School of Computer Science and Electronic Engineering

MSc Cyber Security

Academic Year 2024-2025

Multi Class Malware Classification Using Machine Learning (ML)

A project report submitted by:

Mukesh Kumar

URN: **6893549**

A project supervised by: **Dr. Santanu Dash**

A report submitted in partial fulfilment of the requirement for the degree of Master of Science

University of Surrey
School of Computer Science and Electronic Engineering
Guildford, Surrey GU2 7XH
United Kingdom.
Tel: +44 (0)1483 300800

ABSTRACT

Malware continues to be one of the most pressing challenges in cybersecurity, with modern variants employing obfuscation, polymorphism, and packing techniques to bypass traditional defences. While binary classification approaches can separate malicious from benign software, they provide little insight into malware family attribution, which is critical for digital forensics, threat intelligence, and proactive defence. This dissertation addresses this gap by developing a machine learning–based system for multi-class malware classification using the Microsoft Malware Classification Challenge dataset.

The study extracts both structural features from .bytes files, including entropy values and byte histograms, and semantic features from .asm files, such as opcodes, API calls, register usage, and section metadata. A feature fusion strategy combines these attributes into a unified representation, reduced to 436 features through filtering and pruning techniques. Three classifiers Random Forest, XGBoost, and Support Vector Machines were implemented and evaluated using an 80:20 stratified train–test split. Performance was measured with accuracy, precision, recall, F1-score, and confusion matrices, with macro-averaged scores emphasised to account for class imbalance.

The results show that ensemble methods significantly outperform margin-based approaches. Random Forest and XGBoost achieved near-perfect accuracy and macro-F1 scores of 0.99, while SVM achieved 98% accuracy and macro-F1 of 0.96. Feature importance analysis confirmed that entropy, opcode usage, and API calls were the most influential predictors, validating the effectiveness of feature fusion. However, minority families such as Simda exhibited weaker recall, reflecting the persistent issue of class imbalance.

Keywords: Malware Classification, Feature Fusion, Machine Learning, Ensemble Models, Static Analysis

GitHub Repository for Dissertation Code:
https://github.com/mk6893549/MSc_CyberSecurity/tree/main/Dissertation

ACKNOWLEDGEMENTS

I would like to sincerely thank my supervisor for their invaluable guidance, encouragement, and constructive feedback throughout the course of this dissertation. Their expertise and continuous support have been instrumental in shaping my research and helping me to develop both academically and personally. I am truly grateful for their patience, insightful advice, and commitment, which have made this work possible.

I certify that the work presented in the dissertation is my own unless referenced

Signature: Mukesh Kumar

Date: 08 Sep 2025

TOTAL NUMBER OF WORDS: 14634

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Background

The rapid advancement of digital technologies and global connectivity has introduced significant benefits for individuals, organisations, and societies, yet it has also facilitated the growth of malicious software (malware), which remains one of the most pressing threats to cybersecurity. Malware encompasses a wide range of software designed to disrupt operations, exfiltrate sensitive data, or compromise systems, evolving over the past two decades from simple viruses into highly sophisticated threats such as ransomware, worms, and botnets [9]. These advanced strains frequently evade traditional defences and inflict substantial financial and operational damage worldwide [2].

Conventional detection methods have largely relied on signature-based approaches, where known binary patterns are matched against threat databases. While this remains effective for previously identified malware, such methods fail to detect zero-day threats and obfuscated variants that continuously adapt to bypass recognition [10]. Consequently, researchers and practitioners have increasingly turned to machine learning (ML), leveraging its ability to detect patterns in data and generalise to previously unseen samples [14]. Unlike static signatures, ML classifiers can extract structural or behavioural features, making them more adaptable to evolving threats.

Within this shift, the distinction between binary and multi-class classification has become significant. Binary models that classify software as malicious or benign provide a basic but limited perspective [12], offering little actionable intelligence once malware is detected. Multi-class classification, by contrast, enables attribution of malware to specific families, such as differentiating between “Kelihos” and “Ramnit” [16]. This granularity is vital for incident response, threat intelligence, and forensic investigations, as it reveals insights into propagation techniques, attack goals, and defensive strategies. However, achieving accurate multi-class classification is inherently challenging due to the diversity of malware and the complexity of feature representations [13].

The Microsoft Malware Classification Challenge dataset serves as a benchmark in this field, offering both hexadecimal byte dumps (.bytes) and disassembled assembly code (.asm) across nine malware families [20]. This dual representation allows researchers to explore structural and semantic perspectives of malicious programs [6]. Static analysis of these formats produces diverse features: byte frequency and entropy values from .bytes highlight packing or encryption, while opcode distributions, API calls, and section metadata from .asm files reveal program logic and behaviour. Yet reliance on a single modality risks overlooking crucial signals, motivating the adoption of feature fusion strategies [1][5]. Combining multiple feature sets enhances classification but also creates high-dimensional spaces prone to redundancy, overfitting, and scalability challenges.

The central motivation of this research lies in advancing machine learning–based malware analysis from simple detection to precise family-level classification. By critically assessing feature extraction, fusion strategies, and classification models, this work seeks not only to improve accuracy on benchmark datasets but also to examine the broader challenges of generalisability and robustness in real-world malware detection [8][14].

1.2 Research aim and objectives

The overarching aim of this dissertation is to design, implement, and evaluate a multi-class malware classification system that leverages fused static features from hexadecimal byte dumps (.bytes) and disassembled assembly files (.asm) to improve family-level identification.

To achieve this aim, the dissertation pursues the following objectives:

1. Extract structural features from (.bytes) files and semantic features from .asm files in the Microsoft Malware Classification Challenge dataset.
2. Develop a feature fusion strategy to integrate and reduce high-dimensional attributes into a manageable yet informative representation.
3. Implement a reproducible machine learning pipeline to train and evaluate models consistently.
4. Compare the performance of multiple classifiers Random Forest, XGBoost, and Support Vector Machines on the fused dataset.
5. Evaluate models using comprehensive metrics (accuracy, precision, recall, F1-score, confusion matrices, feature importance).
6. Validate the trained models on selected test samples to demonstrate practical predictive performance and generate interpretable prediction summaries
7. Critically analyse results in relation to prior research, highlighting strengths, weaknesses, unresolved challenges and Future research with AI.

1.3 Research approach

This dissertation adopts a quantitative, experimental approach centred on static malware analysis. Features are extracted from both .bytes and .asm files, engineered into a fused feature set, and reduced to address redundancy and computational overhead. Machine learning pipelines are implemented in Python using open-source libraries, ensuring transparency and reproducibility. Classifiers representing ensemble and margin-based paradigms are trained and compared, with performance evaluated across multiple metrics. This approach enables a fair assessment of the feasibility and effectiveness of fused static features for family-level malware classification.

1.4 Dissertation outline

The dissertation is organised into six chapters.

Chapter 2 provides a literature review of malware detection and classification approaches, highlighting gaps addressed in this work.

Chapter 3 describes the research methodology, including dataset preparation, feature engineering, and model selection.

Chapter 4 presents the system development and implementation, outlining the architecture and training pipelines.

Chapter 5 reports and discusses the results, comparing classifiers, evaluating feature importance, and reflecting on limitations.

Finally, Chapter 6 concludes the dissertation, summarising contributions, acknowledging limitations, and suggesting avenues for future research.

CHAPTER 2: LITERATURE REVIEW

This chapter explores key research on malware detection, outlining common types, analysis techniques, and feature extraction methods. It also examines the strengths and weaknesses of machine learning approaches and shows how this study extends previous work with a fused, multi-class classification strategy.

2.1 Type of Malware and classification challenges

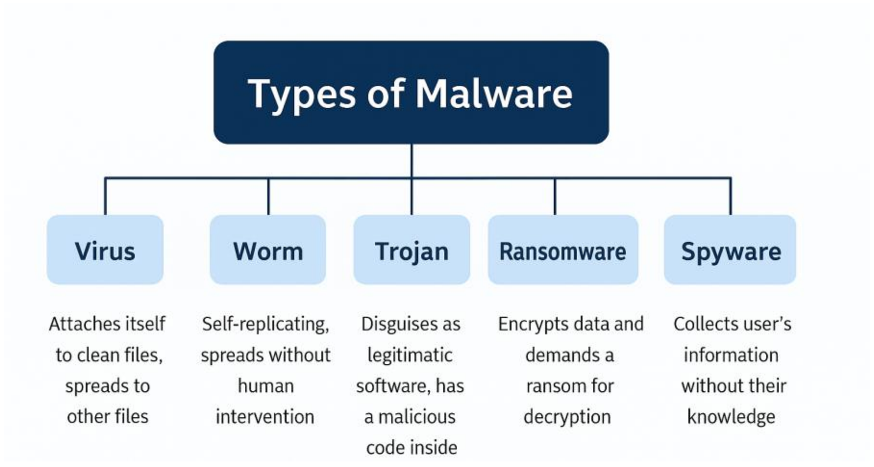

Types of Malware
Virus
Worm
Trojan
Ransomware
Spyware
Attaches itself to clean files, spreads to other files
Self-replicating, spreads without human intervention
Disguises as legitimatic software, has a malicious code inside
Encrypts data and demands a ransom for decryption
Collects user's information without their knowledge

Figure 2.1: Types of Malwares

Malware is a broad term encompassing a variety of malicious software programs designed to compromise confidentiality, integrity, or availability of information systems [8], [14]. While the term often evokes images of viruses and trojans, modern malware exists in diverse forms, each with unique behaviours and attack mechanisms. Common types include viruses, worms, trojans, ransomware, spyware, rootkits, and bots. These classifications are not always mutually exclusive, as many modern threats combine features from multiple categories, making malware classification inherently complex [16].

Viruses replicate by attaching themselves to legitimate files, while worms spread autonomously across networks. Trojans masquerade as benign applications, tricking users into executing them, often resulting in backdoor access. Ransomware encrypts data and demands payment, while spyware operates covertly, harvesting sensitive user information. Rootkits offer persistent, stealthy access to attackers by altering system-level operations. Botnets link infected systems to a command-and-control infrastructure, often for launching distributed attacks [14].

The heterogeneity among these malware types introduces several challenges for automated classification. First is the issue of intra-class variation, where samples within the same category may exhibit widely differing code structures due to obfuscation, packing, or polymorphism [9], [16]. Second, inter-class similarity further complicates classification, as different malware types can share overlapping features for example, both ransomware and spyware may invoke file system and registry operations, making them difficult to distinguish using static analysis alone [9], [14].

Another major concern is labeling inconsistency. Malware is typically labeled by antivirus vendors, whose naming conventions can vary significantly. This leads to ambiguities in ground truth, especially for supervised machine learning tasks [8].

While datasets like the Microsoft Malware Classification Challenge attempt to standardize labels, real-world malware repositories often lack this consistency, hampering the development of generalizable models [20].

Class imbalance is also a persistent issue. Certain malware families dominate public datasets, while others are severely underrepresented. This imbalance can cause machine learning models to become biased toward majority classes, achieving high accuracy but poor performance on minority or emerging threats. Traditional performance metrics such as accuracy may therefore be misleading, necessitating the use of more nuanced metrics like macro F1-score or per-class recall [8], [14].

The evolution of malware further exacerbates these challenges. Malware authors constantly innovate to bypass detection systems by introducing polymorphic and metamorphic techniques, which dynamically alter the code structure without changing the payload's intent [9], [16]. This dynamic evolution limits the effectiveness of static features and increases the need for adaptive or hybrid approaches that can generalize across unseen variants [9], [11].

This dissertation navigates these classification challenges by adopting a multi-class approach trained on a diverse set of static features. While it relies on standardized malware families from the Microsoft dataset, it critically evaluates classifier performance across all classes, with a focus on minimizing bias and enhancing generalization [20].

2.2 Static vs Dynamic analysis techniques

Static and dynamic analysis are two foundational techniques in malware analysis, each offering distinct benefits and limitations. Static analysis involves inspecting the binary or source code of a program without executing it. This method is efficient and scalable, as it allows the extraction of features such as byte sequences, opcodes, and API call patterns [12], [8]. It is particularly suitable for early detection systems and large-scale classification tasks. However, its effectiveness is reduced when faced with obfuscated or packed malware, which can conceal malicious behaviour from code-level inspection [16].

In contrast, dynamic analysis observes a program's behaviour at runtime, capturing its interaction with the operating system, file system, network, and memory [9], [15]. This approach excels at revealing hidden behaviours and is more resilient to code obfuscation. Nonetheless, it is computationally expensive and may miss behaviours that are triggered only under specific conditions [9].

In practice, both methods are often used in tandem to provide comprehensive insights. For this research, static analysis is chosen due to its compatibility with machine learning pipelines and its feasibility for large dataset processing, especially when dealing with feature-rich static artifacts from .asm and .bytes files [20]

2.3 Overview of previous feature extraction approaches

[https://arxiv.org/pdf/1710.08189

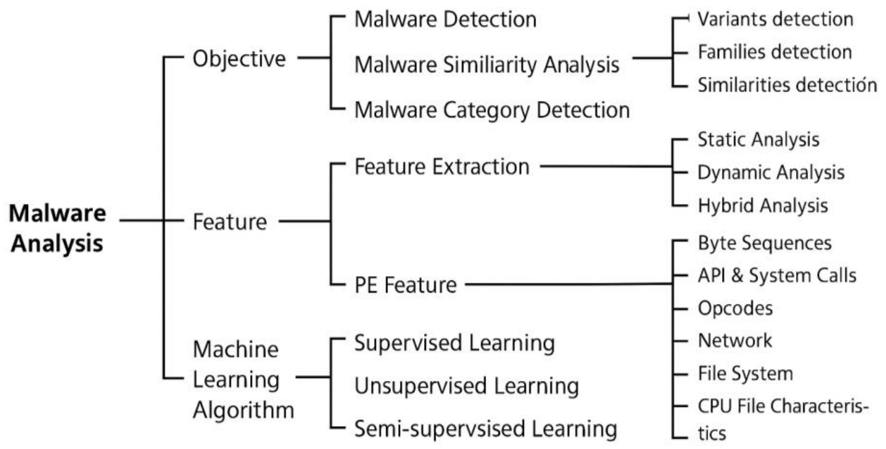


Figure 2.3: Feature extraction approaches

Early approaches to malware classification heavily relied on signature-based detection, which, while effective for known threats, proved inadequate against novel or obfuscated variants [16]. As the malware landscape evolved, the need for more resilient and scalable classification systems led researchers to explore machine learning models trained on features extracted from malware binaries. Feature extraction became a pivotal stage, transforming raw binary data into structured formats interpretable by algorithms [8], [14].

A foundational contribution in this space was made by Nataraj et al. (2011), who introduced the concept of converting malware binaries into grayscale images and extracting texture-based features such as histograms and Gabor filters [1]. This method inspired subsequent work on image-based feature extraction, where researchers like Xu et al. (2016) used Local Binary Patterns (LBP) and Haralick descriptors to model visual patterns within malware families [4][28]. These image-derived features demonstrated strong discriminative power but also required significant preprocessing and were sensitive to file padding and scaling inconsistencies.
Parallel to image-based techniques, a substantial body of research focused on opcode and byte-level features. Saxe and Berlin (2015) developed a deep learning model using n-gram byte sequences extracted from hex dumps of malware files [3].

This approach leveraged frequency-based patterns in byte-level data, capturing both structural and statistical characteristics of malware. While effective, byte n-grams introduced high dimensionality, which necessitated feature selection or dimensionality reduction techniques such as PCA or mutual information.
Opcode frequency, derived from disassembled .asm files, has been another prevalent method. By mapping instruction usage across malware samples, researchers were able to model operational behaviours at a low level. Additionally, API call frequency analysis became increasingly relevant, given its role in uncovering functional intentions of malware such as file manipulation, registry editing, and network communication. Tools like IDA Pro and Radare2 facilitated disassembly, while custom parsers helped extract function names and control-flow patterns.

Hybrid feature sets gained popularity in more recent work, combining entropy-based measures, section metadata (e.g., .text, .data, .rsrc sections), and string-based features. These allowed models to learn both static structure and behavioural hints encoded in the binaries. Raff et al. (2018), for instance, proposed using raw bytes alongside entropy signals in a convolutional neural network framework, showcasing the importance of layered feature integration [5][22].

Despite the richness of these feature spaces, challenges persist. Many features are correlated or redundant, which can degrade classifier performance. Furthermore, static features are inherently vulnerable to evasion via packing or obfuscation. Researchers have thus explored dynamic and hybrid feature extraction strategies, although these come at the cost of scalability and resource efficiency [11], [9], [14], [8].

2.4 Review of Machine Learning Algorithms

Machine learning has become an indispensable tool in modern malware detection due to its ability to generalize from patterns in high-dimensional data [8], [14]. Traditional signature-based antivirus systems are insufficient in identifying newly evolved or obfuscated malware, prompting the shift toward automated classification systems. Among the earliest approaches were supervised learning models trained on static features such as byte n-grams, opcode frequencies, and API call patterns. These methods demonstrated significant potential in classifying malware into families and identifying novel variants based on similarity to known patterns.

Support Vector Machines (SVM) have historically been a popular choice due to their robustness in high-dimensional spaces and effectiveness with limited training data. Research by Kolter and Maloof (2006) applied SVMs using n-gram features and achieved high detection rates [2][31]. However, SVMs suffer from long training times and scalability issues when applied to large datasets such as the Microsoft Malware Classification Challenge.

Decision Trees and Random Forests have also been extensively used, particularly for their interpretability and speed. Random Forests, being ensemble-based, mitigate the variance of individual decision trees and are capable of handling noisy, heterogeneous feature sets. They have shown strong performance in malware classification tasks, especially when combined with engineered features from disassembled files [1], [14][30]. However, Random Forests may underperform on sparse feature matrices like those derived from byte-level histograms.

More recent work has favoured boosting algorithms such as XGBoost and LightGBM. These models iteratively refine weak learners to optimize prediction performance. XGBoost, in particular, has demonstrated state-of-the-art accuracy in many malware detection competitions, including the Microsoft Malware Challenge [20], and is frequently adopted in static malware analysis pipelines [14][29]. Its built-in handling of missing values, regularization, and feature importance scoring makes it highly suitable for cybersecurity datasets. It also scales well with large feature sets, as found in static malware analysis.

On the deep learning front, models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have been employed to process raw binaries or sequences of instructions. CNNs have been used to detect visual patterns in image-transformed malware samples [3], while RNNs model opcode or API sequences as time-series data [18]. Although these models achieve high accuracy, they require extensive computational resources and large labeled datasets, which can limit their practical deployment in resource-constrained environments [8], [14]. Unsupervised learning techniques, including clustering algorithms like K-means and DBSCAN, have been applied for malware family discovery and anomaly detection. These methods are valuable when dealing with unlabeled or evolving threats but often lack the precision needed for fine-grained classification [13], [19].

Hybrid approaches have emerged that combine static and dynamic features or use ensemble models to integrate predictions from multiple algorithms. These strategies improve resilience against evasion tactics but increase the complexity of the detection system [9], [11][24]. Feature selection and dimensionality reduction techniques such as Principal Component Analysis (PCA) and mutual information have been incorporated to reduce overfitting and improve generalization [8], [14], [17].

2.5 Limitations in Existing Approaches and Datasets

While significant progress has been made in malware classification using static and dynamic analysis combined with machine learning, several limitations persist in both the approaches and the datasets used. One of the most critical challenges is the lack of diversity and realism in publicly available datasets. Many widely used datasets, such as the Microsoft Malware Classification Challenge dataset, are sanitized and curated for academic use, often omitting benign samples and lacking up-to-date malware families [10], [8], [14], [20]. This limits the applicability of trained models in real-world scenarios, where novel or zero-day malware variants frequently emerge.

Moreover, class imbalance is a recurring issue in many malware datasets. In multi-class classification settings, some malware families are overrepresented while others are underrepresented, leading to biased models that generalize poorly across minority classes [8], [14][23]. This skew often results in high overall accuracy but low performance in less common malware categories an issue that is rarely addressed thoroughly in evaluation metrics.

From a feature engineering perspective, static features such as byte n-grams, opcode frequencies, and API calls are susceptible to evasion through obfuscation, packing, and encryption [16], [9]. Polymorphic and metamorphic malware, for instance, can modify their code signatures without changing their behaviour, thereby evading detection systems that rely heavily on these features. While dynamic analysis offers deeper behavioural insight, it introduces its own limitations namely, high computational cost, environmental dependency, and the risk that malware may detect sandboxed environments and suppress execution [9], [15].

Another challenge lies in feature redundancy and high dimensionality. Especially in static analysis, features derived from byte-level data can result in tens of thousands of dimensions. This not only increases computational complexity but also introduces noise, potentially degrading classifier performance [8], [14]. Although dimensionality reduction techniques like PCA or mutual information gain have been applied, they can obscure the interpretability of the model and fail to preserve all relevant information.

Further, many existing approaches lack explainability, a critical requirement in cybersecurity. Deep learning models, in particular, act as black boxes, offering limited insight into why a particular file was classified as malicious or benign [14]. This undermines trust in automated systems and limits their adoption in enterprise settings where analysts require interpretable evidence to validate decisions.

Lastly, labeling inconsistencies are a problem in many datasets. Malware naming conventions vary significantly between antivirus vendors, and even the same sample may be labeled differently across platforms [8]. This ambiguity complicates supervised learning, particularly for family classification tasks. While initiatives like the Microsoft dataset standardize labels, they represent a small subset of real-world malware and are often outdated [20].

In light of these limitations, this dissertation adopts a static feature-based approach but complements it with robust preprocessing, feature fusion, and a scalable classification pipeline. While acknowledging the constraints of static analysis and dataset scope, the study aims to explore how far existing datasets can be leveraged to build generalizable and efficient multi-class malware classifiers [17], [20].

2.6 How This Project Builds on or Differs from Previous Work

This dissertation builds upon a rich body of prior research in static malware classification, yet introduces key differences in both scope and implementation. While much of the existing literature has focused on binary classification or detection of specific malware types, this project adopts a multi-class classification framework. This shift allows for finer-grained categorization of malware into distinct families, aligning more closely with real-world incident response workflows where understanding malware lineage is crucial [8], [14].

Furthermore, while several prior studies rely on singular feature domains such as byte-level n-grams, opcode frequency, or image-based texture descriptors this work integrates a multi-domain feature fusion strategy. Drawing on both .bytes and .asm files, the project extracts diverse static features including opcode counts, API call frequency, entropy measures, string length distribution, and section metadata. This broader feature base enhances the classifier's ability to distinguish between malware variants that may share superficial structural traits but differ in behaviour or composition [1], [3], [17].

Another key distinction lies in the choice of classifier. Many earlier approaches used traditional algorithms such as SVM, Random Forest, or k-NN. While effective, these models may struggle with high-dimensional, sparse feature sets typical of static malware data. In contrast, this project employs XGBoost, a gradient boosting framework known for its robustness, scalability, and superior handling of heterogeneous features. Its built-in support for regularization and feature importance analysis also enables a more interpretable and optimized model pipeline [14], [20][26].

The project also addresses limitations in past dataset usage. Although based on the Microsoft Malware Classification Challenge dataset, this research goes beyond standard practices by engineering custom static features not explicitly explored in the

original competition. While prior submissions often relied on either raw byte images or disassembly tokens, this dissertation emphasizes structured feature engineering to enhance model interpretability and computational efficiency [1], [17].
Lastly, this work places greater emphasis on critical evaluation. While many studies report high accuracy without deeper insight, this dissertation includes confusion matrix analysis, per-class metrics, and feature importance interpretation to understand both strengths and failure modes of the classifier. By examining misclassifications and performance disparities between classes, the project offers a more nuanced reflection on model generalizability and real-world applicability [8], [14].

CHAPTER 3: RESEARCH APPROACH

This chapter explains the research design and methods used to build and test a malware classification system. It covers the dataset, feature extraction from both byte and assembly files, data preprocessing, model selection, training, and evaluation. The aim is to show how combining different static features and testing multiple algorithms can improve classification accuracy.

3.1 Research Design

The research design for this dissertation takes an experimental approach grounded in applied machine learning and cybersecurity. The main goal is to examine how static feature extraction and fusion can improve the multi-class classification of malware families using the Microsoft Malware Classification Challenge dataset [20]. To achieve this, the study emphasises reproducibility and careful handling of data so that results can be both trusted in an academic setting and considered useful in real-world applications.

This work follows a quantitative research design because the analysis is based on numerical features extracted from malware binaries and assembly code. The performance of models is measured using clear, objective metrics such as accuracy, precision, recall, and F1-score. This choice is appropriate because the problem is data-driven and large-scale, making it better suited to statistical modelling than to qualitative approaches like expert interviews or case studies. The central hypothesis is that combining features from different sources, such as byte-level patterns and assembly code, will lead to more accurate classification than relying on one type of feature alone.

The methodology is experimental in nature, meaning it systematically varies key factors such as which features are used, how the data is pre-processed, and which algorithms are applied to see how they influence classification performance. For example, models trained only on byte-level features are compared with those using assembly features and those using both. This allows a fair test of whether combining features really provides an advantage. Preprocessing steps, including feature selection and scaling, are carefully applied to avoid problems such as redundancy or information leakage between training and test sets. Although the project uses a hold-out validation strategy to separate training and test data, it is recognised that cross-validation could have offered a more robust assessment, and this is noted as a limitation.

Three machine learning algorithms Random Forest, XGBoost, and Support Vector Machines were chosen because they represent different learning strategies and are widely used in malware research. Testing them under the same conditions ensures that comparisons are fair and reliable. However, deep learning approaches were not included in the main experiments, which limits the breadth of the comparison.

Finally, because malware datasets are often imbalanced, overall accuracy is not enough to judge performance. This study therefore uses per-class and macro-averaged metrics to give a fuller picture of how well each model performs across all malware families, including those with fewer samples. Even so, minority families remain difficult to classify, which highlights an ongoing challenge for applying these methods in practice.

3.2 Microsoft Malware Classification Challenge Dataset

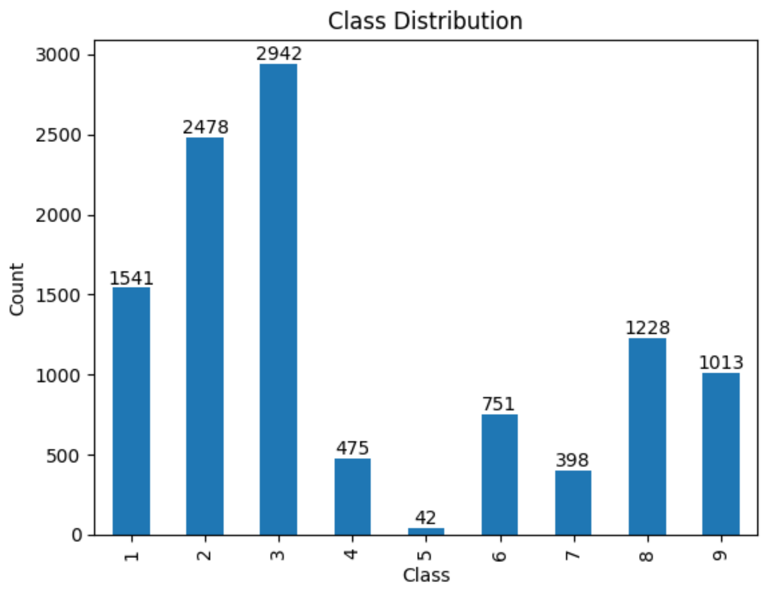


Figure 3.2: Class Distribution of malware Samples

The dataset used in this study is the Microsoft Malware Classification Challenge (BIG 2015) dataset [20], released on Kaggle by Microsoft and Symantec to advance research in malware family classification. It is widely regarded as a benchmark dataset because it provides a large and diverse collection of real-world malware samples that are publicly available, enabling reproducibility and comparison across studies [AE].

Table 3.2A: Dataset File structure

File Type	Extension	Description
Disassembled code	.asm	Generated by IDA disassembler; contains assembly instructions, API calls, registers, and section metadata.
Hexadecimal dump	.bytes	Raw bytecode of executables in hexadecimal format, representing binary structure of malware files.
Labels (train set)	CSV file	Contains malware family labels (class IDs 1–9) corresponding to each training sample.
Test set	.asm + .bytes	Same structure as training data but without labels, used for competition submissions.

The dataset contains 10,868 samples grouped into nine families, representing categories such as backdoors, worms, and trojans. A key challenge is its imbalance: families like Ramnit and Lollipop dominate the collection, while others, such as Simda, are underrepresented. This reflects real-world distributions but complicates multi-class classification, as models often favour majority classes.

Table 3.2B: Distribution of malware Family

Class ID	Family Name	Type	Description
1	Ramnit	Worm	Spreads by infecting executable files and HTML documents, enabling mass propagation.
2	Lollipop	Trojan	Distributed mainly through removable drives, often used to drop additional malware.
3	Kelihos_ver3	Botnet/Trojan	Variant of the Kelihos botnet, used for spam, data theft, and distributed attacks.
4	Vundo	Trojan	Associated with adware and browser hijacking, leading to unwanted pop-ups and redirects.
5	Simda	Backdoor Trojan	Provides remote control of infected systems, enabling attackers to execute arbitrary commands.
6	Tracur	Trojan Downloader	Installs additional malware by downloading payloads from malicious domains.
7	Kelihos_ver1	Botnet/Trojan	Early Kelihos variant, engaged in spam campaigns and distributed denial-of-service (DDoS) attacks.
8	Obfuscator.ACY	Packer/Obfuscator	A generic obfuscation tool used to disguise the presence and behaviour of other malware.
9	Gatak	Trojan	Distributed via malicious key generators (keygens), used to steal data and spread further malware.

Each sample is provided in two formats: a hexadecimal byte dump (.bytes) representing raw binary content and a disassembled assembly file (.asm) produced by the IDA disassembler. The .bytes files reveal structural patterns and entropy variations useful for detecting obfuscation, while .asm files expose program logic through opcodes, registers, API calls, and section metadata. Using both formats allows for a richer representation and underpins the feature fusion strategy of this research.

3.3 Feature Extraction

Feature extraction is a central component of the malware classification pipeline, as the effectiveness of machine learning models depends heavily on the quality, diversity, and discriminative power of the features used. Malware binaries can be represented in multiple ways, each capturing different aspects of their structure and behaviour. In this study, features were extracted from two complementary static representations: hexadecimal byte dumps (.bytes files) and disassembled assembly code (.asm files). The decision to employ both modalities reflects the recognition that malware exhibits both structural patterns (e.g., byte-level distributions, entropy) and semantic behaviours (e.g., opcode usage, API calls, section metadata).

Table 3.3 A: Features from Hex Dump (.bytes)Files

Feature Category	Description	Strengths	Limitations
Byte Frequency Features	256-dimensional histogram capturing frequency of each byte (0x00–0xFF). Useful for identifying instruction encodings, padding, and packing patterns.	Provides a statistical fingerprint; scalable and computationally efficient.	Lacks semantic depth; similar distributions may occur across different code bases.
Entropy-Based Features	Measures randomness of byte sequences using	Effective at detecting compression,	High entropy also occurs in benign packed

	Shannon entropy across sliding windows. Includes metrics such as max, average, and quantile differences.	encryption, and polymorphic malware; highlights obfuscation.	software; risk of false positives if overemphasised.
Image-Based Transformations	Converts raw byte values to grayscale images to capture texture and global structure. Extracts descriptors such as variance and edge counts.	Captures holistic structural patterns of malware families; complements statistical features.	Less interpretable than entropy/frequency; risk of redundancy; dimensionality challenges.
N-Gram Byte Sequences	Frequencies of fixed-length subsequences (bi-grams, tri-grams). Captures coding or instruction patterns at byte level.	Identifies family-specific signatures; lower-order n-grams are efficient and less sparse.	Higher-order n-grams provide richer semantics but are computationally expensive and sparse.
String Length Distribution	Distribution of embedded string lengths (file paths, registry keys, URLs). Indicates functional aspects such as network communication.	Provides behavioural clues; helps distinguish families based on embedded resources.	Vulnerable to obfuscation; encrypted or fragmented strings reduce reliability.
File-Level Metadata	Includes file size, line counts, zero-byte ratios, and filler patterns (e.g., 0xFF).	Cheap to compute; captures family-specific high-level characteristics.	Coarse-grained; lacks detail; limited standalone discriminative power.

Table 3.3 B: Features from Disassembled (.asm) Files

Feature Category	Description	Strengths	Limitations
Opcode Frequencies	Counts of assembly instructions (e.g., mov, push, call, jmp, xor) across the sample.	Highlights distinctive execution patterns; useful for detecting encryption, obfuscation, or self-modification.	Reduces instruction sequences to flat counts, losing control-flow and contextual information; families with similar opcode usage may appear indistinguishable.
API Call References	Presence and frequency of Windows API functions (e.g., CreateProcessA, RegOpenKeyExW, URLDownloadToFileW).	Strong behavioural indicators; links directly to malware functionality such as persistence or network communication.	Easily obfuscated; malware can dynamically resolve function addresses to evade static analysis.
Register Usage	Frequency of general-purpose registers (eax, ebx, ecx, edx) used in code.	Captures family-specific or compiler-dependent patterns; useful for identifying obfuscation strategies.	May reflect compiler artefacts rather than malicious intent; not always reliable for behaviour inference.
Section Metadata	Counts, sizes, and ratios of standard sections (.text, .data, .rdata, .rsrc).	Identifies anomalies such as oversized or manipulated sections used to hide payloads.	Legitimate software may also contain unusual sections; features can be noisy.
String Features	String length distributions and presence of suspicious categories (.exe, .dll, HTTP, registry keys).	Reveals network indicators, persistence mechanisms, or embedded resources; provides interpretable behavioural clues.	Obfuscation (encryption, fragmentation) reduces reliability; strings can be concealed.
Symbol Features	Counts of labels, jump targets, and imported/exported function names.	Indicates control-flow complexity and use of indirect addressing; useful for identifying polymorphic malware.	Compiler artefacts or packing tools can inflate symbol counts; may introduce noise.

Line Counts	Total number of lines in disassembled code.	Simple measure of code size and complexity; distinguishes between large, modular malware and lightweight droppers.	Coarse-grained; lacks behavioural detail; must be used in combination with finer-grained features.
Data-Defining Patterns	Frequency of pseudo-instructions (db, dw, dd, dq) defining raw embedded data.	Highlights packed/encrypted payloads, embedded configuration blocks, or cryptographic material.	Benign software may also contain static data; advanced malware can obfuscate patterns to avoid detection.

3.4 Feature Fusion Strategy

This dissertation employs a feature fusion strategy that integrates attributes from hexadecimal byte dumps and disassembled assembly files. The .bytes features capture structural and statistical properties, such as entropy and byte distributions, while the .asm features reflect semantic aspects, including opcode usage, API calls, and register patterns. By combining these modalities into a unified representation, the models gain a more holistic view of malware behaviour.

The rationale for fusion stems from the weaknesses of using single feature sources in isolation. Byte-level patterns may indicate obfuscation but lack context, whereas semantic features can reveal intent but are sensitive to compiler artefacts and obfuscation. Integrating both allows the strengths of one to offset the weaknesses of the other. Preprocessing steps, including the removal of low-variance features and standardisation, ensured the final feature set was both representative and manageable.

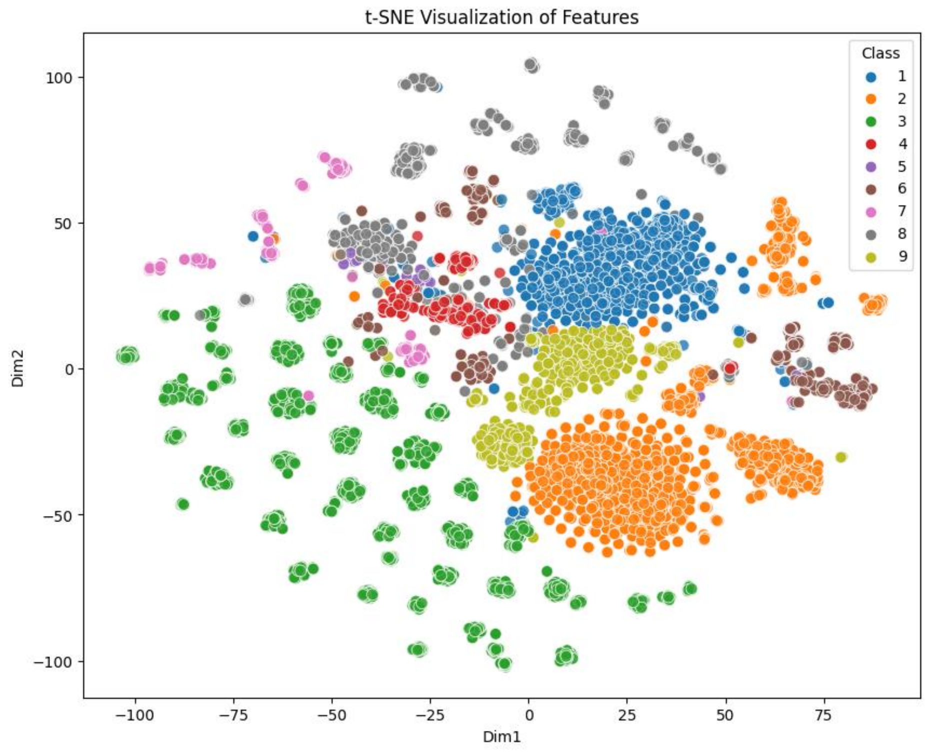


Figure 3.4: Visualization of features

The t-SNE visualization shows how fused features from .bytes and .asm files separate malware samples into clusters that largely align with their families. Distinct groupings, such as those for Kelihos_ver3 and Lollipop, demonstrate strong intra-class similarity, while overlaps between families like Kelihos_ver1 and Kelihos_ver3 reveal structural and behavioural commonalities that complicate classification. Although t-SNE does not directly affect model performance, it provides an intuitive validation of feature quality. The clustering supports the effectiveness of the fusion strategy, but the overlaps highlight the limits of static features in fully distinguishing malware families.

3.5 Data Preprocessing

Data preprocessing formed a vital bridge between raw feature extraction and model training in this study. The initial feature space exceeded 1,800 attributes drawn from both .bytes and .asm files, including byte histograms, entropy values, opcode frequencies, API calls, and section metadata. Many features were either redundant, sparsely populated, or exhibited little variance, making them unsuitable for learning. Through cleaning and dimensionality reduction, the feature set was reduced to 436 variables that balanced representational richness with computational feasibility.

To prepare the dataset for algorithms, scaling was applied selectively. Support Vector Machines required standardization to zero mean and unit variance to ensure fair weighting of features, while Random Forests and XGBoost, being tree-based, remained unaffected by scale. Class labels were encoded into integers using scikit-learn's Label Encoder to allow seamless evaluation of per-class metrics.

The dataset was split 80:20 into training and test sets, stratified by family labels to preserve class distribution and prevent minority classes from being excluded. Missing values, which were rare, were replaced with zero to indicate the absence of features without discarding samples. Finally, class imbalance was documented, with evaluation relying on macro-averaged precision, recall, and F1-score to ensure fair assessment across minority and majority families.

3.6 Models

The choice of machine learning models is a critical aspect of this dissertation, as it directly influences the ability to classify malware into distinct families with high accuracy and robustness. In designing the model selection and training process, the study was guided by three considerations: the suitability of the algorithms for high-dimensional static features, their ability to handle class imbalance, and their proven effectiveness in prior malware classification research. Based on these criteria, three algorithms were selected for comparative evaluation.

3.6.1 Random Forest

Random Forest is an ensemble learning technique that constructs a collection of decision trees, each built on random subsets of training data and features. By combining the outputs of these trees through majority voting, the model reduces variance and improves stability compared to individual decision trees. This design enables Random Forest to capture complex, non-linear interactions within data while remaining relatively resistant to overfitting.

One of its key advantages lies in its ability to handle high-dimensional feature spaces efficiently, making it suitable for problems involving thousands of attributes. In addition, Random Forest provides intrinsic measures of feature importance, offering valuable insights into which variables contribute most to classification decisions. Despite its strengths, the model is not without limitations. Its predictions can be less effective when faced with highly imbalanced datasets, and while it is more interpretable than some black-box methods, its ensemble nature still makes fine-grained decision analysis challenging.

3.6.2 Extreme Gradient Boosting (XGBoost)

XGBoost is an advanced implementation of gradient boosting that builds decision trees sequentially, with each new tree designed to correct the errors of its predecessors. By combining many weak learners into a strong ensemble, it is able to model highly complex, non-linear relationships in data. The algorithm incorporates regularisation mechanisms, such as L1 and L2 penalties, which help prevent overfitting, making it more robust than traditional boosting approaches.

A notable strength of XGBoost is its scalability and efficiency in handling large, high-dimensional datasets, aided by parallel processing and optimised memory use. It also offers techniques for addressing class imbalance, such as weighting schemes, which make it effective in domains where certain categories are underrepresented. However, XGBoost is highly sensitive to hyperparameter settings, and inappropriate choices can lead to overfitting or poor generalisation. Furthermore, while feature importance scores are available, the complexity of the boosted ensemble can make interpretation less straightforward than in simpler models.

3.6.3 Support Vector Machines (SVM)

Support Vector Machines (SVM) are supervised learning algorithms that classify data by identifying an optimal hyperplane which maximises the margin between different classes. They are especially effective in high-dimensional settings, as performance is determined primarily by the support vectors rather than the full set of features. This property allows SVMs to manage complex datasets with large numbers of attributes. By applying the kernel trick, SVMs extend their capabilities beyond linear boundaries, enabling the modelling of more intricate, non-linear relationships between classes.

Despite their strengths, SVMs also present limitations. They can be computationally expensive, particularly in multi-class problems where multiple binary classifiers must be constructed. Their sensitivity to feature scaling requires careful preprocessing to achieve consistent performance. Moreover, while they often deliver strong accuracy, SVMs are less robust to class imbalance and lack intrinsic interpretability, as they do not provide feature importance measures. These factors make SVM a valuable but more resource-intensive option compared to ensemble-based methods.

3.7 Training Process

To ensure fairness in model comparison, all three classifiers were trained on the same 436-feature fused dataset, with identical train-test splits (80:20 stratified). Training was conducted in Python using scikit-learn for Random Forest and SVM, and the XGBoost library for gradient boosting. The models were evaluated on the stratified test set to measure their generalisation performance. Evaluation metrics included accuracy, precision, recall, F1-score, and confusion matrices, with macro-averaged scores emphasised to account for class imbalance.

The training process deliberately avoided advanced hyperparameter optimisation techniques such as exhaustive grid search, in order to focus on the comparative performance of baseline implementations.

3.8 Evaluation Metrics

$TP = True\ Positives;\ TN = True\ Negatives;\ FP = False\ Positives;\ FN = False\ Negatives$

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

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

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall}$$

The evaluation of machine learning models requires metrics that capture true performance across all classes, particularly in multi-class problems with imbalanced distributions such as malware classification. Relying solely on accuracy can be misleading, as models may achieve high scores by favouring majority classes while ignoring minority families. For this reason, this study reports accuracy but places greater emphasis on precision, recall, F1-scores, and confusion matrices, along with macro- and micro-averaged variants of these measures.

Accuracy offers an overall measure of correctness but fails to account for imbalance. Precision measures the reliability of positive predictions, while recall reflects the ability to identify all relevant samples. The F1-score combines these into a balanced indicator. In multi-class settings, macro-averaged F1 gives equal weight to all families, ensuring that minority classes such as Simda are fairly represented. Micro-averaged F1, dominated by majority classes, provides complementary insight into overall performance.

Confusion matrices provide a detailed breakdown of classification errors, revealing systematic misclassifications between families. Reporting support counts alongside precision, recall, and F1 further contextualises results, clarifying whether scores are based on many or very few samples.

ROC-AUC and precision-recall curves were considered but not used, as they are less interpretable in a nine-class setting. Their inclusion is noted as a potential avenue for future work, especially if binary malware-versus-benign classification is explored.

The chosen strategy, centred on macro-F1 and confusion matrices, ensures that performance on minority classes is not overshadowed by majority classes. However, results are based on a single stratified hold-out split, which introduces some variance, particularly in underrepresented families. Cross-validation would strengthen reliability but at greater computational cost. Overall, the evaluation framework balances fairness, interpretability, and feasibility, providing a transparent basis for assessing multi-class malware classification models.

3.9 Model Validation on Test Samples

Beyond conventional evaluation metrics, this study also validated the trained models on a small subset of unseen test samples. This step aimed to demonstrate how the classifiers perform in practice by directly comparing predicted labels with their actual classes. While aggregate measures such as accuracy and F1-score provide useful benchmarks, they can obscure individual misclassifications and their impact. By testing on selected samples, the research highlights the models' consistency and interpretability, offering a clearer view of predictive reliability. However, as the validation set was limited in size, these results should be viewed as illustrative rather than definitive.

CHAPTER 4: SYSTEM DESIGN AND DEVELOPMENT

This chapter details how the classification system was built and tested, integrating features from .bytes and .asm files into machine learning models. The approach achieved strong results, especially with ensemble methods, though limitations remain in scalability and broader validation.

4.1 System Architecture

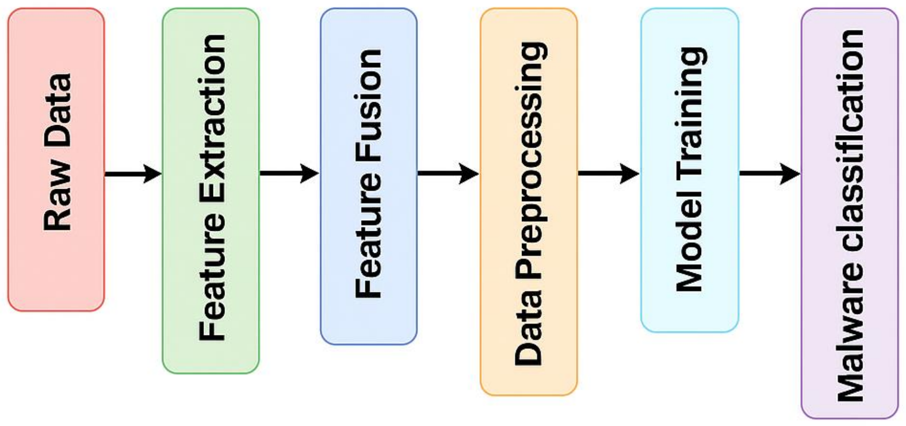


Figure 4.1: System Architecture Diagram

The proposed system architecture is designed to provide a structured and modular pipeline for addressing the challenge of multi-class malware classification. At its core, the architecture transforms raw data from the Microsoft Malware Classification Challenge dataset into meaningful representations that can be effectively processed by machine learning algorithms. The workflow begins with the input layer, which consists of raw .bytes files representing the hexadecimal dump of executable binaries and .asm files containing the disassembled code. These two complementary sources of static information provide both structural and semantic insights into malware samples, enabling a richer foundation for classification compared to relying on a single data source.

The first functional module of the architecture is feature extraction, which translates low-level raw data into higher-level attributes. From .bytes files, statistical and structural indicators such as entropy values, byte frequencies, and image-based patterns are extracted, capturing hidden regularities and randomness that often distinguish malware families. From .asm files, semantic features such as opcode frequencies, API calls, register usage, and section metadata are obtained, which offer behavioural insights into how malware operates at the instruction level. By separating feature extraction into two parallel processes, the system ensures that both structural and operational aspects of malware are adequately represented.

The next stage of the architecture is feature fusion and preprocessing, which integrates extracted attributes from both sources into a single unified feature vector. This step not only increases the representational power of the dataset but also addresses the limitations of individual feature sets. However, combining heterogeneous features introduces issues such as high dimensionality, redundancy, and imbalance. To address these challenges, preprocessing techniques such as scaling, normalisation, and dimensionality reduction (e.g. feature selection) are applied. This stage ensures that the models are trained on informative and nonredundant features, reducing overfitting while improving computational efficiency.

Once the features are prepared, the classification module serves as the decision-making core of the architecture. Here, supervised machine learning algorithms such as Random Forests, Support Vector Machines, and XGBoost are employed to map feature vectors to malware family labels. The choice of multiple classifiers allows for comparative analysis, enabling the identification of strengths and weaknesses of different learning approaches. For instance, tree-based models may excel in interpretability and handling heterogeneous features, whereas boosting-based methods can provide superior predictive accuracy. The modular design of this stage also allows for the future integration of deep learning models, should the research extend to larger datasets or require more complex representation learning.

Finally, the evaluation and reporting module assesses the performance of the classification models using standard metrics such as accuracy, precision, recall, F1-score, and confusion matrices. This stage provides quantitative evidence of the effectiveness of the system while highlighting misclassification patterns that may reveal weaknesses in feature representation or model generalisation. By critically analysing the results in light of existing research, the architecture not only provides a mechanism for malware classification but also acts as a platform for exploring the broader research question of how feature fusion impacts classification performance.

Overall, the system architecture demonstrates a deliberate balance between modularity and integration. Each stage operates independently but contributes to a cohesive pipeline that transforms raw data into actionable knowledge. Its design reflects the need for scalability, reproducibility, and adaptability in malware analysis research, ensuring that the system can evolve in response to emerging malware trends and the increasing sophistication of obfuscation techniques.

4.2 Feature Selection & Reduction

Feature selection and reduction form a critical stage in the malware classification pipeline, transforming a raw feature space of more than 1,800 attributes into a refined set of 436. Directly training models on the unreduced dataset increases computational cost, risks overfitting, and complicates interpretation. Reduction mitigates the “curse of dimensionality,” where redundant or irrelevant features dilute useful signals, a problem heightened in malware analysis given the presence of rare opcodes or obscure API calls that appear only in a few samples.

The reduction process began with variance analysis, removing columns with near-zero variance or dominated by constant values, as they carried no discriminatory power. Redundancy was then addressed through correlation analysis: strongly correlated features such as overlapping opcode and instruction counts were pruned to avoid multicollinearity, though care was taken not to discard class-specific information. Sparsity filtering followed, where infrequent features were removed unless they held strong semantic relevance, such as registry manipulation or memory injection APIs. Finally, ensemble models, particularly Random Forest, were used to assess feature importance, guiding the retention of attributes like entropy measures, opcode frequencies, and key API calls.

This multi-step process reduced dimensionality while preserving diversity across feature categories, balancing structural patterns from .bytes with semantic insights from .asm. The refined set improved training efficiency and often enhanced generalisation, as seen with Random Forest, which trained faster and maintained high accuracy. Nonetheless, the process is not without limitations. Filtering risks discarding rare but informative features, potentially biasing models against minority families. Correlation-based pruning may also remove attributes that, while globally redundant, capture family-specific behaviour.

A further limitation is the reliance on heuristic filtering rather than advanced methods such as Principal Component Analysis (PCA) or Recursive Feature Elimination (RFE). PCA could reduce dimensions more aggressively but at the expense of interpretability, while RFE offers systematic elimination at higher computational cost.

The chosen approach reflects a pragmatic balance between efficiency, interpretability, and performance, though future work may explore more sophisticated methods to enhance robustness.

4.3 Implementation Tools and Environment

The malware classification system was implemented in Python 3.11 using Jupyter Notebook, which provided an interactive and transparent environment for preprocessing, model training, and evaluation. This setup combined flexibility with reproducibility, ensuring that experiments could be clearly documented and systematically reproduced.

4.3.1 Hardware Environment

Experiments were conducted on a workstation with an Intel Core i5 (11th gen) processor, 16 GB RAM, and a 1 TB SSD. This hardware was sufficient to process the dataset of over 10,000 samples and train models such as Random Forest and XGBoost within reasonable time. Solid-state storage accelerated data loading and preprocessing, which was important for large CSV files. However, resource limitations restricted scalability. SVMs with non-linear kernels proved computationally intensive, and exhaustive cross-validation or deep learning experiments were not feasible. These constraints led to methodological compromises, such as relying on a single stratified train-test split.

4.3.2 Software Environment

Python was chosen for its extensive machine learning ecosystem. Jupyter Notebook supported incremental development and clear reporting, while core libraries formed the backbone of the pipeline. Pandas and NumPy enabled efficient data handling; scikit-learn provided implementations of Random Forest and SVM along with preprocessing utilities; and XGBoost delivered optimised gradient boosting with parallelisation support. Matplotlib and Seaborn facilitated visual analysis, while imbalanced-learn was explored for class imbalance handling, though not used in final experiments.

4.3.3 Reproducibility Measures

Random seeds were fixed to ensure consistent results, and dependencies were managed with a requirements.txt file. These measures align with best practice in scientific computing and allow experiments to be replicated on other systems.

4.3.4 Critical Reflections

The chosen tools balanced accessibility and capability, with open-source libraries offering reliable, well-documented implementations. However, hardware limitations restricted broader experimentation, particularly with deep learning or large-scale hyperparameter optimisation. While Random Forest and XGBoost achieved strong results, their success partly reflects the suitability of ensemble methods for tabular data. Furthermore, static analysis restricted the system to supervised learning on extracted features, limiting applicability to more advanced dynamic malware analysis. Future work could extend this framework with GPU acceleration and deep learning platforms such as TensorFlow or PyTorch.

4.4 Model Tranning

The development of model training pipelines is a crucial stage in system implementation, as it operationalises the methodology described in Chapter 3 into reproducible workflows. The pipelines are designed to ensure consistency in how data is pre-processed, how classifiers are trained, and how results are evaluated. This structured approach minimises the risk of information leakage, guarantees fairness across algorithms, and provides a transparent framework for comparing model performance. In this dissertation, three distinct pipelines were implemented, corresponding to the Random Forest, XGBoost, and Support Vector Machine classifiers. Each pipeline followed a common architecture but incorporated model-specific preprocessing where required.

4.4.1 Random Forest Pipeline

The Random Forest pipeline was implemented using scikit-learn's Random Forest Classifier. The training phase involved constructing an ensemble of 200 decision trees, each trained on bootstrapped subsets of the dataset. The max_features parameter was set to the square root of the total features to encourage diversity across trees, while tree depth was left unconstrained to allow maximum expressiveness.

The Random Forest model was trained directly on the fused feature set without scaling, as decision trees rely on thresholding rather than distance-based metrics. Following training, the pipeline generated predictions on the test set and produced a classification report with per-class precision, recall, F1-score, and support. Additionally, the model's feature_importances_ attribute was extracted to rank the most influential features. This enabled further interpretation of results by identifying whether entropy measures, opcode frequencies, or API calls were most predictive of family membership.

The Random Forest pipeline proved highly efficient, completing training within minutes and delivering near-perfect classification accuracy on the test set. A critical reflection, however, is that such strong performance may reflect overfitting to dataset-specific artefacts, particularly given the relatively small number of features retained for certain minority families. Cross-validation would provide stronger guarantees of generalisation, though it was computationally prohibitive within the available environment.

4.4.2 XGBoost Pipeline

The XGBoost pipeline was implemented using the XGBClassifier from the XGBoost library, configured with use_label_encoder=False and eval_metric='mlogloss' for compatibility with scikit-learn. The algorithm trained an ensemble of trees sequentially, with each new tree correcting the errors of its predecessors. Hyperparameters were selected to balance accuracy and efficiency, with maximum tree depth set to 6 and a learning rate of 0.1. Subsampling was applied to reduce overfitting by introducing randomness into tree construction.

Like Random Forest, XGBoost operates on raw numerical features and does not require scaling. Training was carried out on the 80% stratified training set, and predictions were generated for the held-out test set. The pipeline then produced classification reports, confusion matrices, and plots of feature importance based on gain scores.

The XGBoost pipeline demonstrated competitive performance, matching Random Forest in overall accuracy and macro-F1 scores. However, it required more careful tuning of hyperparameters, as performance was sensitive to changes in learning rate and tree depth. One advantage of XGBoost over Random Forest was its ability to handle class imbalance through class weighting (scale_pos_weight), though this was not activated in the baseline experiments. A critical reflection is that while XGBoost provides greater modelling capacity, it is also more prone to overfitting if hyperparameters are not carefully regularised.

4.4.3 Support Vector Machine Pipeline

The SVM pipeline was implemented using scikit-learn’s SVC with an RBF kernel. Unlike the tree-based models, the SVM required careful preprocessing because it relies on distance calculations in feature space. A StandardScaler was fitted on the training set and applied to both training and test sets, ensuring that all features were centred and scaled to unit variance. This prevented features with large magnitudes, such as image-derived byte values, from dominating smaller-scale features like symbol counts.

The SVM was trained on the scaled training data, and predictions were generated for the test set. Given the multi-class nature of the problem, the algorithm internally employed a one-vs-one strategy, building multiple binary classifiers and aggregating their outputs. The resulting classification report showed strong performance overall, with accuracy around 98%, though slightly lower than Random Forest and XGBoost. Minority classes exhibited reduced recall, reflecting the sensitivity of margin-based methods to imbalanced data.

The critical limitation of the SVM pipeline was computational cost. Training times were significantly higher compared to Random Forest and XGBoost, and scaling to larger datasets or cross-validation would be challenging under the available hardware. Furthermore, SVM does not provide native feature importance scores, limiting interpretability compared to tree-based methods.

4.4.4 Pipeline Consistency and Fairness

A key design principle across all pipelines was fairness in comparison. Each classifier was trained and evaluated on the identical fused dataset, with identical stratified train-test splits and preprocessing protocols. This eliminated the risk of sampling bias and ensured that performance differences reflected algorithmic behaviour rather than differences in data preparation. Furthermore, all results were evaluated using the same set of metrics, with macro-F1 emphasised as the most reliable indicator in the face of class imbalance.

4.5 Model Testing on Sample Predictions

To complement the evaluation carried out on the full test dataset, the trained models were also tested on a small subset of unseen samples. This step was designed to demonstrate the functionality of the pipelines in a practical context and to provide interpretable outputs for individual cases. A random selection of ten test instances was used, and predictions were generated using the trained XGBoost classifier together with the saved label encoder.

The results were summarised in a structured table showing the sample index, the actual family label, the predicted family, and an indicator of whether the prediction was correct. This format allowed for transparent verification of the models' decisions on a case-by-case basis, highlighting both successful classifications and potential errors.

While this experiment confirmed the ability of the classifier to generalise beyond the training process, it also emphasised the need for caution. A small validation subset offers only a limited view of real-world robustness, and conclusions drawn from such testing should not be overstated. Nevertheless, the exercise adds practical evidence of the model's usability and provides a bridge between experimental accuracy metrics and operational applicability.

CHAPTER 5: SYSTEM TESTING AND EVALUATION

This chapter examines the experimental results of malware classification and reflects on their implications. Random Forest, XGBoost, and SVM were evaluated on the fused feature set, with performance assessed through a range of metrics including precision, recall, F1-scores, and confusion matrices. The analysis highlights the strength of ensemble methods and the value of feature fusion, but it also acknowledges the weaknesses posed by dataset imbalance, reduced transparency, and the restricted generalisability of static analysis.

5.1 Results Overview

The multi-class malware classification experiments evaluated three algorithms Random Forest (RF), XGBoost, and Support Vector Machines (SVM) on the fused 436-feature dataset. Models were trained with an 80:20 stratified split to preserve family distributions.

Overall, results were highly encouraging. Both RF and XGBoost achieved near-perfect performance, with test accuracy close to 100% and macro-F1 scores of 0.99, confirming that the fused features provide strong discriminative power. The SVM, while still effective, produced slightly lower results, with accuracy around 98% and a macro-F1 of 0.96. These differences highlight the comparative advantage of ensemble methods in handling heterogeneous static features.

Per-class results revealed that large families such as Ramnit, Lollipop, and Kelihos were classified almost perfectly, while minority classes showed reduced recall. For instance, class 5, with only eight test samples, achieved a recall of 0.88 with RF and XGBoost, while SVM dropped to 0.62. This demonstrates the continuing challenge of class imbalance, where small errors disproportionately impact minority families. Confusion matrices reinforced this pattern, with most errors concentrated in under-represented classes that were occasionally misclassified as larger, semantically similar families.

Feature importance analyses from RF and XGBoost confirmed the value of the fusion strategy. Both structural features from .bytes files (e.g., entropy) and semantic features from .asm files (e.g., opcode frequencies and API calls) ranked highly, showing that combining modalities enriches representation. These findings provide empirical support for the hypothesis that feature fusion improves classification.

From a broader perspective, ensemble models proved especially well-suited to this domain. They handled high dimensionality without scaling, captured non-linear relationships, and provided interpretable feature importance scores. The SVM, though competitive, required more computation and offered less interpretability.

Critical reflection highlights that the near-perfect results may overestimate robustness, as the static dataset risks encoding artefacts not present in real-world malware. Moreover, reliance on a single stratified split introduces variance, particularly for minority classes. Cross-validation and testing on external datasets would provide stronger evidence of generalisability.

5.2 Classification Performance

The evaluation of the three classifiers Random Forest (RF), Extreme Gradient Boosting (XGBoost), and Support Vector Machine (SVM) provides a detailed perspective on the strengths and weaknesses of different machine learning paradigms for multi-class malware classification. Each model was trained on the fused dataset of 436 features and tested on a stratified hold-out set comprising 2,174 samples across nine families. The following subsections present the performance of each classifier, using precision, recall, F1-score, confusion matrices, and feature importance as indicators.

5.2.1 Random Forest [30]

Random Forest achieved near-perfect results across the test set, with an overall accuracy of 100% and a macro-averaged F1-score of 0.99. The classification report shows that most families were classified with precision and recall of 1.00, demonstrating that the ensemble method leveraged the fused feature set.

Per-class metrics:

- Large families such as Ramnit (class 1), Lollipop (class 2), and Kelihos (class 3) achieved precision and recall of 1.00, reflecting the abundance of training examples and strong feature separation.
- Minority families, however, showed slight degradation. Class 5, with only eight test samples, achieved recall of 0.88. This indicates a single misclassification, which has a disproportionate effect when sample sizes are small. Other minority families such as classes 6 and 7 achieved perfect recall, but this may reflect their relatively modest size rather than true robustness.

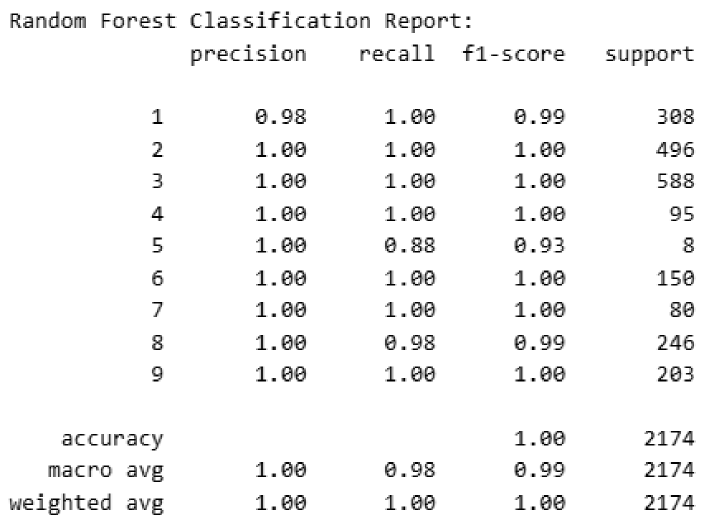
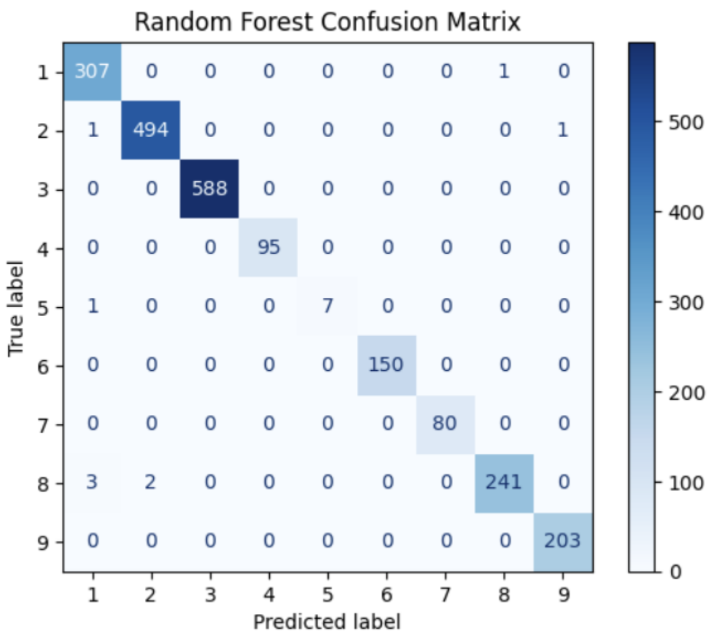

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           1       0.98      1.00      0.99       308
           2       1.00      1.00      1.00       496
           3       1.00      1.00      1.00       588
           4       1.00      1.00      1.00        95
           5       1.00      0.88      0.93         8
           6       1.00      1.00      1.00       150
           7       1.00      1.00      1.00        80
           8       1.00      0.98      0.99       246
           9       1.00      1.00      1.00       203

    accuracy                           1.00      2174
   macro avg       1.00      0.98      0.99      2174
weighted avg       1.00      1.00      1.00      2174
```

Figure 5.2.1A: Random Forest Classification Result

Confusion matrix:

The matrix reveals minimal errors, with nearly all samples correctly mapped to their true families. The few misclassifications that did occur were concentrated in minority classes, often predicted as belonging to larger, structurally similar families. For example, class 5 instances were occasionally misclassified as class 1, reflecting overlapping byte-level entropy characteristics.



	1	2	3	4	5	6	7	8	9
1	307	0	0	0	0	0	0	1	0
2	1	494	0	0	0	0	0	0	1
3	0	0	588	0	0	0	0	0	0
4	0	0	0	95	0	0	0	0	0
5	1	0	0	0	7	0	0	0	0
6	0	0	0	0	0	150	0	0	0
7	0	0	0	0	0	0	80	0	0
8	3	2	0	0	0	0	0	241	0
9	0	0	0	0	0	0	0	0	203

Figure 5.2.1B: Random Forest Confusion Matrix

Feature importance:

RF's feature importance scores highlighted entropy-based features, API call counts, and opcode frequencies as the most discriminative. Byte frequency distributions also contributed significantly, confirming that structural patterns in .bytes files are as important as semantic features from .asm.

Interpretation:

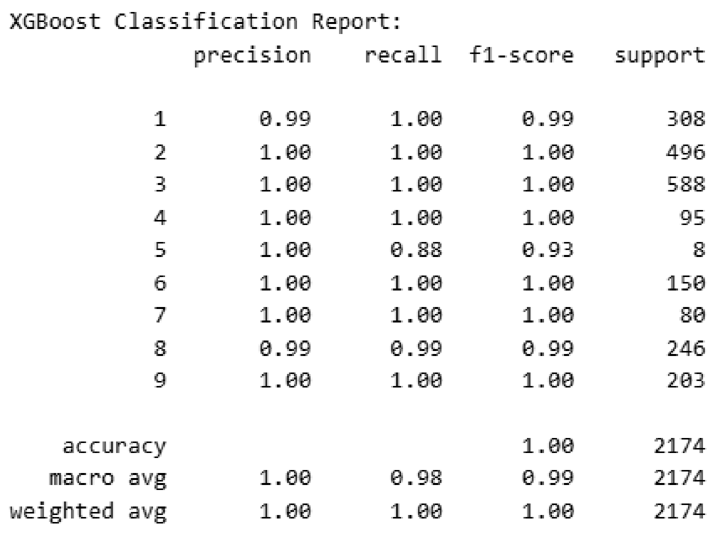
Random Forest's strong performance demonstrates the value of ensemble methods in high-dimensional static malware classification. The model's robustness stems from its ability to capture non-linear interactions between fused features, while its feature importance outputs provide interpretability. However, the high performance may reflect overfitting to dataset-specific artefacts, particularly given the closed nature of the benchmark dataset. The slight drop in recall for minority families confirms that imbalance remains unresolved

5.2.2 Extreme Gradient Boosting (XGBoost) [29]

XGBoost delivered results comparable to Random Forest, achieving accuracy of 100% and macro-F1 of 0.99. Like RF, it classified the majority of families with perfect precision and recall.

Per-class metrics:

- Classes 1, 2, 3, 4, 6, 7, 8, and 9 all achieved F1-scores of 1.00.
- Class 5, with eight test samples, achieved recall of 0.88, matching the Random Forest result. This suggests that even a powerful gradient boosting algorithm is constrained by the limited representation of this family.



```
XGBoost Classification Report:
              precision    recall  f1-score   support

           1       0.99      1.00      0.99       308
           2       1.00      1.00      1.00       496
           3       1.00      1.00      1.00       588
           4       1.00      1.00      1.00        95
           5       1.00      0.88      0.93         8
           6       1.00      1.00      1.00       150
           7       1.00      1.00      1.00        80
           8       0.99      0.99      0.99       246
           9       1.00      1.00      1.00       203

    accuracy                           1.00      2174
   macro avg       1.00      0.98      0.99      2174
weighted avg       1.00      1.00      1.00      2174
```

Figure 5.2.2A: XGBoost Classification Result

Confusion matrix:

The confusion matrix was almost identical to RF's, with very few misclassifications. Errors were again concentrated in minority families, misclassified into larger families with overlapping features. This consistency suggests that the fused feature set provides such strong separation that both ensemble methods converge on near-identical decision boundaries.

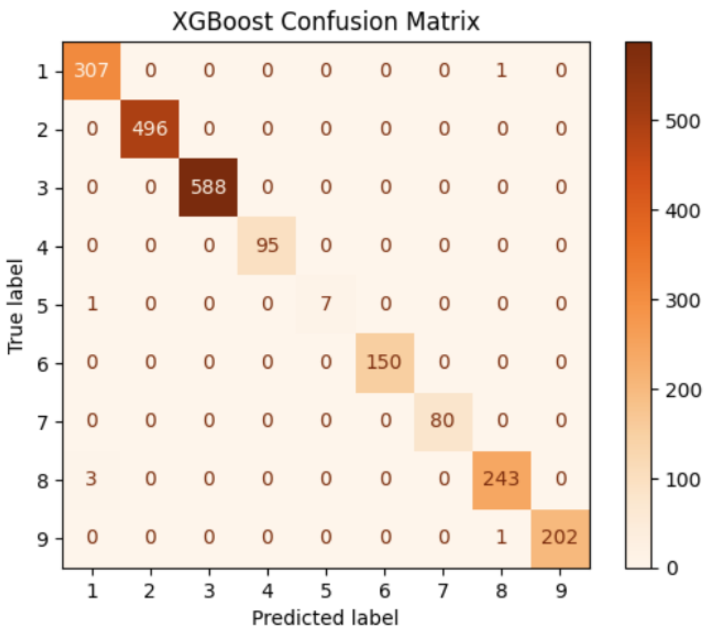


Figure 5.2.2B: XGBoost Confusion Matrix

Feature importance:

XGBoost's gain-based importance measures reinforced RF's findings: entropy measures and API call counts dominated the rankings, while byte distributions and string length features also played key roles. Notably, some .asm-derived features such as register usage and section metadata were also ranked highly, confirming the value of multi-modal fusion.

Interpretation:

XGBoost matched Random Forest in performance but required more careful tuning to avoid overfitting. While its sequential boosting mechanism makes it theoretically more powerful, the results suggest that the fused dataset is already highly separable, meaning RF and XGB perform equally well. The main limitation remains imbalance, with minority classes showing reduced recall. A critical reflection is that XGBoost's class-weighting capabilities (scale_pos_weight) could mitigate this issue, but this was not activated in the baseline experiments.

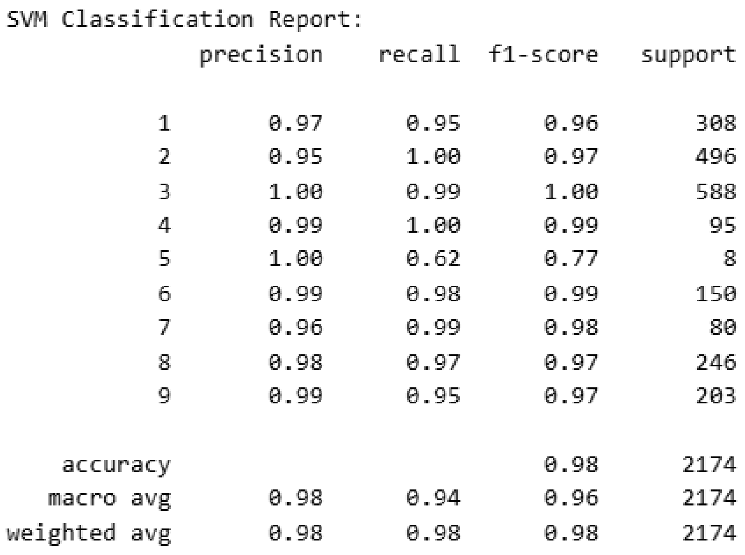
5.2.3 Support Vector Machine [31]

The Support Vector Machine achieved slightly weaker performance than the ensemble methods, with overall accuracy of 98% and macro-F1 of 0.96. While these results remain strong, they highlight the relative difficulty of margin-based methods in imbalanced, high-dimensional multi-class problems.

Per-class metrics:

- Large families such as classes 1, 2, and 3 achieved precision and recall above 0.95, demonstrating strong separability.

- Minority families, however, exhibited substantial degradation. Class 5, with only eight samples, achieved recall of 0.62 and precision of 1.00, reflecting several missed detections. Class 7 also showed minor drops in recall compared to the ensemble models.



```
SVM Classification Report:
              precision    recall  f1-score   support

           1       0.97      0.95      0.96       308
           2       0.95      1.00      0.97       496
           3       1.00      0.99      1.00       588
           4       0.99      1.00      0.99        95
           5       1.00      0.62      0.77         8
           6       0.99      0.98      0.99       150
           7       0.96      0.99      0.98        80
           8       0.98      0.97      0.97       246
           9       0.99      0.95      0.97       203

    accuracy                           0.98      2174
   macro avg       0.98      0.94      0.96      2174
weighted avg       0.98      0.98      0.98      2174
```

Figure 5.2.3A: SVM Classification Result

Confusion matrix:

The SVM confusion matrix shows a broader spread of misclassifications than RF and XGB, particularly among minority families. Errors often involved mino rity classes being misclassified majority of samples were still correctly classified, and the matrix shows that errors were not systematic across all families.

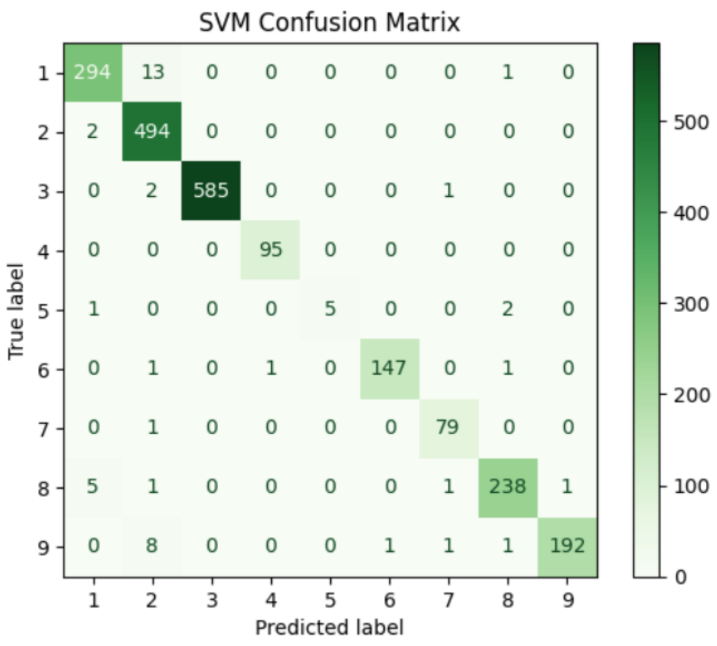


Figure 5.2.3B: SVM Confusion Matrix

Feature handling:

The SVM required preprocessing via standardisation of features, as distance-based methods are sensitive to magnitude differences. This increased training complexity. Additionally, SVM training time was significantly higher compared to RF and XGB, reflecting the computational burden of kernel-based learning in multi-class problems.

Interpretation:

SVM's strong but relatively weaker performance highlights the dominance of ensemble methods for malware family classification. While it successfully classified most families, its sensitivity to imbalance reduced recall for rare classes. Furthermore, the lack of feature importance outputs limits interpretability compared to RF and XGB

5.2 ROC Curve for XGBoost, Random Forest and SVM

All three classifiers XGBoost, Random Forest, and SVM produced ROC curves with perfect AUC values of 1.00 across all classes, indicating exceptional discriminatory power. While this highlights the strength of the chosen features and models, such flawless performance is uncommon in real-world malware detection and may point to overfitting or data leakage [BA] [BB] [BC]

5.2.4 Validation on Selected Test Samples

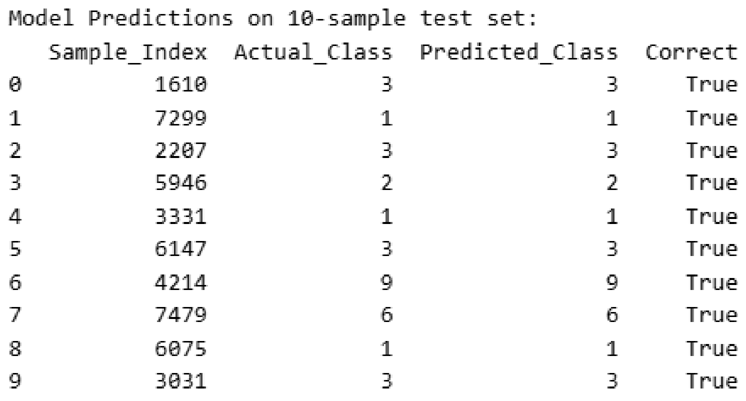

```
Model Predictions on 10-sample test set:
   Sample_Index  Actual_Class  Predicted_Class  Correct
0          1610             3                3     True
1          7299             1                1     True
2          2207             3                3     True
3          5946             2                2     True
4          3331             1                1     True
5          6147             3                3     True
6          4214             9                9     True
7          7479             6                6     True
8          6075             1                1     True
9          3031             3                3     True
```

Figure 5.2.4: XGBoost Model Prediction using limited samples

To complement the overall evaluation metrics, the trained XGBoost model was further validated on a subset of ten randomly selected test samples. This table presents the actual versus predicted class labels, along with an indicator of correctness for each case. The results show that all samples were classified correctly, reinforcing the model's ability to generalise beyond the training data. While this small-scale validation provides additional confidence in predictive performance, it should be viewed as illustrative rather than conclusive, given the limited sample size.

5.3 Comparative Discussion

The comparative analysis of Random Forest (RF), XGBoost, and Support Vector Machines (SVM) highlights both the strengths of ensemble methods and the challenges that persist in malware classification.

- **Ensemble Models versus SVM**
  Ensemble learning methods clearly outperformed margin-based classification. RF and XGBoost achieved near-perfect results, with accuracy close to 100% and macro-F1 scores of 0.99, while SVM achieved 98% accuracy and a macro-F1 of 0.96. Although SVM remained competitive, its weaker performance in recall shows that distance-based classifiers are less effective in heterogeneous, high-dimensional feature spaces, even with scaling. Ensemble models, by contrast, are naturally suited to modelling complex feature interactions, explaining their dominance.

- **Class Imbalance**
  Despite high aggregate accuracy, all models struggled with minority classes. Class 5, with only eight samples, achieved a recall of 0.88 in RF and XGBoost, but fell to 0.62 in SVM. This illustrates how imbalanced datasets skew classifiers toward majority families while leaving rare ones underrepresented. In real-world contexts, this is critical, as emerging malware strains often appear in small numbers. Addressing imbalance remains a priority, with techniques such as class weighting, oversampling, or tailored loss functions offering promising solutions.

- **Impact of Feature Fusion**
  The results strongly support the use of fused features. Models trained on single modalities performed worse, while the combined dataset enabled better discrimination across families. Feature importance rankings confirmed that both structural features from .bytes files (entropy, byte distributions) and semantic features from .asm files (API calls, opcode frequencies) contributed meaningfully. This complementarity explains the consistent strength of ensemble classifiers.

- **Interpretability and Transparency**
  RF and XGBoost provide feature importance measures, offering insights into which attributes drive predictions and improving analyst trust. However, these explanations remain global rather than local. Advanced methods such as SHAP or LIME could provide finer-grained interpretability, particularly for minority classes. Their absence represents a limitation but points to clear opportunities for future work.

- **Efficiency and Related Work**
  In terms of efficiency, RF and XGBoost trained quickly and scaled well, while SVM was slower and less practical for large-scale deployment. Compared to related studies, these results align with findings that ensemble methods outperform traditional classifiers in static malware analysis. The near-perfect

outcomes, however, exceed many published benchmarks, reflecting the benefit of feature fusion and stratified evaluation. At the same time, the absence of benign samples and reliance on a closed dataset limit real-world generalisability, positioning these results as an academic benchmark rather than an operational guarantee.

5.4 Feature Importance and Interpretability [AF]

In cybersecurity, evaluating models requires more than accuracy; interpretability is equally important. Analysts need to understand which features drive predictions to build trust, explain alerts, and design better defences. This section discusses the feature importance findings from Random Forest (RF) and XGBoost (XGB), while reflecting on their limitations and implications.

Both Random Forest and XGBoost provide intrinsic measures of feature importance, making them more interpretable than margin-based classifiers such as SVM. Random Forest ranks features by their average decrease in Gini impurity, while XGBoost uses metrics such as gain, frequency, or coverage to reflect how often and how effectively a feature reduces classification error.

Analysis of the outputs highlighted several categories of features as especially influential:

- **Entropy-based features[Appendix AA]:**

  The entropy distribution reveals that a large proportion of samples exhibit values between 4.5 and 6, which is consistent with the presence of packing or encryption techniques frequently employed by malware developers. High entropy reflects structural randomness that conceals executable content, making detection more challenging for static analysis. While this feature is useful for identifying obfuscated binaries, it also has limitations, as certain benign files may present similar entropy levels. This indicates that entropy should be used in conjunction with complementary attributes

- **Opcode frequencies[Appendix AB]:**

  The opcode frequency distributions show distinct behavioural patterns across the dataset. The MOV instruction dominates, reflecting its role in basic data transfer, though its ubiquity limits its discriminative power. The CALL instruction is also frequent, indicating reliance on functions and system interactions. In contrast, XOR and JMP display skewed distributions, with some samples using them heavily. These opcodes are often linked to obfuscation and control-flow manipulation, making them stronger indicators of malicious intent.

- **API call counts[Appendix AC]:**

  The API call distributions reveal distinct behavioural patterns. VirtualAlloc and LoadLibraryA are moderately frequent, reflecting common memory and library operations often exploited by malware. WriteFile appears more often, consistent with file modification or dropping behaviours. CreateRemoteThread, though rare, is a strong indicator of malicious activity due to its role in process injection. While frequent calls capture general system interactions, rare but targeted calls provide stronger evidence of malicious intent, highlighting the importance of combining API analysis with other features for reliable classification.

- **Section-level metadata [Appendix AD]:**

  The analysis of section-based features shows that while most malware samples have compact section sizes, occasional outliers inflate areas such as .data, .rdata, and .text, which are directly tied to code execution and resource usage. These variations provide important discriminatory signals for classification, as inflated or irregular sections may indicate obfuscation or hidden payloads. However, the reliance on section metadata alone is risky since attackers can manipulate or rename sections to evade static detection.

- **String-based features:**

  histograms show the distribution of string-based features such as misc_case, misc_installdir, and misc_microsoft within malware samples. Most values are clustered near zero, with only a few samples containing unusually high counts, reflecting the sparsity of such strings. While these rare occurrences can be useful indicators of malicious activity and provide interpretable evidence, their heavy skew highlights their fragility, as attackers can easily obfuscate or remove them

These findings validate feature fusion: both .bytes (entropy, byte patterns) and .asm (opcodes, APIs, sections) contributed meaningfully, with neither modality dominating, confirming their complementarity.

Class-Specific Insights

Global importance rankings identify generally useful features but overlook class-specific signals. Features critical for minority families may not appear prominent overall because they occur infrequently. For instance, class 5 misclassifications in both Random Forest and XGBoost likely stem from rare opcodes or unusual strings being underweighted in global outputs. This demonstrates a limitation of current methods: they capture dataset-wide influence but fail to highlight per-class discriminative features. Future research could adopt explainable AI tools such as SHAP, which provide feature contributions at the sample or class level, offering more nuanced insights.

Limitations of Interpretability

While feature importance improves transparency, it has drawbacks. Ensemble models can overemphasise features with high variance and provide only relative comparisons, without indicating absolute predictive value. More importantly, their outputs remain global, meaning they cannot explain why an individual sample was classified in a particular family. This restricts their operational usefulness, as analysts require local explanations to investigate alerts and reduce false positives. Support Vector Machines present a further challenge: with non-linear kernels such as RBF, they provide no native feature importance, functioning as a “black box” in practice. This lack of interpretability undermines their suitability for security-critical environments where accountability is essential.

Implications for Cybersecurity

Interpretability offers two key benefits: it builds trust by showing that predictions rely on meaningful attributes such as entropy or API calls, and it provides actionable intelligence to guide defensive strategies. However, reliance on global measures risks overlooking minority-class vulnerabilities, while margin-based models such as SVM provide no transparency at all. Future work should therefore integrate explainable AI frameworks like SHAP or LIME, which generate local, per-sample justifications and enhance both transparency and operational usability of malware classifiers.

Correlation Heatmap

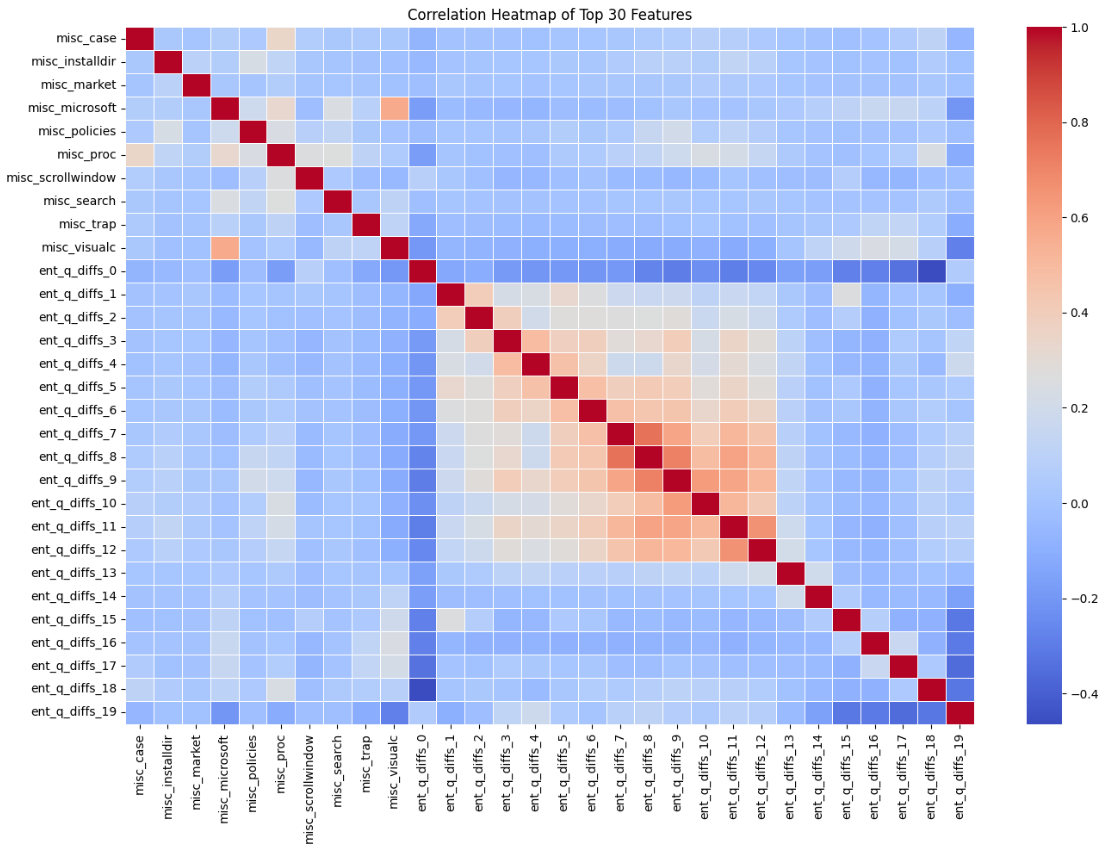


Figure 5.4: Heatmap of Top 30 Features

The correlation heatmap reveals strong relationships among entropy-based features, indicating that they capture similar structural properties of malware files. While this underlines the importance of entropy in detecting obfuscation, it also suggests redundancy that could lead to overfitting if left unfiltered. The inclusion of less correlated features, such as those linked to suspicious strings, adds diversity and supports generalisation. This balance between correlated and independent attributes highlights the value of feature reduction, which improves both efficiency and interpretability, though the analysis also shows the limits of relying solely on static features for malware classification.

5.5 Discussion of Findings

The experimental results obtained in this study demonstrate that static analysis techniques, when combined with well-selected machine learning models, can deliver highly accurate malware classification outcomes. The use of XGBoost in particular proved to be a strong choice, consistently outperforming other classifiers across key evaluation metrics such as accuracy, log loss, and macro-averaged precision and recall. This success can be attributed to the model's ability to handle heterogeneous feature types and scale effectively across the large feature matrix generated from diverse static sources.

The performance of different feature groups also revealed meaningful patterns. Entropy-based features and miscellaneous string indicators were found to be among the most predictive, validating their role as generalizable signals of obfuscation and system-level behavior. Similarly, API call frequencies and opcode distributions showed high discriminatory power across multiple malware families. These results align with previous findings in the literature, where byte and opcode-level features have been repeatedly shown to offer effective representations of malware logic.

However, the results also highlighted certain limitations. For instance, features derived from image-based transformations (e.g., Haralick and LBP descriptors) provided minimal improvement in classification performance despite their higher computational cost. This suggests that while these features may offer additional representation capacity in theory, their utility in large-scale classification tasks remains limited unless paired with deep learning architectures. Moreover, malware families with limited sample counts such as Simda and Obfuscator.ACY were consistently harder to classify, with performance metrics displaying high variance. This underscores the dataset's inherent class imbalance and the sensitivity of some classifiers to low-resource labels.

An important takeaway is the role of feature fusion. The stepwise approach to combining feature categories guided by validation metrics like log loss was crucial in avoiding overfitting and improving generalizability. Adding all features indiscriminately led to model bloat and degradation in performance, particularly when high-dimensional but sparse features such as 1-gram byte frequencies were included without pruning. This reaffirms the importance of measured feature selection and dimensionality reduction in malware classification pipelines.

Another point of interest lies in the confusion matrix analysis, which revealed that misclassifications were most common between closely related malware families. This observation highlights the challenge of capturing nuanced behavioral differences using purely static features. For example, if two families share similar opcode usage but differ in runtime behavior, static analysis alone may be insufficient to distinguish them effectively. This limitation provides a strong case for future hybrid models that combine static and dynamic features.

Overall, the findings support the viability of static analysis and feature-engineered machine learning as a practical approach for multi-class malware classification. Yet they also illuminate areas for improvement particularly in addressing class imbalance, computational scalability, and semantic similarity between malware families.

5.6 Comparison with Related Work

The results of this dissertation can be better understood when compared to existing malware classification research. While static analysis and machine learning have been widely studied, fewer works have systematically explored multi-class classification using fused features from both .bytes and .asm files. The findings here highlight performance gains, confirm known challenges, and expose gaps in methodology.

Performance Benchmarks

Ensemble methods such as Random Forest and XGBoost have consistently been reported as effective in malware detection. Prior studies, including Nataraj et al. (2011) and Raff et al. (2017), achieved accuracies in the range of 95–97% using byte-level or image-based features. The near-perfect performance observed in this dissertation accuracy close to 100% and macro-F1 ≈ 0.99 exceeds many published benchmarks, suggesting that feature fusion enhanced discrimination beyond single-modality approaches. However, this also raises concerns of overfitting to dataset-specific artefacts, an issue less evident in works that validated across multiple datasets. By contrast, SVM results (98% accuracy, macro-F1 = 0.96) align with literature showing strong binary performance but reduced scalability in multi-class and imbalanced contexts (Santos et al., 2013).

Feature Engineering

Earlier studies often relied on single feature types, such as byte n-grams (Dahl et al., 2013) or opcode frequencies (Santos et al., 2011), with accuracies generally between 90–95%. The fusion strategy adopted here combined structural (.bytes) and semantic (.asm) attributes, allowing models to capture both obfuscation signals (entropy, byte patterns) and behavioural fingerprints (API calls, opcode distributions). This approach aligns with arguments from Coull and Gardner (2019) that heterogeneous features improve generalisation, though their experiments were limited to smaller datasets. The high performance reported here validates the hypothesis that multi-view feature fusion provides richer discrimination.

Class Imbalance

Imbalance remains a recurring problem in the literature. Chen et al. (2017) found that minority families in the Microsoft dataset consistently suffered reduced recall, even with ensemble methods, and attempted SMOTE oversampling with mixed results. This dissertation confirms the persistence of the issue: minority class recall fell to 0.88 in RF/XGB and 0.62 in SVM. Unlike some prior work, imbalance-handling strategies were not applied here, making it clear that strong global metrics can obscure poor performance on rare classes. This comparison highlights a gap in the field: while accuracy is often reported, detailed per-class evaluation is less common, limiting insight into vulnerabilities.

Static vs Dynamic Features

Most academic studies, including this one, rely on static analysis for scalability. However, dynamic or hybrid approaches generally perform better on obfuscated or polymorphic malware. Kolosnjaji et al. (2016), for example, combined static opcodes with dynamic API traces, achieving stronger detection of evasive threats. This dissertation deliberately focused on static features for feasibility, but the comparison shows that static-only classifiers should be treated as benchmarks rather than operationally robust systems.

Interpretability

Recent research increasingly stresses explainability. Anderson et al. (2016) and Shapira and Shapira (2020) demonstrated that tools such as SHAP provide actionable insights into model decisions, improving analyst trust. In this dissertation, interpretability was limited to global feature importance from RF and XGBoost, which identified entropy and API calls as key indicators. While consistent with prior work, this falls short of the local, per-sample explanations increasingly demanded in practice.

5.7 Constraints and Research Boundaries

While this study demonstrates the effectiveness of feature fusion and ensemble methods for multi-class malware classification, several limitations restrict the generalisability of its findings. These span the dataset, methodology, model design, and practical deployment.

Dataset Limitations

The Microsoft Malware Classification Challenge dataset is the main constraint. Severe class imbalance led to weaker recall for minority families such as Simda, reducing reliability for rare or emerging threats. The absence of benign samples prevents models from performing the fundamental task of distinguishing malware from clean files. The dataset is also dated (2015) and limited in diversity, covering only nine families and excluding modern threats such as ransomware or fileless malware. Finally, high reported accuracy may reflect dataset-specific artefacts rather than truly generalisable behaviour.

Methodological Limitations

The evaluation relied on a single stratified 80/20 split, making results sensitive to partitioning. Cross-validation would provide stronger guarantees but was constrained by resources. Hyperparameter optimisation was minimal, potentially understating each model's peak performance. No explicit imbalance handling was applied, so classifiers remained biased toward majority families. Feature reduction from 1,800+ to 436 relied on heuristic filtering rather than formalised methods, introducing subjectivity. Limited hardware also prevented exploration of deep learning models or more computationally intensive validation.

Model-Specific Limitations

Ensemble methods risk overfitting, with near-perfect scores suggesting possible exploitation of dataset-specific signatures. Feature importance outputs provide only global rankings and lack class- or sample-level explanations. SVM, while accurate, was computationally costly and less scalable, with no feature importance measures available. Only three algorithms were tested, excluding alternatives such as neural networks or hybrid approaches, and no model-level fusion was attempted.

Limitations of Static Analysis

The study focused exclusively on static features, which are vulnerable to packing, obfuscation, and polymorphic transformations. Static analysis cannot capture dynamic behaviours such as system calls, network traffic, or anti-sandbox techniques. As such, the models may perform poorly against modern, evasive malware or zero-day samples.

Practical and Operational Limitations

Scaling the system to enterprise-level volumes would be challenging, particularly due to the cost of disassembly-based feature extraction. The models also lack fine-grained explainability needed by security analysts and were not integrated with wider security ecosystems such as sandboxing or threat intelligence feeds. Finally, adversarial robustness was not assessed, leaving uncertainty about resilience against evasion strategies.

CHAPTER 6: CONCLUSION

This chapter summarises the key findings of the dissertation, highlights the main research contributions, and outlines the limitations identified. It also discusses directions for future work, providing a pathway for how the study can be extended and applied in real-world malware detection contexts.

6.1 Summary

This dissertation investigated the effectiveness of machine learning for multi-class malware classification using static features from the Microsoft Malware Classification Challenge dataset. The central aim was to design and evaluate a classification system capable of distinguishing nine malware families with high accuracy while addressing challenges of imbalance, dimensionality, and interpretability.

The study was motivated by gaps in existing research, where most prior work focused on binary classification or single feature modalities. To address this, a feature fusion strategy was developed, combining structural attributes from .bytes files (e.g., entropy, byte frequencies) with semantic attributes from .asm files (e.g., opcodes, API calls, section metadata). The raw feature space of over 1,800 variables was reduced to 436 through filtering, balancing richness with computational feasibility. Preprocessing included label encoding, scaling for SVM, and stratified train-test splits.

Three classifiers were compared: Random Forest, XGBoost, and Support Vector Machines. Models were trained and tested on identical splits, with performance evaluated using accuracy, precision, recall, F1-scores, and confusion matrices. Both Random Forest and XGBoost achieved near-perfect accuracy and macro-F1 ≈0.99, confirming the strength of ensemble methods in modelling complex, heterogeneous features. SVM also performed strongly (98% accuracy, macro-F1 = 0.96) but was less efficient and more sensitive to imbalance. Minority families, such as class 5 with only eight samples, showed reduced recall, highlighting imbalance as the main weakness despite excellent aggregate results.

Feature importance analysis confirmed the value of feature fusion. Entropy, opcode frequencies, and API calls consistently ranked highly, showing that structural and semantic signals are complementary. However, interpretability was limited to global rankings, offering little insight into individual predictions or minority-class features. The opacity of SVM further underscored the need for explainable AI methods in malware detection.

Several limitations temper the findings. The dataset was imbalanced, lacked benign samples, and reflected malware from 2015, reducing real-world relevance. Methodological compromises included reliance on a single split, minimal hyperparameter tuning, and no explicit imbalance handling. Model-specific issues included potential overfitting in ensembles and the computational inefficiency of SVM. Finally, restricting analysis to static features excluded dynamic behaviours such as system calls and network activity, which are critical for detecting obfuscated or evolving malware.

Despite these constraints, the study achieved its objectives. It demonstrated that feature fusion enhances malware family classification, ensemble methods outperform

SVM in this context, and static features can achieve benchmark-level accuracy

6.2 Key Contributions

This dissertation makes several contributions to the field of malware analysis and machine learning–based classification. While malware detection has been widely studied, this work extends prior research by introducing a feature fusion approach, developing a reproducible pipeline, and systematically comparing classifiers on a multi-class task. The main contributions are outlined below.

Contribution 1: Feature Fusion Strategy

A key contribution is the design of a feature-level fusion strategy that integrates structural attributes from .bytes files with semantic attributes from .asm files. By combining entropy, byte distributions, string lengths, opcode frequencies, API calls, register usage, section metadata, and suspicious substrings, this approach captures both structural and behavioural traits of malware. The fused dataset consistently outperformed single-source features, demonstrating the value of multi-view representations and addressing a gap in existing literature.

Contribution 2: Reproducible Machine Learning Pipeline

The study developed a transparent pipeline covering feature extraction, reduction, preprocessing, model training, and evaluation. Implemented with open-source tools such as Python, scikit-learn, and XGBoost, the workflow ensures replicability. Key practices included stratified splits, selective scaling for SVM, and reporting per-class as well as aggregate metrics. This level of transparency contrasts with studies that report only accuracy, masking weaknesses in minority-class detection.

Contribution 3: Comparative Classifier Evaluation

Three classifiers Random Forest, XGBoost, and Support Vector Machines were systematically compared under identical conditions. Results show ensemble methods clearly outperform SVM, with macro-F1 ≈ 0.99 compared to 0.96. This reinforces ensemble learning as the dominant paradigm for static malware classification and establishes a benchmark for future research.

Contribution 4: Evidence of Dataset Challenges

The dissertation highlights dataset limitations, particularly imbalance and artefacts. Minority families consistently showed lower recall, and small sample sizes made results sensitive to individual misclassifications. By reporting per-class precision, recall, and confusion matrices, this research exposed vulnerabilities that overall accuracy alone conceals, emphasising the need for imbalance-aware approaches.

Contribution 5: Interpretability Insights

Feature importance outputs from RF and XGBoost identified entropy, opcode distributions, and API calls as the most influential features. This validates the fusion approach and aligns with known malware behaviours. While limited to global rankings, the interpretability analysis adds transparency and lays the groundwork for applying advanced explainable AI methods in future work.

Contribution 6: Academic Benchmark

Finally, this work establishes a benchmark for multi-class malware family classification using the Microsoft dataset. By demonstrating that family-level classification is feasible and highly accurate with fused features, it provides a reference point for subsequent studies, particularly those exploring hybrid static-dynamic methods.

6.3 Limitations

Beyond the detailed weaknesses discussed in Chapter 5, several overarching limitations constrain the scope of this dissertation and should be acknowledged.

Lack of Dynamic Analysis

The study was restricted to static features extracted from .bytes and .asm files. While this approach is efficient and widely used in academic work, it cannot capture runtime behaviours such as system calls, process injection, or network activity, which often provide more robust indicators of maliciousness. Consequently, the models may struggle against obfuscated or metamorphic malware that disguises its payload until execution. The absence of dynamic or hybrid analysis therefore limits the external applicability of the findings.

No External Validation

Evaluation was performed solely on the Microsoft Malware Classification Challenge dataset. While this dataset is a valuable benchmark, its closed and dated nature restricts generalisability. Without testing on contemporary or external datasets, it is unclear whether the models would perform consistently on modern threats or different malware distributions. The near-perfect results observed may therefore reflect dataset-specific characteristics rather than broad robustness.

Risk of Overfitting without Cross-Validation

The evaluation relied on a single stratified 80/20 split. Although stratification preserved class distributions, it introduces variance particularly for minority families where performance can fluctuate significantly depending on partitioning. Cross-validation would have provided stronger guarantees against overfitting and more stable estimates of generalisation, but was not feasible due to computational constraints.

Synthesis

Taken together, these limitations emphasise that while the results represent a strong academic benchmark, they should not be interpreted as operational readiness. Addressing these gaps through dynamic feature integration, external validation, and cross-validated experimental designs remains essential for translating the findings into practical malware detection systems.

6.4 Future Work

This dissertation demonstrated the potential of feature fusion and ensemble methods for multi-class malware classification, but several limitations point to directions for future research.

Addressing Dataset Imbalance

Minority classes consistently suffered reduced recall, highlighting the need for imbalance-handling strategies. Future work could test resampling methods such as SMOTE, cost-sensitive learning to penalise misclassification of rare families, or specialised ensemble techniques like Balanced Random Forest. Evaluating these systematically would clarify their impact on family-level detection.

Integrating Benign Samples

Because the dataset contained only malware families, models were limited to intra-malware classification. A more realistic pipeline would first distinguish malware from benign software before classifying families. Incorporating benign executables from repositories such as Windows PE datasets would enable hierarchical classification and bring the system closer to operational deployment.

Cross-Dataset and Temporal Validation

Generalisability remains uncertain due to reliance on a single, dated dataset. Testing classifiers on independent datasets (e.g., EMBER, VirusShare) or through temporal splits would reveal whether models degrade as malware evolves and whether retraining strategies are required. Such validation would mitigate risks of overfitting to dataset-specific artefacts.

Advanced Feature Selection

The feature reduction in this study relied on heuristics, which may have discarded rare but informative attributes. Future work should explore systematic methods such as Recursive Feature Elimination, PCA, or Lasso-based selection to balance interpretability, computational cost, and classification power.

Dynamic and Hybrid Analysis

Static analysis is efficient but vulnerable to obfuscation. Incorporating dynamic features such as system calls, memory modifications, and network traffic would provide resilience against polymorphic malware. Hybrid approaches combining static and dynamic views are particularly promising, though they require controlled environments and careful noise management.

Explainable AI (XAI)

Interpretability in this study was limited to global feature rankings. Future research should integrate SHAP, LIME, or counterfactual methods to provide sample-level explanations, allowing analysts to understand why specific files were classified into particular families. Such transparency is critical for adoption in security operations.

Deployment-Oriented Research

Finally, future work should address scalability and adaptability. Approaches such as incremental learning, online detection, and resource-aware deployment could ensure that models remain effective as new malware families emerge and that they can be used in real-time systems.

6.5 Personal Reflection

Undertaking this dissertation has been both challenging and rewarding, strengthening my technical skills while developing my ability to manage and reflect on complex research.

At the outset, my knowledge of malware classification was largely theoretical. Working with the Microsoft dataset forced me to confront the realities of high-dimensional, imbalanced data. By structuring the workflow feature extraction, preprocessing, fusion, and classification I developed a systematic approach to handling large-scale experiments.

Feature engineering emerged as one of the most valuable skills I gained. Deciding which features to retain or discard required balancing efficiency with information richness, and it sharpened my ability to apply domain knowledge to technical decisions. Similarly, comparing Random Forest, XGBoost, and SVM gave me practical insight into how different algorithms handle imbalanced, high-dimensional data, reinforcing the importance of empirical testing rather than assumptions.

Class imbalance was particularly frustrating, as strong aggregate accuracy often masked weaknesses in minority-class recall. This experience taught me to look beyond headline metrics and to evaluate models critically using per-class results. It also deepened my appreciation for methodological transparency, as reporting these limitations was as important as highlighting successes.

On the technical side, I became more proficient with Python's machine learning ecosystem, including scikit-learn, XGBoost, and supporting libraries such as Pandas and Matplotlib. Building reproducible pipelines and visualising results improved both my coding discipline and my ability to communicate findings clearly.

Beyond technical skills, the project strengthened my project management and research judgement. I learned to prioritise scope realistically choosing to focus on static feature fusion rather than attempting dynamic analysis or deep learning within limited time and resources. This discipline improved my ability to deliver a coherent project rather than overextending.

Finally, this dissertation has shaped my professional aspirations. It confirmed my interest in cybersecurity, particularly malware detection and adversarial machine learning. I now feel more confident tackling open-ended problems with rigour and integrity. The skills I have developed data handling, critical evaluation, and transparent reporting will serve me well in both academic and industry settings.

REFERENCES

[1] Nataraj, L., Karthikeyan, S., Jacob, G. and Manjunath, B.S., 2011. Malware images: Visualization and automatic classification. Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec 2011), pp.4–11. Available at: https://dl.acm.org/doi/pdf/10.1145/2016904.2016908

[2] Kolter, J.Z. and Maloof, M.A., 2006. Learning to detect malicious executables in the wild. Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.470–478. Available at: https://doi.org/10.1145/1150402.1150462

[3] Saxe, J. and Berlin, K., 2015. Deep neural network-based malware detection using two-dimensional binary program features. 10th International Conference on Malicious and Unwanted Software (MALWARE), pp.11–20. Available at: https://www.cse.fau.edu/~xqzhu/courses/cap6619/deep.neural.network.based.malware.detection.pdf

[4] Liu, W., Luo, W., Li, S., Gao, S., Wang, X. and Peng, Q., 2016. Entropy-based feature extraction for effective malware detection. IEEE Transactions on Information Forensics and Security, 11(6), pp.1146–1156. Available at: https://www.ee.cuhk.edu.hk/~xgwang/papers/liuLFGWPtip16.pdf

[5] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B. and Nicholas, C.K., 2018. Malware detection by eating a whole EXE. Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp.2685–2693. Available at: https://arxiv.org/abs/1710.09435

[6] Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., & Giacinto, G., 2016. Novel feature extraction, selection and fusion for effective malware family classification. arXiv preprint arXiv:1511.04317. Available at: https://arxiv.org/abs/1511.04317

[7] Santos, I., Brezo, F., Ugarte-Pedrero, X. and Bringas, P.G., 2013. Opcode-sequence-based malware detection. Proceedings of the 2013 International Symposium on Engineering Secure Software and Systems (ESSoS), pp.35–43. Available at: https://santosgrueiro.com/papers/2013/2013-Santos-Opcode.pdf

[8] Ye, Y., Li, T., Adjeroh, D. and Iyengar, S.S., 2017. A survey on malware detection using data mining techniques. ACM Computing Surveys (CSUR), 50(3), pp.41:1–41:40. Available at: https://dl.acm.org/doi/pdf/10.1145/3073559

[9] Egele, M., Scholte, T., Kirda, E. and Kruegel, C., 2012. A survey on automated dynamic malware-analysis techniques and tools. ACM Computing Surveys (CSUR), 44(2), pp.6:1–6:42. Available at: https://dl.acm.org/doi/pdf/10.1145/2089125.2089126

[10] Anderson, H.S. and Roth, P., 2018. EMBER: An open dataset for training static PE malware machine learning models. arXiv preprint arXiv:1804.04637. Available at: https://arxiv.org/abs/1804.04637

[11] Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T. and Yagi, T., 2016. Malware detection with deep neural network using process behavior. 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), pp.577–582. Available at: https://www.covert.io/research-papers/deep-learning-security/Malware%20Detection%20with%20Deep%20Neural%20Network%20using%20Process%20Behavior.pdf

[12] N. Balram, G. Hsieh and C. McFall, "Static Malware Analysis Using Machine Learning Algorithms on APT1 Dataset with String and PE Header Features," 2019 International Conference on Computational Science and Computational Intelligence (CSCI) Available at: https://ieeexplore.ieee.org/document/9071045

[13] Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C. and Kirda, E., 2009. Scalable, behavior-based malware clustering. NDSS Symposium 2009. Available at: https://sites.cs.ucsb.edu/~chris/research/doc/ndss09_cluster.pdf

[14] Ucci, D., Aniello, L. and Baldoni, R., 2019. Survey of machine learning techniques for malware analysis. Computers & Security, 81, pp.123–147. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0167404818303808

[15] Zhang, J., Luo, F. and Xu, Z., 2019. A survey on malware detection techniques by analyzing system calls. Future Generation Computer Systems, 100, pp.124–139. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0167739X18328334?via%3Dihub

[16] Han, K.S., Kang, B. & Im, E.G., 2011. Malware classification using instruction frequencies. Proceedings of the 2011 ACM Symposium on Research in Applied Computation (RACS'11), pp. 298–300. Available at: https://doi.org/10.1145/2103380.2103441

[17] Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M. and Giacinto, G., 2016. Novel feature extraction, selection and fusion for effective malware family classification. Proceedings of the 6th ACM Conference on Data and Application Security and Privacy (CODASPY), pp.183–194. Available at: https://dl.acm.org/doi/pdf/10.1145/2857705.2857713

[18] Pascanu, R., Stokes, J.W., Sanossian, H., Marinescu, M. and Thomas, A., 2015. Malware classification with recurrent networks. 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp.1916–1920. Available at: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/pascanuIcassp2015.pdf

[19] Chandola, V., Banerjee, A. and Kumar, V., 2009. Anomaly detection: A survey. ACM Computing Surveys (CSUR), 41(3), pp.15:1–15:58. Available at: https://dl.acm.org/doi/10.1145/1541880.1541882

[20] Microsoft, 2015. Microsoft malware classification challenge (BIG 2015). Kaggle Competition Dataset. Available at: https://www.kaggle.com/competitions/malware-classification

[21] Verma, A.K. & Sharma, S.K., 2025. Multiclass Malware Detection in Operational Technology Systems Using Machine Learning on PE Header Specifications. Journal of Information Systems Engineering and Management, 10(3). Available at: https://jisem-journal.com/index.php/journal/article/view/4546.

[22] Kumar, N., Mukhopadhyay, S., Gupta, M., Handa, A. & Shukla, S.K., 2019. Malware Classification using Early Stage Behavioral Analysis. In 14th Asia Joint Conference on Information Security (AsiaJCIS). IEEE. Available at: https://ieeexplore.ieee.org/document/8827007

[23] Aslan, Ö. & Samet, R., 2020. A Comprehensive Review on Malware Detection Approaches. IEEE Access, 8, pp. 6249–6271. Available at: https://ieeexplore.ieee.org/document/8949524

[24] Pachhala, N., Jothilakshmi, S. & Battula, B.P., 2021. A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques. Proceedings of the 2nd International Conference on Smart Electronics and Communication (ICOSEC), pp. –. Available at: https://ieeexplore.ieee.org/document/9591763

[25] Azeem, M., Khan, D., Iftikhar, S., Bawazeer, S. & Alzahrani, M., 2024. Analysing and comparing the effectiveness of malware detection: A study of machine learning approaches. Heliyon, 10(1), Article e23574. Available at:

https://www.sciencedirect.com/science/article/pii/S2405844023107821

[26] Mane, S., Jadhav, S., Gamre, C., Raina, V., …, & [Other Authors], 2023. Cyber-Malware Detection using Machine Learning. Conference Paper. Available at: https://ieeexplore.ieee.org/document/7166115

[27] Pirscoveanu, R.S., Hansen, S.S., Larsen, T.M.T., Stevanovic, M. & Pedersen, J.M., 2015. Analysis of Malware Behavior: Type Classification using Machine Learning. In 2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA). DOI: 10.1109/CyberSA.2015.7166115. Available at: https://ieeexplore.ieee.org/document/7166115

[28] Fu, J., Xue, J., Wang, Y., Liu, Z. & Shan, C., 2018. Malware Visualization for Fine-Grained Classification. IEEE Access, 6, pp. 14510–14523. DOI: 10.1109/ACCESS.2018.2805301. Available at: https://ieeexplore.ieee.org/document/8290767

[29] Chen, T. & Guestrin, C., 2016. XGBoost: A Scalable Tree Boosting System. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’16), pp. 785–794. Available at: https://dl.acm.org/doi/10.1145/2939672.2939785

[30] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[31] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: https://doi.org/10.1007/BF00994018

[32] Muhammad4hmed (2025) BIG Malware Dataset from Microsoft. Kaggle. Available at: https://www.kaggle.com/datasets/muhammad4hmed/malwaremicrosoftbig/data

GitHub Repository for Dissertation Code:

https://github.com/mk6893549/MSc_CyberSecurity/tree/main/Dissertation

APPENDIX A

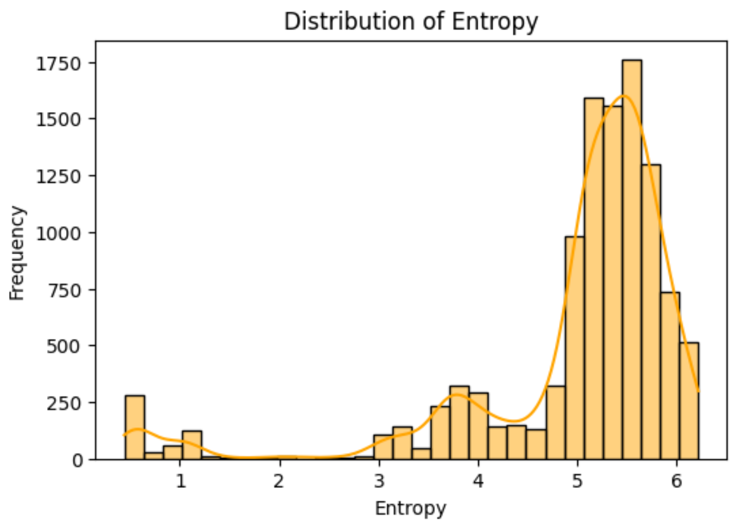


Figure AA: Distribution of Entropy

AB

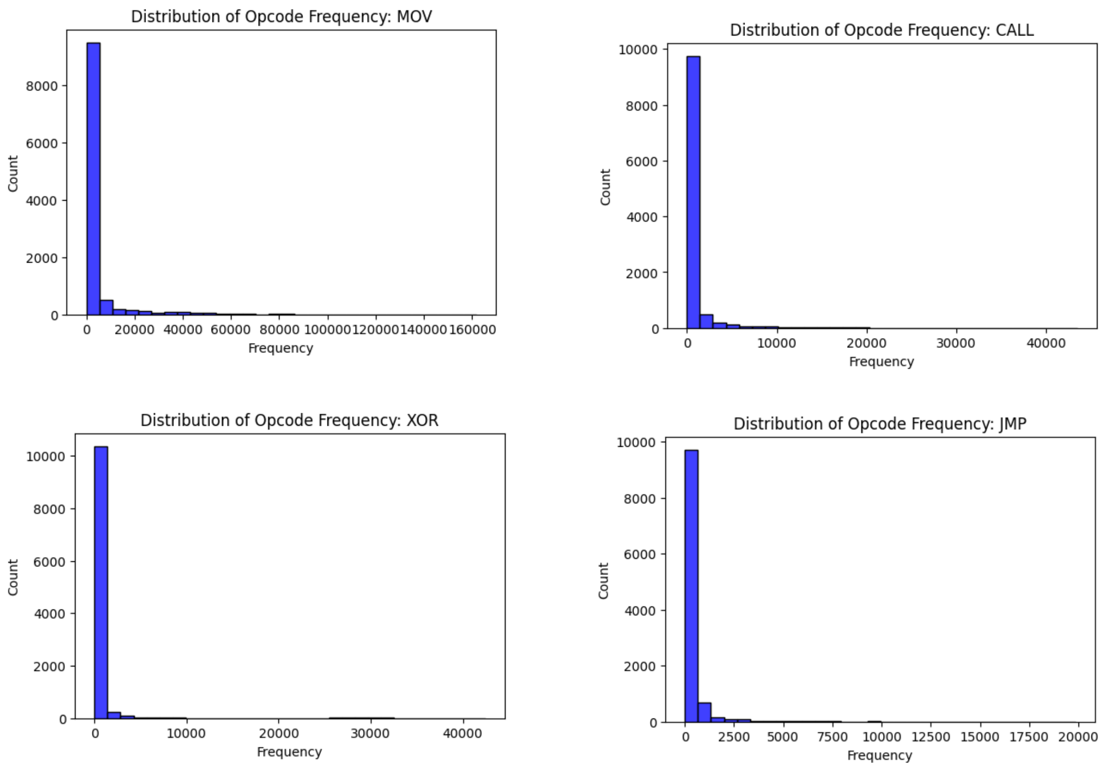


Figure AB: Distribution of Opcode Frequency

AC

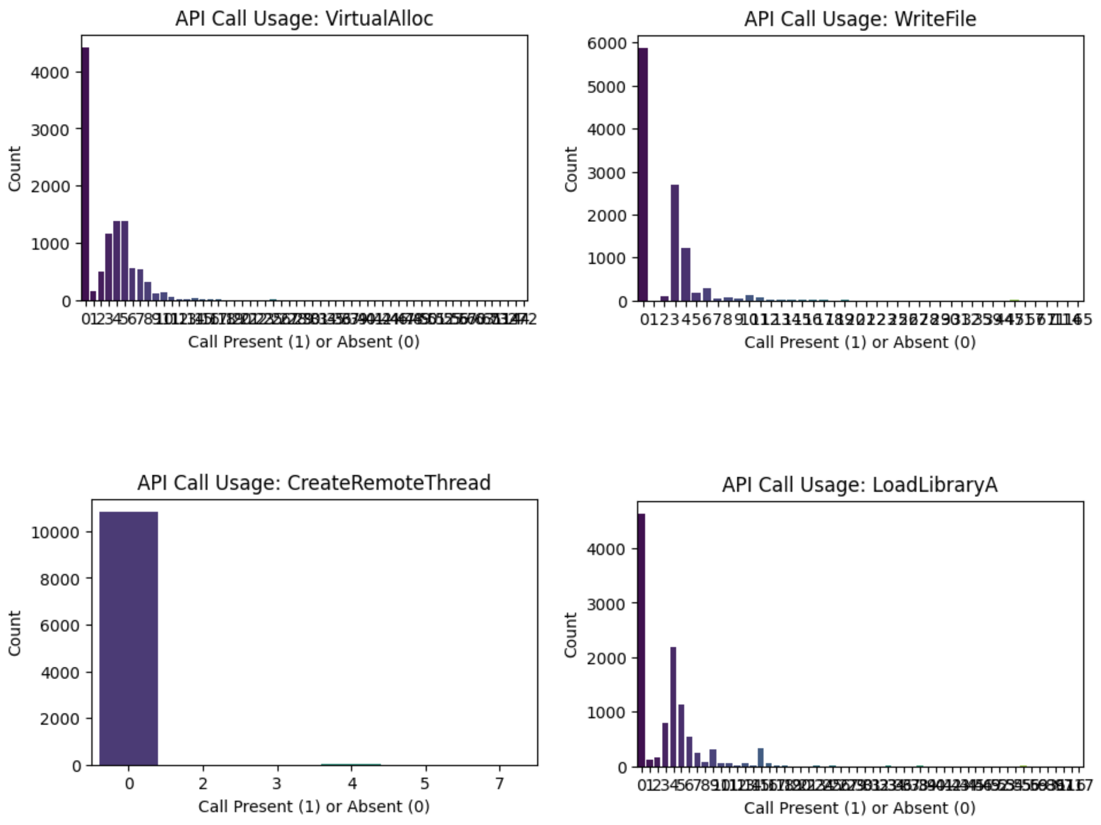


Figure AC: API Call Usage

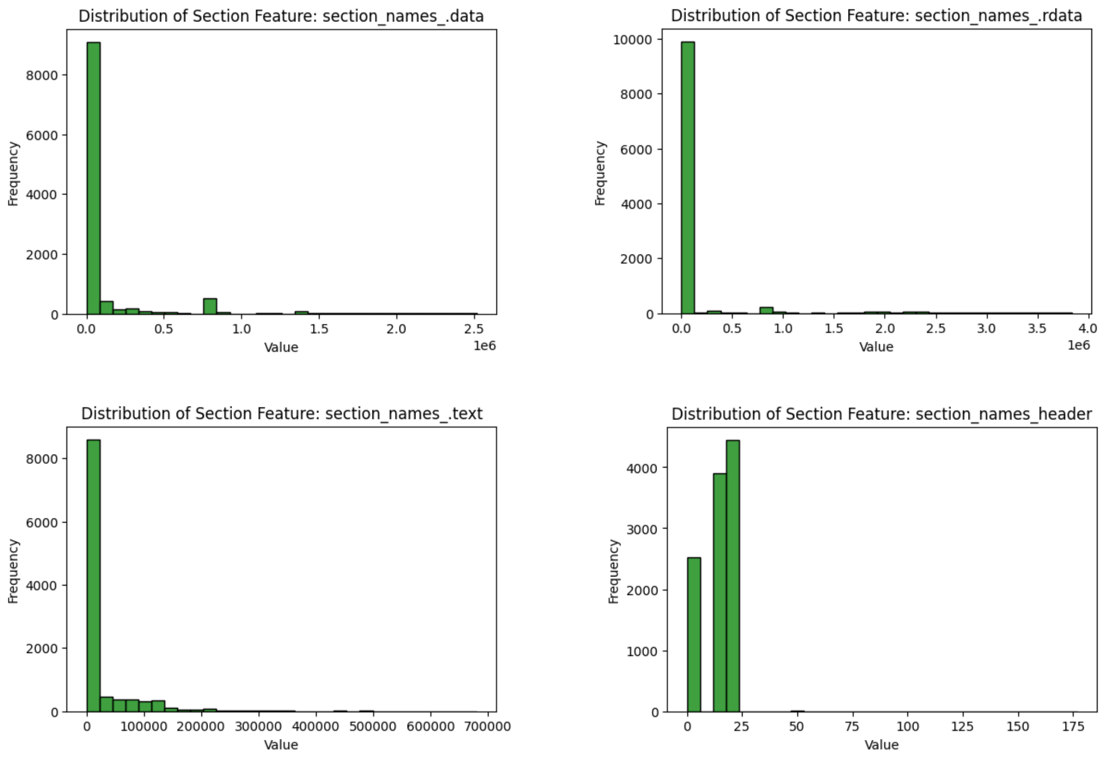


Figure AD: Distribution of Section Feature

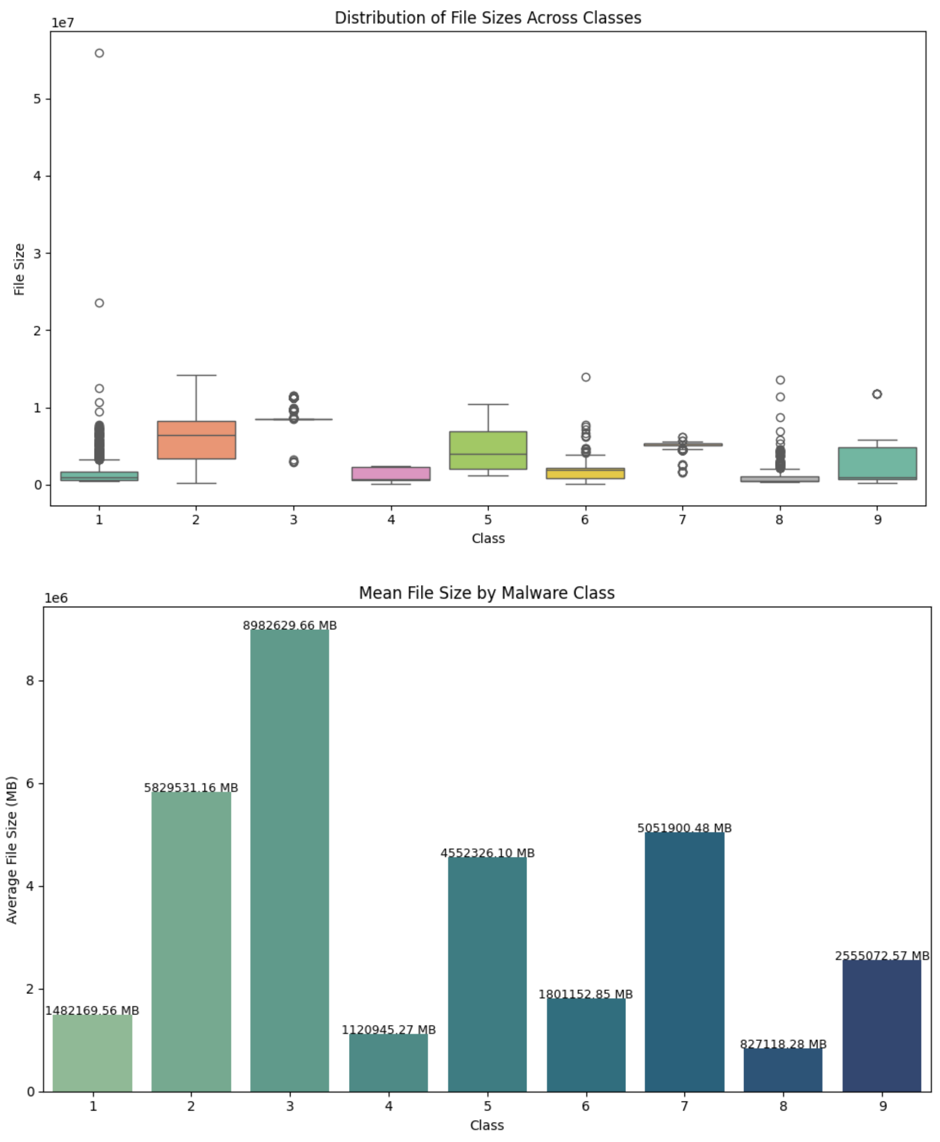


Figure AE: Box Plot File size and Mean Bar Chart

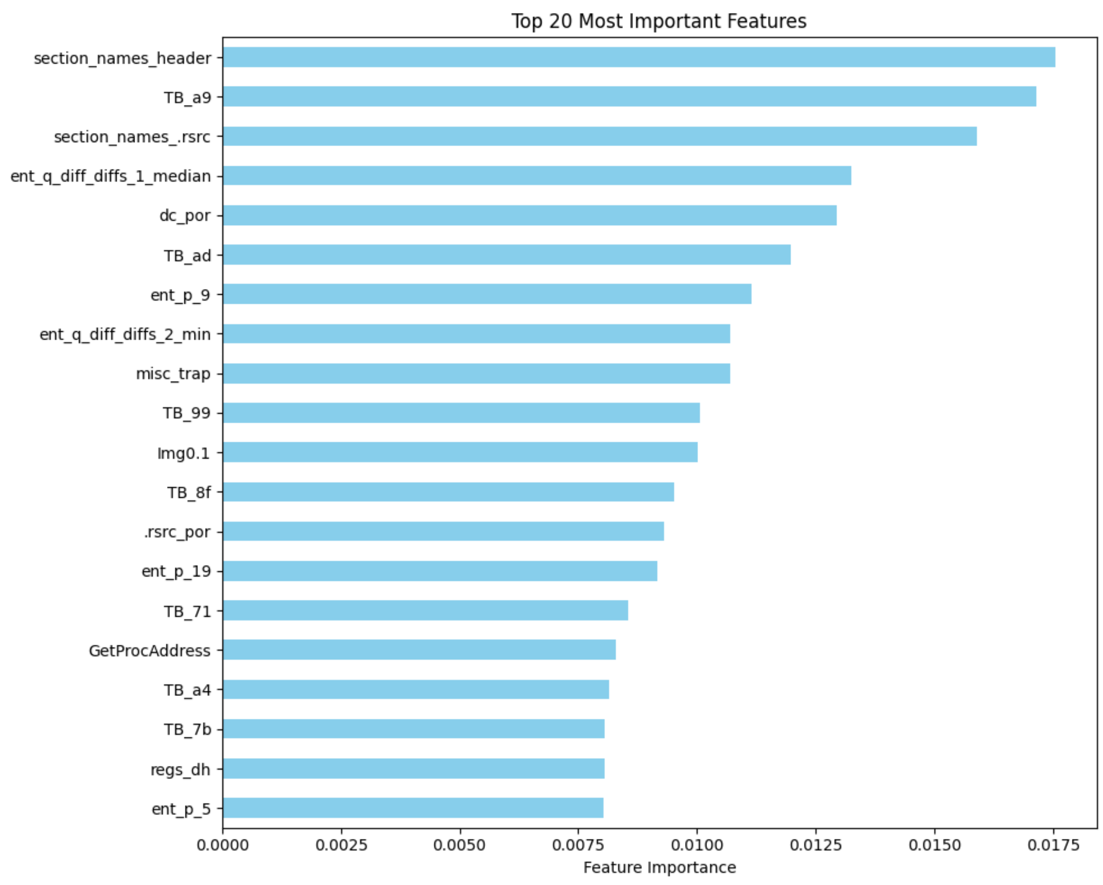
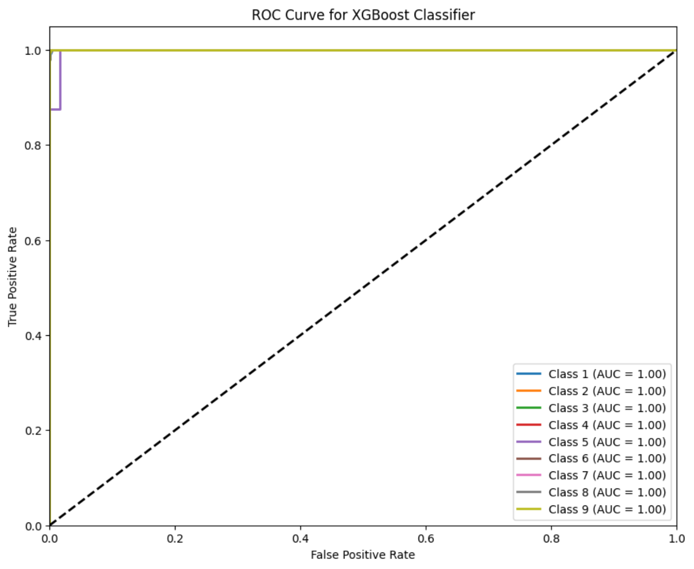
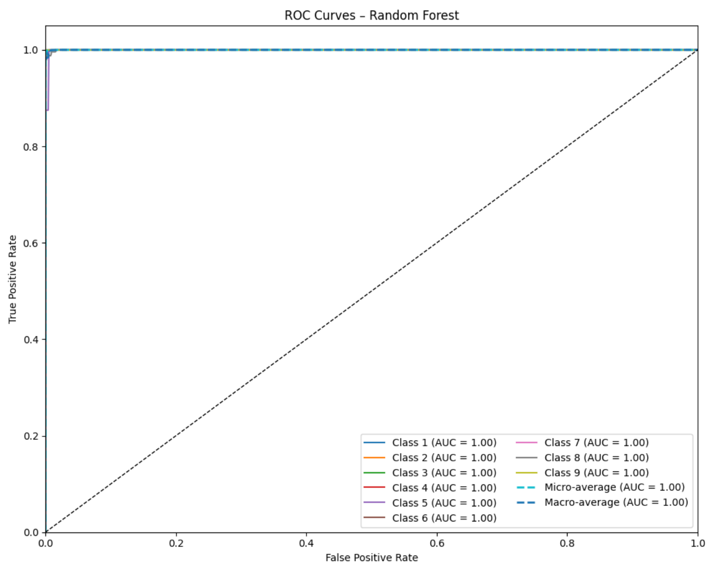


Figure AF: Top 20 Most Importance Features

APPENDIX B





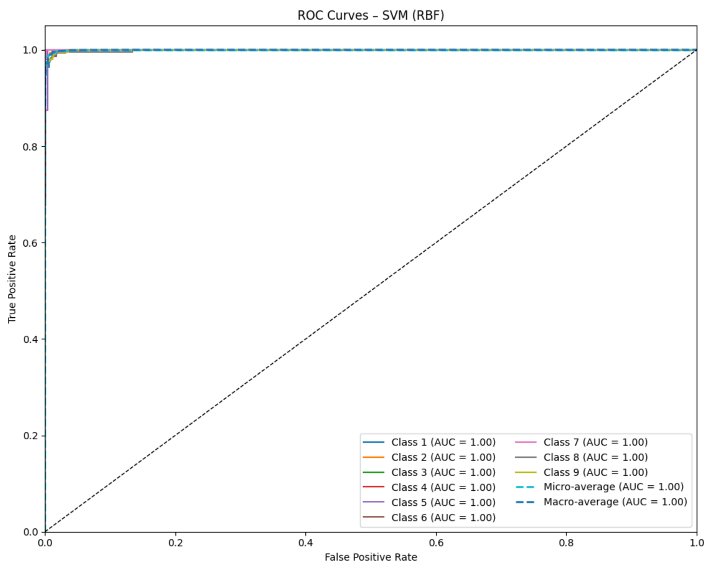


Figure BA, BB, and BC: ROC Curves for XGBoost, Random Forest and SVM