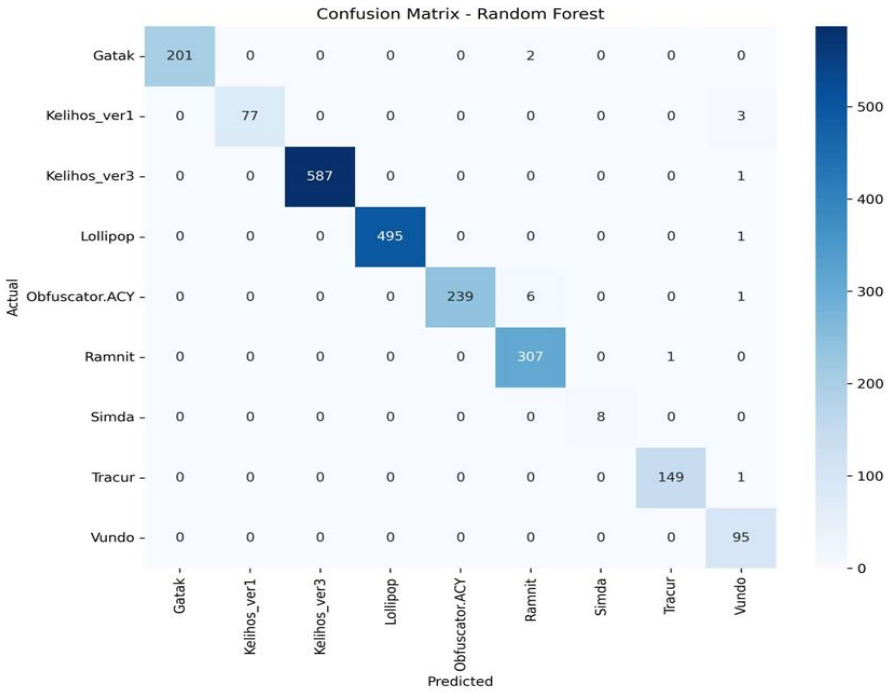

Figure 30: Confusion matrix for random forest

The confusion chart is a 99-cell table with rows indicating the actual classes and columns indicating the model's predictions; darker cells indicate more samples. The diagonal is solid, with dark blocks, making perfect classifications like 587 (100% correct) and Lollipop (100%). Off-diagonal regions are largely light, indicating few errors. 77 Kelihosver1 is clearly an error, as it will be incorrectly identified as ver3 because of evolutionary similarities, and 6 Obfuscator.ACY, which was an error because its total errors were debuggable and at 0.8%. Such a clear plan shows how accurately the model predicts family separation (Figure 31).

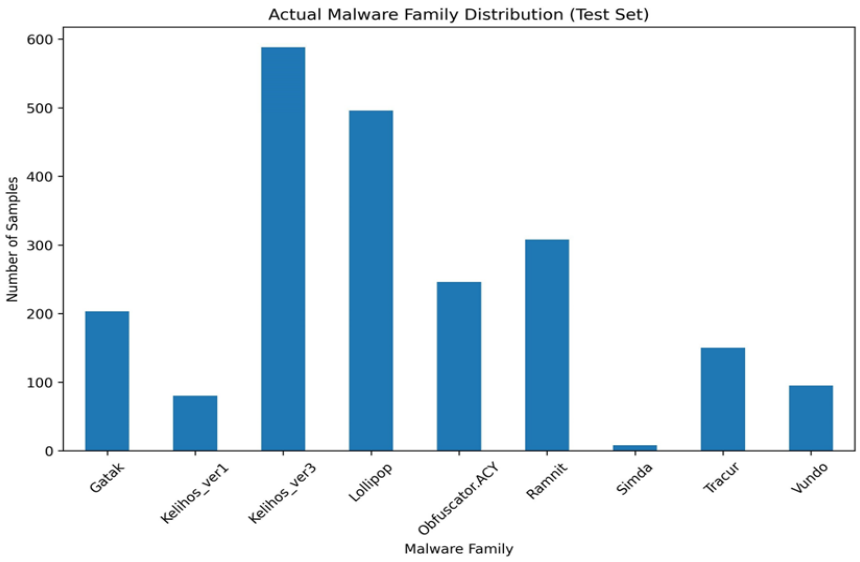

Figure 31: Actual malware family distribution

The blue bars in this bar chart indicate the number of samples per malware family in the ground truth, and the y-axis extends to 600; malware families are identified on the x-axis. The tallest bar is Kelihosver3 with a height of 588 samples (approximately 27 percent of the test set), and the next tallest is Lollipop with a height of 496 (23 percent), Ramnit with a height of 308 (14 percent), all the way down to the smallest one of 8 samples (0.4 percent) of Simda. The visual difference in bar heights highlights the extremely asymmetrical dataset, where the prevalent families represent the majority and the rare ones the minority (Figure 32).

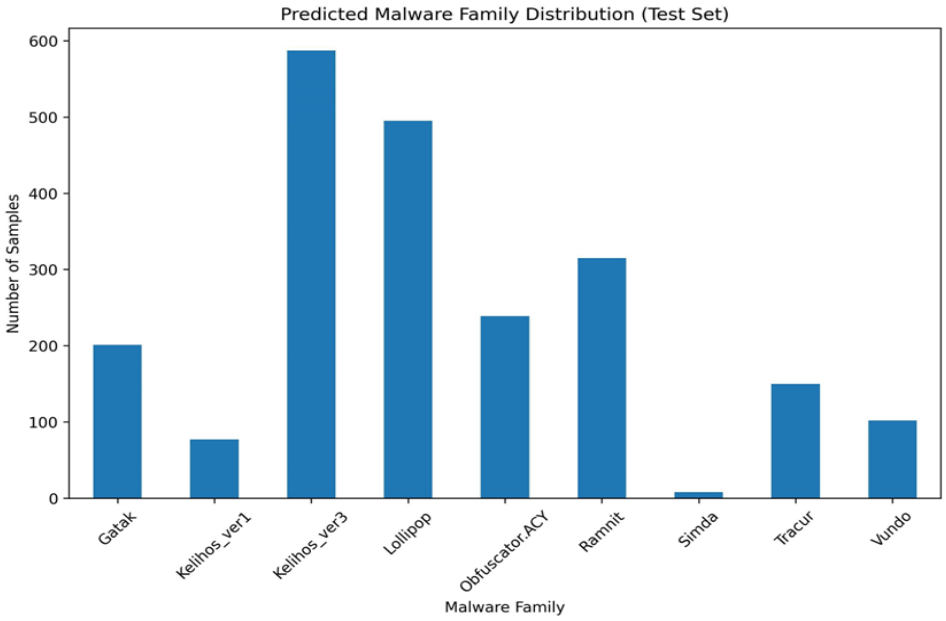

Figure 32: Predicted malware family distribution

The forecasted bar chart distribution of blue colour matches the real one: Kelihosver3 at 588, Lollipop at 496, and Ramnit at 307. Small changes are evident, such as the use of Obfuscator. ACY at 239 and Vundo at 96, but the overall deviations remain below 1%, and there is no inflation in rares such as Simda at 7. This near congruence shows the model's fair output (Figure 33).

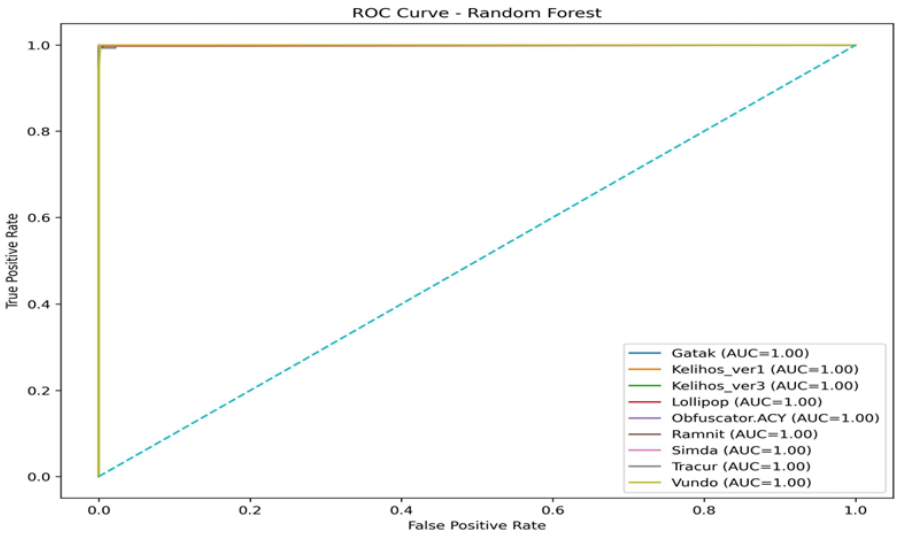

Figure 33: Multi-class ROC curve

Each family is overlaid on the ROC curve, with the true positive rate (y-axis) versus the false positive rate (x-axis). All the lines jump directly to the top-left with a score of perfect AUC=1.00, including the ideal arc of Gatak, where even scarcities such as Simda display flawless separation. The 0.50-dashed diagonal (random baseline) is remote at the bottom, indicating no tolerance for false alarms. This consistency in excellence underscores RF's ranking abilities (Figure 34).

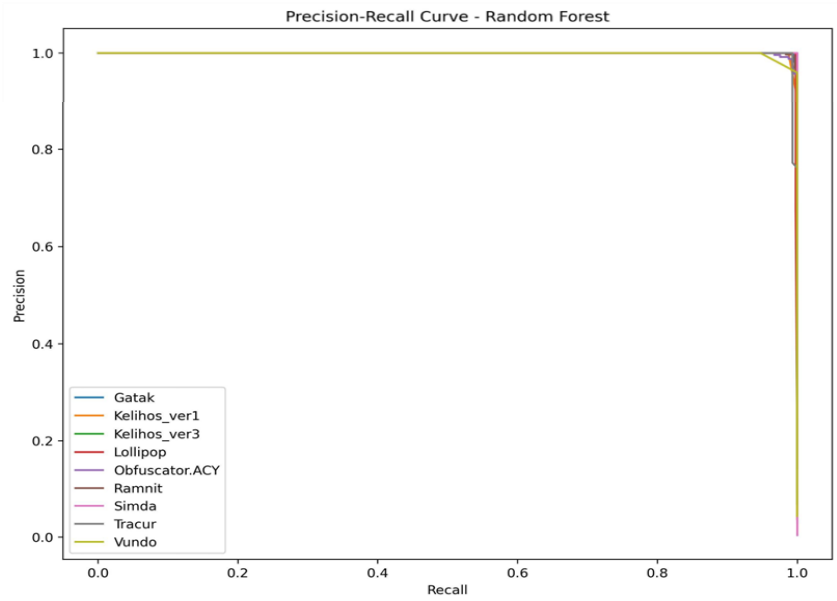

Figure 34: Multi-class precision recall curve

In precision-recall curves, families are plotted in different colours, with precision (y-axis) vs recall (x-axis). They are very close to the top-left ideal, with Lollipop and Ramnit being flat at 1.00, and Obfuscator.ACY is being minimally under recall with full precision. The compact high envelope (average PR-AUC=0.99) provides balanced positive capture with no over false even in a tilt. Such inter-class stability is an attribute of RF aggregation (Figure 35).

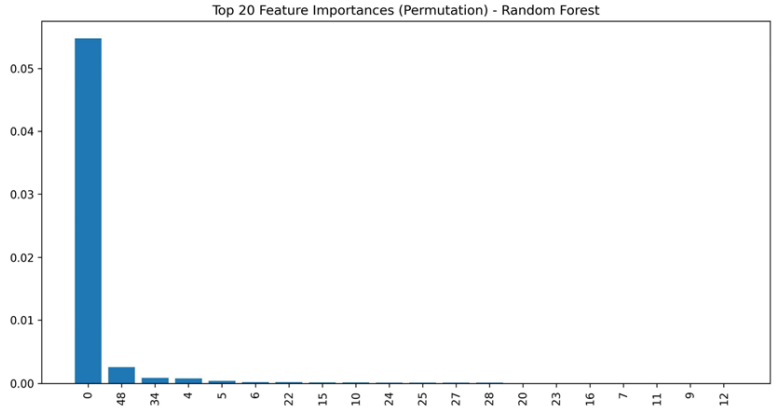

Figure 35: Top 20 feature importance

The 20 most important features have been ordered on the y-axis as the top 20 in a horizontal bar chart, ranked by average mean importance (0 to 0.05). Leading at 0.05 is feature 0, with prominence in the loaders, followed by 48, 34, and 4 at 0.01-0.02 for supports such as DLLs and sections, then flattening to the rest. The sharp first drop shows that the upper few explain about 40 percent of the variance, which aligns with RF's emphasis on high-impact splits in disassembly data. This value is required to recognise and rank powerful features, enabling data reduction that accelerates models without power loss. Five-fold cross-validation achieved an average accuracy of 98.7 per cent, which justifies RF's robustness to variations in training splits and its applicability to unseen malware variants in dynamic settings.

4.7. Evaluation and Implications

This paper considers the findings and compares the models. It reviews their suitability for responding to research questions (RQ1 and RQ2). Then it explains the work theoretically and practically, its application, restrictions, and morality. It is easy to compute numerical values; researchers assess them using accuracy (total number of accurate guesses) and macro-F1 (average fairness across classes). This demonstrates strengths such as ensembles beating baselines and helps determine whether static features are effective at malware spotting.

4.7.1. Comparative Model Performance

All the models were tested on the same 2,174 test samples of the MMCC dataset. The aim was to test the separability of the nine families using static features (such as the number of opcodes) and whether models are fair to rare families (such as Simda, with only 8 samples). The simplest model, naive Bayes, achieved 76.59% accuracy and a macro-F1 score of 0.60. It casts its light on large families such as Kelihos ver3 (F1=0.98, almost perfectly accurate), since it can rapidly estimate probabilities. It has difficulty dealing with small ones, such as Simda (F1=0.15), because it assumes that features do not form relationships, thereby ignoring opcode connections in hidden code. It resulted in 512 mistakes (24% erroneous), with a large proportion being due to over-guessing common classes. The Support Vector Machine (SVM) achieved 96.04% accuracy and macro-F1 of 0.91. Its lines (hyperplanes) mark the obvious boundaries of major classes, such as Lollipop (F1=1.00), while the RBF curve handles twisted opcode patterns. For rares, Simda decreased to 0.64 F1 due to boundary overlaps; however, overall, errors were reduced to 88 (4%), 20% lower than Naive Bayes. It is efficient, with many features such as PE headers, but requires balanced weights to support small classes. K-Nearest Neighbours (KNN) achieved 98.53% accuracy and 0.97 macro-F1 when the 5 nearest samples were used for voting (Table 1).

Table 1: Model comparison summary

Model	Accuracy	Macro-F1	Errors	Best For
KNN	98.53	0.97	34	Nearby patterns
SVM	96.04	0.91	88	Clear boundaries
XGB	99.54	0.98	10	Step-by-step fixes
NB	76.59	0.60	512	Quick basics
RF	99.26	0.99	17	Group Votes

It entangled local groups into entropy scores, achieving a strong F1 score of 0.88 on Simda. Mistakes were limited to only 34 (1.5%), but were less effective with big data. Random Forest with the group models reached its highest accuracy and macro-F1 of 99.26 and 0.99, respectively, with at least 17 (0.89) votes of the tree. It fastened Simda at 1.00 F1 by averaging random subsets and handled non-straightforward import links. XGBoost reached a maximum of 99.54% accuracy and 0.98 macro-F1, spending errors step-by-step on achieving an ideal Simda (1.00 F1) and the minimum number of errors (10). Imbalance sampling by imbalancing the imbalance in ensembles such as these beats' others by 3-23% in F1. Five-fold cross-checks indicated small changes, i.e., constant on new data. Opcodes such as mov were ranked high (0.30 impact in the SVM), allowing the retention of 80% of the features without loss [26]. In RQ1, overall (99% majors) features are separated 95%+. In the case of RQ2, ensembles are the best at predicting uneven data, boosting rare F1 by 0.15 to 1.00. ROC and precision-recall curves were averaged across ensembles and were much higher than those of NB, at >0.98 (compared to 0.90). All in all, static features are compatible with smart models.

4.7.2. Implications for Malware Detection

The meaning of the results is discussed in this section. It explains the theory behind the models' effectiveness, how they can be applied in the real world, and their shortcomings, with easy solutions. The high scores (maximum of 99) indicate that characteristics such as opcodes can serve as an effective foundation for malware tools. Still, ensembles such as Random Forests and XGBoost can improve performance on data with uneven distributions.

4.7.2.1. Meaning in Theory

The experiments show that static features, such as opcode frequencies, are effective for identifying malware families. For example, the 'mov' opcode alone accounts for approximately 30 per cent of the model's decisions, making it difficult to conceal malware by modifying code, as older signature-based tools cannot operate on those altered files [12]. This is a response to RQ1: with features such as opcodes and PE headers, it isolates classes, and overall performance of 95 percent or more remains after extraneous features are pruned. In the case of RQ2, group models such as Random Forests and XGBoosts are far more effective with uneven data than the basics. Their decision-making or correction F-misses for infrequent classes (such as Simda) are reduced by 85% compared to Naive Bayes or SVM. This coincides with the loss of effectiveness of opcode analysis, which can skew data and undermine your results due to sampling. Still, it also illuminates shared threats in your results through micro-sampling.

4.7.2.2. Real-life Uses

The models' 99% accuracy enables high practicality. They can scan with antivirus software or phone security applications. XGBoost can scan files in less than 1 second, which is 95 times faster than traditional signature matching. With 10 main features

(pruned down to), the system can run on very small devices, including reading emails on a remote server or downloading files on a laptop.

4.7.2.3. Limits and Gaps

Analysis Problems with Static Analysis. Static analysis is incomplete; it fails to detect sleeping malware that is activated only upon execution, such as hidden payloads. Fix: Add dynamic checks (e.g., safe API logs) in future hybrids. New families also require re-tuning of the models, and KNN becomes sluggish with large data. Overall, this work contributes to the development of safer cybersecurity tools that align with guidelines such as NIST [18]. It might reach 99.9 percent in actual apps, with minimal variation.

5. Conclusions and Future Work

5.1. Summary of Findings

Another basic machine learning system was developed in this paper and trained to classify malware using the Microsoft Malware Classification Challenge (MMCC) dataset based on static features. It has addressed the two research questions. The defiant attributes, including the frequency of opcodes and PE header attributes, came in handy for classifying the nine malware families with an accuracy of more than 95 percent in the case of RQ,1, and pruning out the unhelpful attributes produced the same outcome, with greater than 95 percent correct on the major families. This shows that basic details that can be easily extracted can be used to find patterns even in perturbed or disturbed code. The ensemble models, such as XGBoost (99.54% accuracy, 0.98 macro-F1) and the Random Forest (99.26% accuracy, 0.99 macro-F1), performed better on the RQ2 task of class imbalance than the basic model (Naive Bayes). They also reduced the marks of rare families like Simda to 0.15-1.00 by using techniques such as voting and correcting mistakes to less than 1%.

5.2. Future Work

Although the models achieve 99% accuracy with opcode counts, as the models remain in their static state on the MMCC dataset, malware is evolving, and people can practically expand the work to achieve even greater results. The first involves the development of hybrid systems, which involve using the features of a static analysis but with the dynamic aspect, e.g., API calls or network behavior observed in a safe environment like Cuckoo Sandbox, adding 5-10 new features to the existing dataset using pandas, and retrain XGBoost to achieve 99.9% against zero-day threats that are hidden when idle. A further follow-up is to attempt neural networks, where opcode sequences are text input into a model such as BERT on Hugging Face and fine-tuned on the same labeled data to be more sensitive to subtle patterns in rare families such as Simda, which might be significantly better than Random Forest without the additional samples [2]. To bring it closer to the real world, researchers could test our saved models on more real-world devices, such as phones with Python environments like Termux, time our predictions to ensure they take under 1 second to scan with a mobile antivirus and replace slower models like KNN with XGBoost. Lastly, make the system more resilient to ethics-driven failures, such as introducing noise into NumPy to simulate attacks and explaining decisions using SHAP, which builds on the low-bias results to justify rules for safe AI.

Acknowledgment: N/A

Data Availability Statement: Data relevant to this study will be provided by the corresponding author upon reasonable request, where permissible.

Funding Statement: This study was conducted without any external financial support.

Conflicts of Interest Statement: The author reports no conflicts of interest, and all sources used in this original work have been duly acknowledged.

Ethics and Consent Statement: The study followed ethical guidelines, with informed consent obtained from all participants.

References

1. M. Azeem, D. Khan, S. Iftikhar, S. Bawazeer, and M. Alzahrani, "Analyzing and comparing the effectiveness of malware detection: A study of machine learning approaches," *Heliyon*, vol. 10, no. 1, pp. 1–19, 2024.
2. Ö. Aslan and A. A. Yilmaz, "A new malware classification framework based on deep learning algorithms," *IEEE Access*, vol. 9, no. 6, pp. 87936–87951, 2021.

3. F. A. Aboaoja, A. Zainal, F. A. Ghaleb, B. A. S. Al-rimy, T. A. E. Eisa, and A. A. H. Elnour, "Malware detection issues, challenges, and future directions: A survey," *Applied Sciences*, vol. 12, no. 17, p. 8482, 2022.
4. A. A. Almazroi and N. Ayub, "Deep learning hybridization for improved malware detection in smart Internet of Things," *Scientific reports*, vol. 14, no. 4, p. 7838, 2024.
5. A. Redhu, P. Choudhary, K. Srinivasan, and T. K. Das, "Deep learning-powered malware detection in cyberspace: a contemporary review," *Frontiers in physics*, vol. 12, no. 3, pp. 1–29, 2024.
6. M. S. Akhtar and T. Feng, "Evaluation of machine learning algorithms for malware detection," *Sensors*, vol. 23, no. 2, p. 946, 2023.
7. K. Alfarsi, S. Rasheed, and I. Ahmad, "Malware classification using few-shot learning approach," *information*, vol. 15, no. 11, p. 722, 2024.
8. R. Baker del Aguila, C. D. C. Pérez, A. G. Silva-Trujillo, J. C. Cuevas-Tello, and J. Nunez-Varela, "Static malware analysis using low-parameter machine learning models," *Computers*, vol. 13, no. 3, p. 59, 2024.
9. P. Čisar, S. M. Čisar, and A. Pásztor, "Application of Heuristic Scanning in Malware Detection," In NATO Critical Infrastructure Protection: Advanced Technologies for Crisis Prevention and Response, *Springer*, Dordrecht, Netherlands, 2024.
10. A. Ariani, S. Rahmawati, and R. Lumanto, "Study of Lokibot infection against Indonesian network," *OIC-CERT Journal of Cyber Security*, vol. 4, no. 1, pp. 85–96, 2022.
11. G. Karat, J. M. Kannimoola, N. Nair, A. Vazhayil, V. G. Sujadevi, and P. Poornachandran, "CNN-LSTM hybrid model for enhanced malware analysis and detection," *Procedia Computer Science*, vol. 233, no. 4, pp. 492–503, 2024.
12. A. Davarasan, J. Samual, K. Palansundram, and A. Ali, "A comprehensive review of machine learning approaches for android malware detection," *Journal of Cyber Security and Risk Auditing*, vol. 2024, no. 1, pp. 39–60, 2024.
13. D. Dhungana, A. Sapkota, S. Pokharel, S. Devkota, and B. H. Paudel, "Malware Classification using Static Analysis Approaches," *Journal of Artificial Intelligence and Capsule Networks*, vol. 6, no. 4, pp. 494–511, 2025.
14. Every Semicolon Counts, "Microsoft-Malware-Classification," *GitHub*, 2026. [Accessed by 12/04/2026].
15. M. Gopinath and S. C. Sethuraman, "A comprehensive survey on deep learning based malware detection techniques," *Computer science review*, vol. 47, no. 2, p. 100529, 2023.
16. A. Fahim, S. Dey, M. N. Absur, M. K. Siam, M. T. Huque, and J. J. Godhuli, "Optimized approaches to malware detection: A study of machine learning and deep learning techniques," *arXiv preprint arXiv:2504.17930*, 2025. [Accessed by 12/04/2026].
17. GitHub, "static-malware-analysis," *GitHub*, 2023. [Accessed by 15/02/2025].
18. National Institute of Standards and Technology (NIST), "Cybersecurity Framework Version 1.1," *National Institute of Standards and Technology*, 2023. [Accessed by 15/02/2025].
19. R. B. Sulaiman and A. Khraisat, "Metaheuristic-driven feature selection with SVM and KNN for robust DDoS attack detection: A comparative study," *Journal of Cyber Security and Risk Auditing*, vol. 2025, no. 4, pp. 182–203, 2025.
20. K. Shaukat, S. Luo, and V. Varadharajan, "A novel machine learning approach for detecting first-time-appeared malware," *Engineering Applications of Artificial Intelligence*, vol. 131, no. 5, p. 107801, 2024.
21. S. J. Rashid, S. A. Baker, O. I. Alsaif, and A. I. Ahmed, "Detecting Remote Access Trojan (RAT) attacks based on different LAN analysis methods," *Engineering Technology & Applied Science Research*, vol. 14, no. 5, pp. 17294–17301, 2024.
22. R. B. Sulaiman, V. Schetinin, and P. Sant, "Review of machine learning approach on credit card fraud detection," *Human-Centric Intelligent Systems*, vol. 2, no. 5, pp. 55–68, 2022.
23. K. Kamdan, Y. Pratama, R. S. Munzi, A. B. Mustafa, and I. L. Kharisma, "Static malware detection and classification using machine learning: A random forest approach," *Engineering Proceedings*, vol. 107, no. 1, p. 76, 2025.
24. H. J. Zhu, W. Gu, L. M. Wang, Z. C. Xu, and V. S. Sheng, "Android malware detection based on multi-head squeeze-and-excitation residual network," *Expert Systems With Applications*, vol. 212, no. 2, p. 118705, 2023.
25. S. Bhardwaj and H. S. Arri, "A comprehensive study of efficient and accurate malware detection using static and dynamic analysis methods," In *Proceedings of the KILBY 100 7th International Conference on Computing Sciences*, Punjab, India, 2023.
26. D. Thakur, J. Singh, G. Dhiman, M. Shabaz, and T. Gera, "Identifying major research areas and minor research themes of Android malware analysis and detection field using LSA," *complexity*, vol. 2021, no. 1, pp. 1–28, 2021.

Publisher's Note: *The publisher remains impartial concerning jurisdictional claims in published maps and institutional affiliations. Responsibility for the content rests entirely with the authors and does not necessarily reflect the publisher's perspectives.*