Design and Development of a Human-Centered Explainable Malware Classification System Using XAI and LLMs

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Cyber Security
May 2026
Ghazanfar Ali

Supervisors:
Seppo Virtanen
Kaitai Liang

UNIVERSITY OF TURKU
Department of Computing

Ghazanfar Ali: Design and Development of a Human-Centered Explainable Malware Classification System Using XAI and LLMs

Master of Science (Tech) Thesis, 93 p., 5 app. p.
Cyber Security
May 2026

This thesis explores the interpretability challenges of AI-based cybersecurity systems. Artificial intelligence (AI) has significantly improved malware detection compared to traditional signature-based approaches. However, these AI-based systems often operate as "black boxes," as they do not provide a rationale for their outputs, making the results difficult to trust. Security professionals require clear reasoning to make informed decisions, while non-technical users need simple explanations to understand the outcomes. To address this gap, this research proposes a human-centered explainable AI (XAI) framework that combines a classification layer with traditional XAI techniques such as LIME and SHAP. In the final layer, a large language model (LLM) generates clear and interpretable explanations for human users.

For this research, a balanced subset of the EMBER-2018 dataset containing Windows Portable Executable (PE) files in JSONL format was used. In the data extraction phase, 618 interpretable static features were extracted. In the classification layer, six models were implemented, with XGBoost reaching the best performance, with 97.0% accuracy and an ROC-AUC score of 0.997. In the XAI layer, LIME and SHAP were applied, identifying the compilation timestamp and high entropy as among the most important features. The LLM-based explanation layer uses lightweight local models (llama3.2:3b and deepseek-r1:1.5b), which take the top XAI features and a structured knowledge base as input. The LLM then converts these technical features into clear, human-understandable explanations for security analysts, security managers, and end users. Since the entire system operates locally without reliance on external cloud services, it enhances data security and eliminates the cost associated with API usage.

Keywords: malware classification, explainable AI, machine learning, cybersecurity, large language models, LIME, SHAP, human-centered AI

Contents

List of Figures

List of Tables

1 Introduction

The growing complexity of malware attacks has made cybersecurity one of the most persistent challenges in the digital age. Traditional malware detection relies on static or signature-based techniques, which often fail against new or polymorphic malware. In contrast, using artificial intelligence (AI) techniques such as machine learning (ML) and deep learning (DL) significantly improves malware detection by learning complex patterns in data. However, many AI models do not provide insight into the decision-making process and are often referred to as "black box" models. Therefore, explainable AI (XAI) has become an important area of research, aiming to make AI models' decision-making processes more transparent and interpretable. In the field of cybersecurity, trust in the models is fundamental, which is why the integration of XAI into malware classification could help security professionals trust the system when it categorizes a file or process as malware or benign. Recent advances in Generative AI and Large Language Models (LLMs) provide an opportunity to further enhance XAI by converting technical, low-level explanations into human-friendly, audience-specific descriptions (analysts, managers). This thesis proposes a human-centered explainability framework that integrates XAI techniques with Large Language Models to bridge the gap between technical explanations and practical understanding.

1.1 Research Problem and Objectives

In this thesis, the main research problem is identified as the gap between technical explanations generated by XAI methods and the level of understanding that human stakeholders require in cybersecurity. Although high accuracy is achieved by ML and DL models in malware classification, their black-box nature is considered a major limitation. Because of this, deployment in high-stakes cybersecurity environments is restricted. Currently, different XAI techniques, such as SHAP, LIME, and Grad-CAM, are used to highlight important features. However, the explanations that are produced are often difficult to understand by non-technical users. It is observed that security professionals require explanations that are clear and practical so that better decisions can be supported. In this research, a combination of LLMs and XAI techniques will be explored. Through this approach, explanations that are clearer and more understandable will be generated for malware classification. As a result, greater trust and improved usability are expected to be achieved in real-world cybersecurity environments.

The following objectives are addressed in this research:

1. To evaluate how well the ML and DL models perform malware classification.
2. To enhance the interpretability of malware classification models by integrating XAI techniques.
3. To investigate how LLMs can transform raw XAI outputs into human-centered explanations for different stakeholders.
4. To evaluate the effectiveness of combining XAI with LLMs in improving trust, usability, and decision-making.

The following research questions are examined in this study:

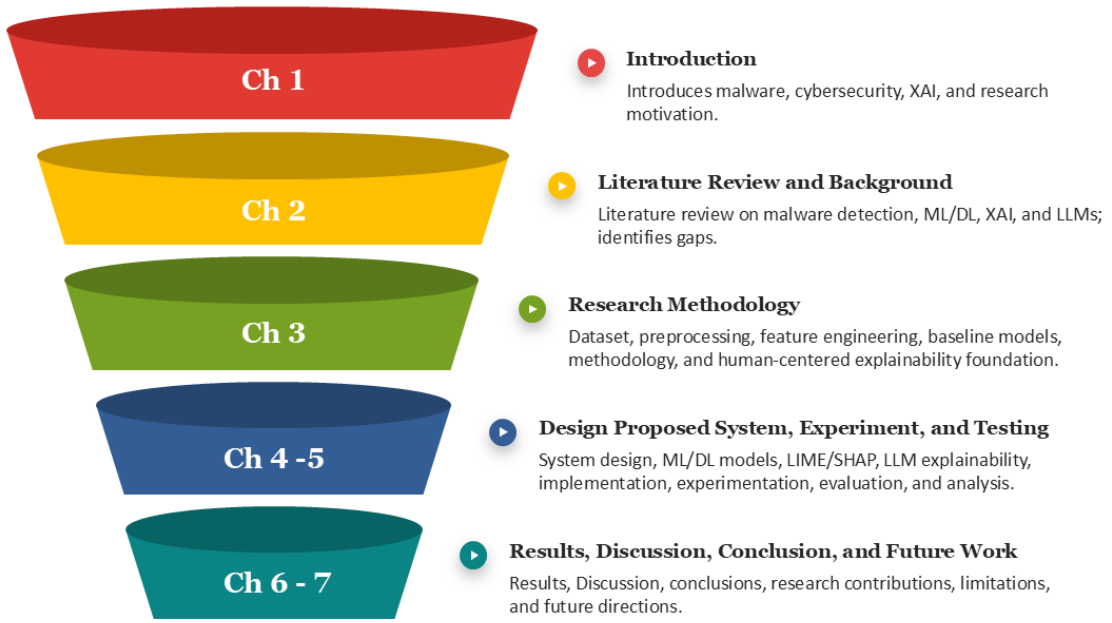

Figure 1.1: Thesis Focus Funnel (adapted from [1])

- RQ1: Which XAI technique provides the most interpretable and useful explanations for malware classification?
- RQ2: How can LLMs enhance the interpretability of XAI outputs, making them useful for security analysts and understandable for managers?
- RQ3: Does the integration of XAI and LLMs improve trust and usability in AI-driven malware detection?

1.2 Structure of Thesis

This thesis is structured into seven main chapters, each of which discusses a specific stage of the research process and contributes to the fulfillment of the thesis. The brief overview of each chapter is presented below. The focus-enhancing structure model, illustrated in Figure 1.1, inspires the thesis structure proposed in [1].

Chapter 1 introduces the background and motivation behind the research. The introduction discusses the growing challenges of malware detection and the need for interpretability in AI-based cybersecurity systems. The research problem, objectives, and questions are presented, followed by this section, which describes the thesis structure and declares the use of AI in the research process.

Chapter 2 provides a thorough review of existing literature relevant to this research. It discusses the role of Artificial Intelligence in cybersecurity, focusing on malware detection and classification techniques using machine learning (ML) and deep learning (DL). The concept of Explainable AI (XAI) is discussed in detail, including local and global explanation techniques such as LIME and SHAP. In addition, it covers the integration of Generative AI and Large Language Models (LLMs) in cybersecurity and explainability. Finally, it identifies the research gap that motivates this study.

Chapter 3 describes the overall research design and methodology used in this study. It introduces the EMBER dataset and explains the data preprocessing and feature engineering steps. It defines the baseline ML and DL models and the process of integrating XAI methods into malware classification. It also discusses the limitations of the existing system, leading to the proposed solution that includes an LLM-based human-centered explanation mechanism.

Chapter 4 presents the detailed architecture and design of the proposed human-centered explainability framework. It explains how the system integrates three main layers: the ML/DL classification layer, the XAI interpretation layer (LIME and SHAP), and the LLM explainability layer. It also describes the data flow between components, the prompt engineering process for the LLM, and the generalization potential to other platforms (such as Android or IoT malware). This chapter defines

the blueprint for implementation and experimentation.

Chapter 5 focuses on the experimental setup and testing procedures. It will explain how various models, such as random Forest, XGBoost, and deep learning networks, are trained on the dataset. The integration of XAI modules (LIME and SHAP) and the LLM explainability module is then described in detail. The evaluation framework will cover both performance metrics (such as F1 Score and ROC-AUC) and explainability metrics (such as fidelity and runtime). In addition, human-centered evaluation metrics, including trust and usefulness, will also be introduced to assess the quality of explanations.

Chapter 6 presents and analyzes the experimental results, evaluates the model performance, and discusses the findings related to XAI and LLM-based explanations. It will also review the LLM-generated explanations and conclude by explaining what these findings mean in real-world cybersecurity applications.

The final chapter, Chapter 7, summarizes the main contributions and findings of this research. It will discuss the impact of the proposed framework on improving trust and interpretability in AI-based malware detection systems. In addition, it will also discuss the limitations of the current study and suggest several directions for future research. For example, extending the framework for new types of data or for multiple systems, and exploring advanced ways of integrating LLM.

The references section lists all academic and technical sources cited throughout the thesis. The appendices will contain supplementary materials such as additional results, code snippets, and configurations that support the main body of the research.

1.3 Declaration of AI Usage

Artificial intelligence tools like ChatGPT are used to refine and rephrase my sentences, removing grammar and punctuation errors. These tools are used to summarize long research papers or extract key points from them, which are then reviewed and rewritten in my own words. All AI-generated outputs are used solely for reference and are carefully checked against their sources. In addition, AI is used to generate the LaTeX format for the algorithms, finding bugs in the code, and improving code structure.

2 Related Work and Background

2.1 Purpose and Goals

This chapter critically analyzes existing approaches to malware detection and explainability, highlighting their limitations and identifying the gaps that motivate the proposed framework. The existing literature and related work are reviewed to examine research on malware analysis, the application of artificial intelligence (AI) in cybersecurity, and various approaches to explainable AI (XAI) [2]. This chapter discusses key concepts of AI in cybersecurity, with a focus on malware analysis and the research gaps in this area.

The literature review identifies improvements in existing research, current limitations, and open research gaps that together motivate the design of a new human-centered explanation framework. This chapter also explains the key concepts of malware analysis, using popular XAI techniques like LIME and SHAP, and LLMs (Large Language Models) to make AI decisions easier for humans to understand.

2.2 Historical Context of AI and Machine Learning

Artificial Intelligence (AI) began with rule-based expert systems in which human experts manually encoded decision-making rules. This approach was limited by the difficulty of capturing all possible scenarios and the inability to adapt to new patterns.

With the advancement of computing power in the late 20th century, Machine Learning (ML) emerged as a data-driven paradigm. Instead of relying on explicit programming rules, ML algorithms learn patterns directly from data using statistical models. Early machine learning techniques, such as Support Vector Machines (SVM), Decision Trees (DT), and Random Forests (RF), demonstrated strong performance in classification tasks. In the field of cybersecurity, machine learning has been applied to malware detection—particularly to analyze Windows Portable Executable (PE) files—since at least 2001 [3].

As computational resources continued to grow and larger datasets became available, Deep Learning (DL) emerged as an extension of machine learning. Deep learning employs multi-layered neural networks that can automatically discover hierarchical feature representations from raw data without manual feature engineering. Models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown remarkable success in malware detection, often outperforming traditional machine learning approaches in handling complex, high-dimensional, and non-linear classification problems [2].

2.3 Artificial Intelligence in Cybersecurity

As Artificial Intelligence and its subfields—Machine Learning and Deep Learning—have advanced, these technologies have emerged as critical components of modern cybersecurity [4]. Advancements in AI have revolutionized the digital landscape by enabling the development of more effective and efficient algorithms that can mimic human intelligence [2]. In the field of cybersecurity, particularly in malware analysis, AI techniques can overcome the limitations of traditional detection methods by automatically learning complex file attributes that distinguish malicious software from benign applications.

Although AI has introduced new attack vectors and techniques, these same tech-

nologies have also improved security measures by providing novel approaches to counter cyber threats [5]. AI applications in cybersecurity include intrusion detection, spam filtering, and malware detection and classification [6]. The increasing sophistication of cyber threats—including AI-powered attacks—underscores the critical importance of employing advanced AI techniques for defense. The economic impact of cybercrime is substantial: global losses were projected to reach $6 trillion annually by 2021, highlighting the urgent need for robust AI-driven security solutions [7].

The advantages and disadvantages of using artificial intelligence in cybersecurity, especially in malware classification, are summarized in Table 2.1 [2, 5, 8, 9].

Table 2.1: Comparative overview of the benefits and drawbacks of using Artificial Intelligence in cybersecurity, specifically in malware classification

Aspect	**Benefit**	**Drawback**
Detection and Analysis	Provides strong and accurate detection of malware and unusual activity	Many models work as "black boxes," which limits user confidence
Operational Efficiency	Helps to minimize workload on analysts, reduces human mistakes, and maintains consistent 24/7 operation	Can be demanding in terms of resources and slow for real-time tasks, especially during dynamic analysis
Adaptability	Can automatically adjust to new and evolving threats, including polymorphic and zero-day malware	Vulnerable to adversarial inputs designed to trick or bypass detection
Data and Reliability	Capable of identifying fine-grained patterns that humans may overlook	Needs large amounts of high-quality labeled data to achieve reliable performance
Ethics and Compliance	Can support better fairness and more ethical decision-making	May introduce biases (social, linguistic, representational) that lead to unreliable outcomes

2.4 Portable Executable (PE) File Format

The portable executable (PE) file format is the standard file structure of the Microsoft Windows operating system's files for executables, object code, and dynamic link libraries (DLLs). This file is based on the Common Object File Format (COFF) and contains the information that is necessary to run the program on the Windows operating system. Files that use this format have extensions such as .exe, .dll, .obj, and .sys [3, 10, 11]. The structure of a PE file is divided into two primary components: headers and sections, as shown in Figure 2.1.

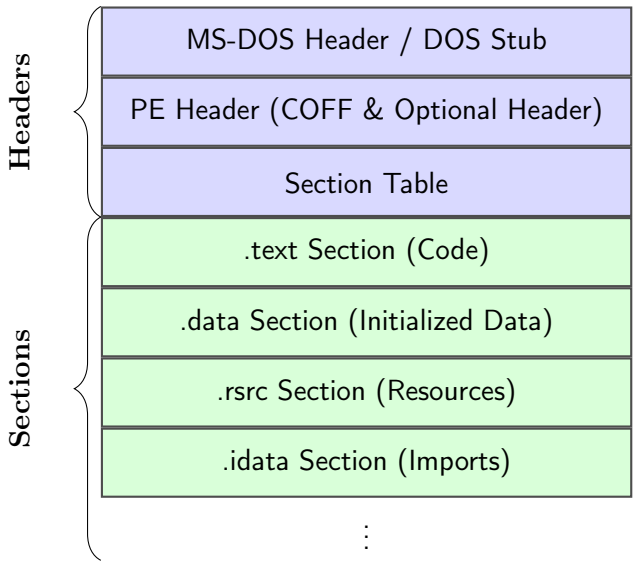

Figure 2.1: Structure of a Portable Executable (PE) File

2.4.1 Headers

The header component consists of the following elements [3, 11].

- **MS-DOS Header**: A legacy structure located at the beginning of the file to ensure backward compatibility. It includes a small DOS stub program that displays a message if the file is executed in a DOS environment.

- **PE Header (COFF Header)**: Marks the beginning of PE-specific information. It includes metadata such as the machine type (e.g., x86 or x64), number of sections, timestamp, and a pointer to the symbol table.
- **Optional Header**: Despite its name, this header is essential for executable files. It provides critical information for the OS loader, including the entry point address, required operating system version, and sizes of code and initialized data. It also contains the data directory, which points to structures such as imports, exports, resources, and security certificates.
- **Section Table**: Defines the layout of the file by specifying the name, offset, size, and characteristics of each section.

2.4.2 Sections

The program content is organized into multiple sections that are loaded into memory during execution [3, 10].

- **.text (Code Section)**: Contains executable instructions and the core logic of the program.
- **.data / .rdata / .bss (Data Sections)**: Store program data, including initialized variables (.data), read-only constants (.rdata), and uninitialized data (.bss).
- **.rsrc (Resource Section)**: Holds non-executable resources such as icons, images, menus, and strings.
- **.idata (Import Section)**: Contains information about external libraries (DLLs) and functions required for execution.

PE files are the primary target for malware because they are widely used in Windows operating systems. Malware authors often modify the PE file structure to

hide malicious behavior using various techniques such as packing (file compression), obfuscation, or modifying checksums to avoid detection by security software. For static analysis, features are extracted from PE file headers and sections, including byte entropy, header details, and API call information. These extracted features are used to distinguish between benign and malicious files [11].

2.5 Malware Detection and Classification Techniques

Malware analysis, such as detection and classification, has become an increasingly important part of cybersecurity due to the advancement of cyber threats [4]. Malware detection can be categorized into two main types: file-based classification and online-based approaches. In file classification, we usually analyze the code of the file to find out whether it is malicious or not. These techniques are divided into three main categories: static analysis, dynamic analysis, and hybrid analysis [9].

2.5.1 Static Analysis

Static analysis is the fundamental method used to analyze the file by inspecting its code without running the file. In this technique, the analyst will investigate different static features, such as PE headers, import and export libraries or services, entropy, printable/readable strings, and n-grams, which are extracted from the file. This method helps get an initial idea about the file that might be malicious [2]. However, static features are not robust, and detection can be bypassed by packed or obfuscated malware. Static analysis is also limited in that it cannot detect the malware that is already running within the system, and cannot identify new variants of the malware [9].

2.5.2 Dynamic Analysis

Dynamic Analysis is a technique in which the file is executed within a secure virtual environment, such as a sandbox or emulator, so that its behavior can be monitored and analyzed in real time [2, 9]. This method is important because some malware exhibits malicious actions only when executed, making it particularly useful for detecting zero-day malware threats. During the dynamic analysis process, various runtime artifacts are collected, including API calls, call graphs, memory modifications, registry updates, hardware information, and network activity [2]. Dynamic analysis is also helpful in detecting malware variants that use evasive techniques like polymorphism or metamorphism, which change their static structure to avoid detection. However, this method is time-consuming and computationally expensive, demanding substantial resources [4]. Additionally, some malware is specifically designed to detect and evade analysis environments, remaining dormant to avoid detection.

2.5.3 Hybrid Analysis

In hybrid analysis, the strengths of both static and dynamic methods are combined to ensure that the detection system is complete and effective in identifying malware [9]. Using static features (such as PE headers) and dynamic features (such as API calls) would give strong and more reliable results. However, studies show that there is not much research in using hybrid analysis that could be more explainable and easier to understand [9]. While hybrid analysis improves detection performance, it also increases explainability complexity, as explanations must account for heterogeneous feature sources.

2.6 Machine Learning and Deep Learning for Malware Classification

Machine learning and deep learning models have become increasingly effective solutions for malware detection. These models automatically exploit the relationship within file attributes for accurate classification.

2.6.1 Machine Learning Models

Traditional machine learning models are widely used in cybersecurity. For example, a Decision Tree (DT) is naturally explainable because it creates clear rules on how the decisions are made [12]. Ensemble methods, such as Random Forest (RF), combine the power of multiple decision tree models to improve the accuracy and reduce overfitting. However, as the model becomes more complex, its power of explainability reduces [9].

2.6.2 Deep Learning Models

Deep learning models, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have demonstrated improved performance in recent studies [4]. Unlike traditional machine learning approaches, these models automatically extract features by identifying patterns directly from raw data, reducing or eliminating the need for manual feature engineering [4].

Models like RNNs, specifically Long Short-Term Memory (LSTM) based models, are used to process sequential data and keep track of time-based patterns or dependencies. For example, these types of models are helpful to analyze the API call sequences to identify the behavioral patterns of malware while executing [8]. Multi-Layer Perceptron (MLP) has also been used for the same purpose for malware detection, particularly for API calls, which are the main communication medium in

dynamic malware analysis [8].

Even though these AI (ML/DL) models provide high accuracy and performance, they are unable to explain their decisions; therefore, their limited interpretability restricts their adoption in critical cybersecurity applications. Due to their internal complexity, these models often lack transparency, interpretability, and reliability [2].

2.7 Explainable AI: Concepts and Need in Cybersecurity

Explainable AI (XAI) is the area of AI that includes various methods or techniques that help to make AI models' decisions more transparent and easier for people to understand. In simple terms, its main goals are to help humans understand and see how an AI system reaches a decision [13].

2.7.1 The Need for Explainable AI in Cybersecurity

The main reason for using XAI in cybersecurity is to address the lack of transparency and interpretability in complex, high-performance AI models used for malware analysis and related cybersecurity tasks. XAI makes AI systems more understandable and helps build trust in their outputs. Since cybersecurity is a critical domain, security professionals need to understand how decisions are made; otherwise, it becomes difficult to trust automated systems [7, 13].

Explanations help identify model weaknesses, detect biases in data, validate predictions, and guide model improvement. XAI also provides insights into important features used to distinguish between malicious and benign files, thereby improving model interpretability [10].

In cybersecurity applications, a lack of interpretability can make it difficult to trust and justify AI-based decisions. For legal and ethical considerations, the ability

to question automated decision-making processes makes clear explanations essential [6].

2.7.2 The Trade-off

There is a major challenge in XAI development, which is a trade-off between model prediction accuracy and its explainability. Some models that are naturally interpretable, such as Decision Trees, are good in transparency but often do not perform well on complex cybersecurity tasks. On the other hand, advanced models such as Deep Neural Networks provide high accuracy in model prediction but fail to provide interpretability. So, there is a need for more research in this area to develop the models that could provide good explainability while maintaining the good accuracy (performance) of the model [6].

2.8 Various XAI Techniques

The purpose of using Explainable Artificial Intelligence (XAI) techniques is to provide insights into how machine learning models make their decisions. These techniques can be categorized based on their scope, such as local or global, and approach, including model-specific or model-agnostic. Since modern AI models, which build on neural network architecture, are opaque and more difficult to analyze, the need for interpretability has increased significantly. Understanding the decision process of the model not only builds trust but also helps in debugging, regulatory, compliance, and ethical AI deployment [14].

2.8.1 Local vs Global Explanation

XAI techniques can be classified as local or global. Local explanations are meant to interpret the decision of the single instance or prediction made by the model

that provides insight into why a particular sample received a specific label. For example, LIME (Local Interpretable Model-Agnostic Explanations) generates a local surrogate model to approximate the complex decision boundary around the chosen instance [15]. On the other hand, global explanations are meant to describe the overall behavior and the feature importance of the model across the entire dataset. This explanation helps to understand which features generally influence the predictions of the model and how the parameters of the model behave overall [16].

There should be a balance between these two perspectives and use cases. For example, global interpretability ensures that there should be transparency in the development of the model, while local interpretability helps practitioners and analysts to understand the case-specific decisions, especially in the most critical fields such as cybersecurity and healthcare [14, 16].

2.8.2 LIME, SHAP, and Other Methods

LIME (Local Interpretable Model-Agnostic Explanations) approximates a complex model locally with an interpretable model, such as linear regression or a decision tree, by perturbing the input data and then observing how the output will change [15]. The strength of this model is in its simplicity and model-agnostic nature, which could explain any classifier or regressor. However, there is also a limitation of this model, which is instability, meaning slight variations in the input could lead to different explanations [15, 16].

The SHAP (SHapley Additive exPlanations) model built on cooperative game theory, assigns an importance value (Shapley value) to each feature by considering all feature combinations. SHAP combines the best parts of other interpretability methods and satisfies the required properties, such as consistency and local accuracy. Unlike LIME, SHAP provides consistent and fair attributions of feature contributions, but it could be computationally expensive for large models [14, 16].

Other explanation methods, such as CAM (Class Activation Maps) and Grad-CAM, are particularly useful for CNNs (Convolutional neural networks), which highlight the region of the image that strongly influences the model predictions and enable visual interpretability. While LIME and SHAP are suitable for tabular and text-based data, the CAM technique is suitable for image-based tasks, which makes complementary approaches in XAI research [16].

Hence, the selection of an XAI technique is dependent on the problem type, data, and the requirements for interpretability (local vs global). For malware classification, LIME and SHAP are useful because they can provide insights into which features drive the classification decisions, which helps to build trust in the decision and understand the AI security systems.

2.9 Generative AI and LLM in Cybersecurity

2.9.1 LLMs for Cybersecurity Applications

Large Language Models (LLMs) such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) variants that are built using advanced deep learning methods based on transformer architecture and self-attention mechanisms. The transformer architecture was first introduced in the paper "Attention is All You Need," and it uses a stack of self-attention with fully connected layers to process input data efficiently [4, 17].

Because LLMs are trained on large data sets, they offer new opportunities to improve malware analysis. These models can process unstructured data like raw code, log files, and error messages. In cybersecurity, LLMs can be applied to tasks such as threat detection, vulnerability detection, spam filtering, and malware classification. These models can also help to identify the malicious functions or summarize the purpose of malware code from the decompiled code [4].

LLMs take the code as text input and can detect malware-related patterns, such as suspicious function calls or unusual network requests, by comparing them with the examples of known malicious patterns that they learned during their training process [4].

2.9.2 LLMs for Explainability Enhancement

LLMs can play an important role in developing human-centered explanations by closing the gap between highly technical XAI outputs and human-understandable explanations. LLMs take the input generated by XAI techniques (like LIME and SHAP values) and translate them into clear natural language explanations, which will be helpful for different audiences, such as security analysts or managers, to understand how AI makes the decision [13].

LLMs' outputs are not deterministic, so prompt engineering plays a critical role in guiding the LLM to generate well-structured outputs in the desired format. This ensures that explanations should be clear and relevant by converting the features (such as API calls or permissions) into straightforward security insights [18].

2.10 Identified Research Gap

The main reason to use explainable AI (XAI) is the lack of transparency and human understanding in the available AI (ML and DL) models, which are quite powerful in malware detection but do not provide an explanation of their decisions. Like many other fields, AI has significantly improved cybersecurity. However, the internal complexity of these models makes it difficult to build trust and confidence, especially in critical areas such as malware detection and digital forensics, where even small mistakes can have serious consequences [2].

Despite significant research, high accuracy in machine learning and deep learning

models, and the existence of XAI techniques like LIME and SHAP, there are still critical challenges that need to be addressed. The following research gaps motivate this thesis:

- **Gap 1: Comparative Effectiveness of XAI Techniques for Malware Classification:** Although several post-hoc XAI methods, including LIME, SHAP, and Grad-CAM, have been used in malware analysis, only limited work systematically compares these techniques. In particular, there is a lack of studies that examine which method produces the most clear and useful explanations for malware classification. Each XAI approach has its own strengths and weaknesses, especially when handling the high-dimensional features and complex binary structures found in PE malware files. [2].

  Current XAI methods often struggle to explain the complex and evasive nature of malware, such as polymorphism and obfuscation, which are deliberately used to hide malicious behavior. In addition, there is no common standard for measuring explainability, and clear, objective evaluation metrics are still missing. As a result, many studies depend on manual and subjective evaluations, showing a strong need for more reliable and quantitative ways to assess how trustworthy and useful these explanations really are [2, 6].

- **Gap 2: Bridging the Semantic Gap Between XAI Outputs and Human Understanding:** One of the major challenges is the gap between the technical outputs generated by existing XAI methods and the needs of various human stakeholders in cybersecurity. Post-hoc techniques such as LIME and SHAP generate numerical scores that show how important certain features are for a decision. However, these results are often presented as raw feature names or complex visualizations, making them too abstract or technical for practical use by many stakeholders, including security analysts, managers, and non-technical users [2, 19].

- **Gap 3: Human-Centric Validation of Trust and Usability in XAI-LLM Systems:** Most previous research has focused mainly on improving the prediction performance of models, measured by metrics such as accuracy or F1 score. However, successfully deploying AI systems in critical security scenarios depends heavily on human factors such as trust, reliability, and usability. There is a significant gap in research that carefully examines how LLM-enhanced XAI explanations affect these human-centric factors for security professionals in real-world environments [8].

By addressing these three interconnected research gaps, this thesis aims to develop a comprehensive human-centered explainability framework that not only improves the technical quality of explanations but also ensures that these explanations are practical, understandable, and trustworthy for diverse stakeholders in cybersecurity operations.

3 Research Methodology

This chapter outlines the methodology used to achieve the research objectives, focusing on the design and evaluation of a human-centered, explainable malware classification system. It includes data set selection and preparation, feature engineering, and selection of the baseline classification model. It also includes how explainable AI (XAI) techniques were used and how Large Language Models (LLMs) were used to enhance interpretability and explainability.

3.1 Dataset and Data Preprocessing

The primary dataset used in this study is the EMBER dataset (Endgame Malware Benchmark for Research), available on GitHub [3]. The version used is EMBER-2018, a widely adopted benchmark dataset for malware analysis research. The EMBER dataset was created to address the gap in the availability of large open and general-purpose datasets to train machine learning models in malware classification [3, 20].

The EMBER-2018 dataset includes features that were extracted from approximately 1.1 million Windows Portable Executables (PE) binary files. It includes a training set of 900,000 samples (300,000 benign, 300,000 malicious, and 300,000 unlabeled) and a separate test set of 200,000 files (100,000 benign and 100,000 malicious). The malicious files in the dataset were confirmed through reports from more than 40 antivirus vendors, while the benign files showed no malicious detections

at the time of collection [3]. For this research, a balanced subset of 2,000 samples (1,000 malicious and 1,000 benign) is extracted from the EMBER-2018 training set to ensure computational feasibility while maintaining class balance. This subset is then split into training (70%), validation (15%), and test (15%) sets using stratified sampling to preserve class distribution across all splits.

3.1.1 Data Preprocessing

The raw EMBER-2018 dataset is available in JSON lines format. The EMBER dataset provides raw features such as lists of imported functions or strings that must be transformed into fixed-length numerical arrays. Two approaches exist: first, using the standard EMBER methodology with the provided embedding functions, which results in a 2,381-dimensional feature space, but the features lack semantic meaning. Second, extracting or creating custom features through custom functions, where each feature has semantic meaning and interpretability [3, 20].

For this research, the second approach is adopted to ensure explainability. Approximately 618 interpretable features are extracted from each PE file (available in JSONL format), covering header metadata, section statistics, import/export information, and string patterns.

Once features are extracted, the dataset is cleaned to handle any missing values, infinite values, or anomalies. After cleaning, features are normalized using the StandardScaler, which transforms each feature to have zero mean and unit variance. This scaling is essential for models like Logistic Regression and neural networks, though tree-based models (Random Forest, XGBoost) are scale-invariant [3, 20]. The scaler is fitted on the training set and applied to the validation and test sets to prevent data leakage.

3.2 Feature Engineering and Representation

For effective feature engineering, there should be an understanding of the provided dataset and its structure. The EMBER-2018 dataset contains a rich set of features that was created for static malware detection. These features are organized into multiple groups, which can be categorized into parsed features that are derived directly from the PE file structure and format-agnostic features that are derived from statistical analysis of the raw bytes without relying on file format. Together, these feature groups provide detailed information about the PE file behavior and structure [3].

The primary focus of feature engineering is to handle the high dimensionality of data to ensure efficient learning, to convert variable-length data such as imports, sections, and strings into consistent numerical input, and extract features that should have semantic meaning that could be inputs to machine learning models for training which also enables interpretability and explainability of model decisions which is the key requirement of this research.

3.2.1 Parsed Features

These features are taken from the PE file headers, which include COFF and Optional headers, as well as section details such as name, size, entropy, and virtual size. They describe the file's static structure and metadata. This information collectively describes how the executable is constructed and intended to run [3].

Header-level features include compilation timestamps, machine architectures, linker version, operating system version, and memory allocation information such as header size and code size. This information could be helpful in malware detection, as malicious files often contain unusual header information that deviates from normal information [3].

Section-based features contain numerous sections that are present in the exe-

cutable, which contain their sizes, virtual sizes, and entropy values. The entropy value is widely used for detecting packing or encryption, which is commonly used by malware to evade static analysis. Statistical measures such as mean, maximum, and standard deviation of section entropy are included to summarize the information in fixed-length form [3].

In addition, import and export table features are derived to capture the interaction of the executable (PE file) with system libraries. These features include the number of imported dynamic link libraries (DLLs), the total number of imported functions, and the presence of commonly exploited system libraries such as KERNEL32.dll, ADVAPI32.dll, and WS2_32.dll. These kinds of imports often show the capabilities related to process manipulation, accessing the registry, networking, or connection persistence [3]. Table 3.1 shows the summary of parsed features.

3.2.2 Format-Agnostic Features

These features are independent of file format and include a normalized 256-bin histogram of raw byte values, a byte-entropy histogram that combines entropy and byte values into 16×16 bins, and statistics related to printable strings, such as the number of strings, their average length, and the presence of paths or URLs persistence [3]. Table 3.2 shows the summary of format-agnostic features.

3.3 Baseline Models

A diverse set of baseline models is selected that covers traditional machine learning and deep learning models. These models are chosen as a solid reference for performance evaluation, providing comprehensive classification capabilities for explainability analysis before the integration of XAI techniques. In total, six models are employed in the experimental setup.

Table 3.1: Summary of Parsed Features

Feature Category	**Feature Example**	**Description**
File Metadata	Physical and virtual file size of the executable	Packed malware often shows abnormal size ratios
COFF Header	Compilation timestamp, machine type	Information about target architecture and build time, malware often uses fake or inconsistent timestamps
Optional Header	Subsystem, linker version, OS version	Unusual build settings or execution details can suggest that a binary may be malicious
Code Properties	Size of code, header size	Unusual memory layout or a strange balance between code and file size can be a sign of obfuscation
Section Count	Number of sections	Total sections in the PE file, Malware may use fewer or irregular sections
Section Entropy	Min, Max, Mean, and Standard deviation	Randomness of section content, High entropy means packing or encryption
High-Entropy Sections	Count of sections with entropy > 7	Encrypted or compressed areas often point to obfuscation and are a strong sign of malicious behavior
Import Table	Number of imported DLLs and functions	Shows external API usage, malware often uses suspicious APIs
Suspicious DLL Imports	`KERNEL32`, `ADVAPI32`, `USER32`, `WS2_32`	Number of imported functions can reveal signs of system manipulation or network-related activity
Export Table	Number of exported functions	Exported symbols from executable
Data Directories	Import, export, TLS, relocation tables	Presence and size of PE data directories

3.3.1 Machine Learning Models

Traditional machine learning classifiers are used to provide an interpretable and well-understood baseline. The following models are selected based on their complementary strengths and widespread use in malware classification research [3, 20].

Table 3.2: Summary of Format-Agnostic Features

Feature Category	**Feature Example**	**Description**
Byte Histogram	Byte frequencies (0–255)	256-dimensional normalized vector, Detects statistical patterns caused by packing
Byte Entropy Histogram	Entropy-byte distribution	256-dimensional normalized vector, Highlights encrypted or compressed regions
String Count	Total printable strings	Scalar, Packed malware often contains fewer strings
Average String Length	Mean string size	Scalar, Obfuscation affects string structure
URL Strings	Presence and count of URLs	Binary flag/count, indicates command-and-control communication
Embedded Executables	Multiple "MZ" headers	Binary flag, indicates droppers or multi-stage malware
String Density	Strings per byte	Ratio, Low density suggests encryption or packing

Random Forest (RF): A Random Forest classifier is configured with 200 trees (n_estimators=200), a maximum depth of 20, minimum samples split of 5, and minimum samples leaf of 2. This model is selected because of its ensemble robustness, its ability to provide feature importance insights, and its resilience to overfitting [12].

XGBoost: The XGBoost model is selected as a representation of the gradient boosting method that shows strong performance in malware classification tasks. This model is configured with 200 estimators, a maximum depth of 10, and a learning rate of 0.1. Its ability to capture non-linear relationships and achieve state-of-the-art results makes it a reliable benchmark for tree-based ensemble methods.

Logistic Regression (LR): A Logistic Regression classifier with L2 regularization (C=1.0) is applied after feature scaling using StandardScaler. This model serves as a linear baseline that offers transparency and ease of interpretation for comparison with complex models. While simpler than ensemble methods, it provides valuable insights into linear feature relationships [9].

3.3.2 Deep Learning Models

Multiple neural network architectures are employed to evaluate the effectiveness of deep learning on structured malware features. These models are implemented using TensorFlow/Keras and trained with early stopping and learning rate reduction callbacks to prevent overfitting [4].

Multi-Layer Perceptron (MLP): A deep MLP is implemented with four dense layers (512, 256, 128, and 2 neurons for binary classification). Each hidden layer includes batch normalization and dropout to improve generalization. The model uses ReLU activation for hidden layers and softmax for the output layer. This architecture represents a standard deep learning baseline for tabular data.

1D Convolutional Neural Network (1D-CNN): A one-dimensional CNN architecture is used with three convolutional blocks containing 64, 128, and 256 filters, respectively. Each block includes convolutional layers, batch normalization, Rectified Linear Unit (ReLU) activation, and max pooling. The architecture is followed by global max pooling and dense layers. This model is designed to capture local patterns and spatial relationships within the feature space [8].

Autoencoder: An autoencoder is employed using an unsupervised learning paradigm. The encoder compresses the 618-dimensional feature space into a lower-dimensional latent representation, while the decoder reconstructs features back to

the original dimensions. This model is trained solely on benign samples and used for anomaly detection, where malware is identified as samples that cannot be reconstructed accurately (high reconstruction error) [4].

3.4 Integration of XAI with Malware Classification

The methodology integrates the Explainable AI (XAI) techniques as a post-hoc component to improve the transparency and interpretability of the high-performing baseline models. Instead of modifying the internal architecture, explainability is applied after the training to better understand the decision [9].

Two widely used model-agnostic explainability methods are selected for this study, namely LIME (Local Interpretable Model-Agnostic Explanations) and SHAP (SHapley Additive exPlanations). These techniques are independent of the underlying model structure, as they generate explanations for predictions after model training without requiring access to the model's internal architecture [9].

3.4.1 Mechanism of Integration

LIME is used to produce the localized explanations by approximating the complex decision boundary of the trained model around a specific input instance. This is achieved by a simpler, interpretable surrogate model that explains the predictions of the individual malware classification [2].

On the other hand, SHAP is based on the concepts of cooperative game theory and assigns an importance score to each feature. These values are used to quantify how much each feature contributes to a specific prediction. SHAP is used to generate both local instance-based explanations and global insights into overall feature importance across the dataset [9].

3.4.2 Role of XAI in the pipeline

Once the models are trained and their performance is validated, XAI techniques are applied as an additional layer for explanation. This layer provides interpretable feature attribution scores and localized explanations that give a clear understanding of how and why some specific instances are classified as malware or benign [2].

3.5 Limitations in the Previous System

Although incorporating high-accuracy classification models with well-known XAI techniques could help to reduce the black-box nature of AI systems, these traditional XAI techniques still have some limitations. These limitations motivate the need for additional solutions that are proposed in this study.

- Machine learning and deep learning models that give high accuracy are often more complex by design. The complexity of the model makes its decision-making process difficult to understand, which is commonly described as a "black-box" problem. As a result, it is hard for security professionals to trust the systems, and they hesitate to rely on them [2].

- In many cases, models that perform better and give good prediction accuracy are also the hardest to interpret. This trade-off creates challenges in security-critical domains, where understanding the model's decision and trusting it is highly important to achieve high accuracy [6].

- XAI methods such as LIME and SHAP provide explanations in the form of numerical values, ranking of features, or complex visualizations. However, these explanations are often too technical and insufficient for human decision-makers, especially for non-technical stakeholders such as managers and leaders.

Due to this gap, the practical usefulness of XAI outputs is limited in real-world security environments [9, 14].

3.6 LLMs for Human-Centered Explainability

To overcome the semantic gap in the existing XAI outputs, this methodology introduces the use of Large Language Models (LLMs) to generate human-centered explanations. This approach forms the foundation for the system design described in the following chapter.

LLMs can transform the low-level and technical feature attribution generated by LIME and SHAP into clear and understandable explanations. This conversion helps to understand the model's reasoning for different users, such as security analysts who need detailed insights and managers who prefer concise, high-level summaries.

A primary step of the methodology involves the use of prompt engineering to guide the LLM. This process requires carefully designed prompts that provide the LLM with the output from the XAI tool (e.g., top 10 contributing features with their corresponding SHAP values). The LLM is then instructed to turn this numerical information into readable explanations tailored to specific stakeholders.

The effectiveness of the LLM-enhanced explanations framework is not only evaluated through technical metrics, but also evaluated through focus on human-centered factors such as fidelity. Especially how the explanations influence user trust, reliability, and overall usability in a practical security environment.

4 Specification and Design of the Proposed System

This chapter presents the detailed architecture and design of the proposed human-centered explainability framework for malware classification. The system integrates three main layers: the ML/DL classification layer, the XAI interpretation layer (LIME and SHAP), and the LLM explainability layer. Each layer works together to transform raw malware files into clear, understandable explanations that are useful to different stakeholders. The chapter also explains the data flow between components, the prompt engineering process for the LLM, and how this framework can be extended to other platforms, such as Android or IoT malware detection.

4.1 Overview of the Human-Centered Explainability Framework

The main goal of this research is to address the gap between the technical explanations generated by traditional XAI methods and the practical needs of human stakeholders in cybersecurity. Although machine learning and deep learning models can achieve high accuracy in malware classification, their decision-making process is often hidden, making it difficult for security professionals to trust and use these systems [2, 9].

The proposed framework has three sequential layers, and each layer adds a specific capability that helps to transform binary malware files into human-understandable explanations. The first layer, the **ML/DL Classification Layer**, analyzes the input PE (Portable Executable) file and uses machine learning or deep learning models to classify it as either malicious or benign with a confidence score. The second layer, the **XAI Interpretation Layer**, uses explainability techniques like LIME and SHAP to identify which specific features of the file influenced the model's decision, producing feature importance scores that show which characteristics were most important in the classification. The third layer, the **LLM Explainability Layer**, uses Large Language Models to translate the technical XAI outputs into clear natural language explanations tailored to different audiences, such as security analysts, managers, and end users.

Each layer builds on the previous one, and together they create a complete pipeline that not only detects malware accurately but also explains the decision in a way that humans can understand and trust. Figure 4.1 shows the high-level architecture of this three-layer framework.

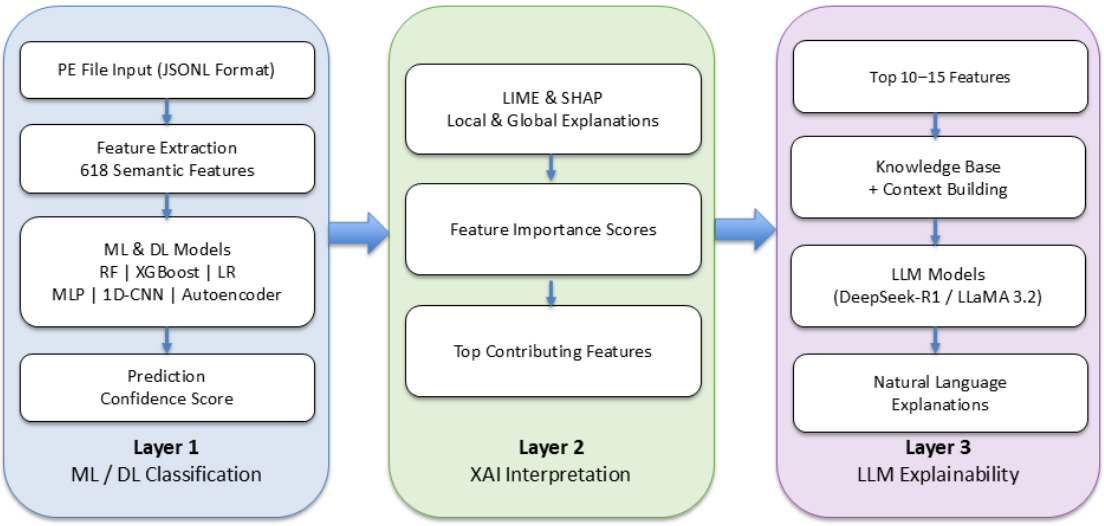

Figure 4.1: Three-Layer Human-Centered Explainability Framework

4.2 System Architecture and Component Design

This section describes the detailed design of each layer and explains how the components interact with each other to create a complete human-centered explainability system.

4.2.1 ML/DL Classification Layer

The classification layer is the foundation of the system. This layer receives a Windows Portable Executable (PE) file as input and produces a classification decision (malicious or benign) along with a confidence score. The layer consists of multiple components that work together to perform the classification task.

Input Processing The first step in this layer is to process the input PE file and extract meaningful features from it. The raw binary file is analyzed to extract various static features such as file headers, section information, import/export tables, and format-agnostic features like byte histograms and string statistics. This feature extraction process converts the binary file into a structured numerical format that can be used as input to machine learning models.

Feature Set Instead of using EMBER's standard 2,381-dimensional feature space (which lacks semantic meaning), this research employs a custom feature extraction methodology to create approximately 618 interpretable features from each PE file [3]. These features are designed to have explicit semantic meaning, which makes them suitable not only for classification but also for explainability.

The feature set is organized into six categories. **Header Features** capture file size, virtual size, timestamps, machine type, operating system version, and other metadata from the COFF and Optional headers. **Section Statistics** include the number of sections, entropy measures (mean, max, min, and standard

deviation), virtual sizes, and counts of high-entropy sections (entropy > 7), which are strong indicators of packed or encrypted malware. **Import Table Features** record the number of imported DLLs, total imported functions, and the presence of specific suspicious libraries such as KERNEL32.dll, ADVAPI32.dll, USER32.dll, and WS2_32.dll. **Export Table Features** count the number of exported functions and symbols. **String Features** capture the total string count, average string length, and the presence of embedded URLs, file paths, and registry keys. Finally, **Byte Statistics** consist of normalized byte histograms and byte-entropy histograms that capture statistical patterns in the binary content.

Algorithm 1 presents the pseudocode for the feature extraction procedure from the EMBER dataset.

Feature Scaling Before feeding the features to the models, they are standardized using Standard Scaler normalization. This process ensures that all features are on a similar scale, which improves model training efficiency and prevents features with large values from dominating the learning process.

Classification Models The system employs six different classification models that represent both traditional machine learning and modern deep learning approaches. On the machine learning side, **Logistic Regression** is a linear classifier with L2 regularization that serves as a simple baseline model with high interpretability. **Random Forest (RF)** is an ensemble of 200 decision trees with a maximum depth of 20, providing good accuracy and natural interpretability through feature importance calculations. **XGBoost** is a gradient boosting model with 200 estimators, a maximum depth of 10, and a learning rate of 0.1, known for achieving state-of-the-art results in malware classification tasks.

On the deep learning side, the **Multi-Layer Perceptron (MLP)** is a neural network with four dense layers (512, 256, 128, and 2 neurons) that includes

Algorithm 1 Feature Extraction from EMBER Dataset

Require: EMBER JSONL dataset files, target samples per class N
Ensure: Balanced feature matrix $\mathbf{X}$ of shape $(2N \times 618)$, label vector $\mathbf{y}$

malicious $\leftarrow$ [], benign $\leftarrow$ []
for each JSONL file in dataset **do**
  for each sample s in file **do**
    $\ell \leftarrow s$.label
    if $\ell = 1$ **and** |malicious| $< N$ **then**
      malicious.append(s)
    else if $\ell = 0$ **and** |benign| $< N$ **then**
      benign.append(s)
    end if
    if |malicious| $\geq N$ **and** |benign| $\geq N$ **then**
      break
    end if
  end for
end for
all_samples $\leftarrow$ malicious $\cup$ benign
$\mathbf{y} \leftarrow [1]^N \cup [0]^N$
for each sample s in all_samples **do**
  $\mathbf{f} \leftarrow \{\}$ ▷ Initialize empty feature dictionary
  Extract general file features (size, virtual size, flags) $\rightarrow \mathbf{f}$
  Extract PE header features (COFF, Optional Header) $\rightarrow \mathbf{f}$
  Extract import table features (DLL names, function counts) $\rightarrow \mathbf{f}$
  Extract export table features $\rightarrow \mathbf{f}$
  Extract section statistics (entropy mean, max, min, std) $\rightarrow \mathbf{f}$
  Normalize byte histogram (256 bins) $\rightarrow \mathbf{f}$
  Normalize byte-entropy histogram (256 bins) $\rightarrow \mathbf{f}$
  Extract string features (URLs, registry keys, paths) $\rightarrow \mathbf{f}$
  Extract data directory features $\rightarrow \mathbf{f}$
  Replace NaN and $\pm\infty$ values with 0 in $\mathbf{f}$
  Append $\mathbf{f}$ to feature list
end for
$\mathbf{X} \leftarrow$ DataFrame(feature list)
return $\mathbf{X}$, $\mathbf{y}$

batch normalization and dropout for regularization, representing the standard deep learning approach for tabular data. The **1D Convolutional Neural Network (1D-CNN)** uses three convolutional blocks (64, 128, and 256 filters) followed by global max pooling, designed to capture local patterns in the feature space. Finally, the **Autoencoder** is an unsupervised model trained only on benign samples that

learns to reconstruct normal file patterns and identifies malware as anomalies — files that cannot be reconstructed accurately.

Output Each model produces a class label (malicious or benign). The output is then passed to the next layer for explainability analysis. Algorithm 2 presents the pseudocode for the complete classification training and evaluation pipeline.

Algorithm 2 ML/DL Classification Training Pipeline

Require: Feature matrix $\mathbf{X}$, label vector $\mathbf{y}$
Ensure: Trained models $\mathcal{M}$, performance metrics $\mathcal{R}$
Split data: $(\mathbf{X}_{\text{train}}, \mathbf{X}_{\text{val}}, \mathbf{X}_{\text{test}}) \leftarrow$ stratified split$(\mathbf{X}, \mathbf{y},\ 70\%/15\%/15\%)$
scaler $\leftarrow$ StandardScaler().fit$(\mathbf{X}_{\text{train}})$
$\mathbf{X}^*_{\text{train}}, \mathbf{X}^*_{\text{val}}, \mathbf{X}^*_{\text{test}} \leftarrow$ scaler.transform$(\mathbf{X}_{\text{train/val/test}})$
for each model $m \in$ {RF, XGBoost, LR, MLP, 1D-CNN, Autoencoder} **do**
    if m is tree-based (RF or XGBoost) **then**
        m.fit$(\mathbf{X}_{\text{train}},\ \mathbf{y}_{\text{train}})$ ▷ No scaling needed
    else if m is Autoencoder **then**
        m.fit$(\mathbf{X}^*_{\text{train}}[\mathbf{y}_{\text{train}} = 0])$ ▷ Train on benign only
    else
        m.fit$(\mathbf{X}^*_{\text{train}},\ \mathbf{y}_{\text{train}})$ ▷ Scaled features
    end if
    $\hat{\mathbf{y}},\ \hat{p} \leftarrow m$.predict$(\mathbf{X}_{\text{test}}),\ m$.predict_proba$(\mathbf{X}_{\text{test}})$
    $\mathcal{R}[m] \leftarrow$ {Accuracy, Precision, Recall, F1, ROC-AUC}
    Save m to disk
end for

4.2.2 XAI Interpretation Layer (LIME and SHAP)

The XAI layer is responsible for explaining why a specific file was classified as malicious or benign. Instead of just saying "this file is malware," this layer identifies which specific features influenced the model's decision. This layer uses two well-known explainability methods: LIME and SHAP.

LIME (Local Interpretable Model-Agnostic Explanations) LIME generates explanations for individual predictions by creating a simple surrogate model that approximates the complex decision boundary around a specific instance [15].

To do this, LIME first takes the original file's feature vector and creates multiple perturbed versions by slightly changing feature values. It then obtains predictions from the trained model for all perturbed samples and trains a simple interpretable model (such as linear regression) on these perturbed samples and their predictions. Finally, it uses this simple model to identify which features had the most influence on the prediction for that specific file. LIME outputs a list of features with their contribution weights, where positive weights indicate features that pushed the prediction towards malicious and negative weights indicate features that pushed towards benign.

SHAP (SHapley Additive exPlanations) SHAP is based on cooperative game theory and assigns an importance value (Shapley value) to each feature by considering all possible feature combinations [14]. SHAP gives explanations that are more consistent and based on strong theory when compared to LIME. For tree-based models (Random Forest, XGBoost), TreeExplainer is used for efficient SHAP value calculation. For neural networks (MLP, 1D-CNN), GradientExplainer or DeepExplainer is applied. For each feature, SHAP calculates how much that feature contributed to moving the prediction away from the baseline (the average prediction across all samples). SHAP outputs both local explanations (why this specific file was classified) and global explanations (which features are most important overall across all files).

Feature Importance Extraction After applying LIME or SHAP to a prediction, the system extracts the top 10-15 most influential features. These features represent the most important characteristics that led to the classification decision. For example, if a file is classified as malicious, the top features might include "high section entropy (7.8)," "missing digital signature," and "imports WS2_32.dll for network communication."

Output Format The XAI layer produces a structured output that includes the sample identifier and true label, the model name and prediction (malicious or benign with confidence), the XAI method used (LIME or SHAP), the list of top features with their names, values, and importance scores, and visualization files such as bar plots for LIME and force plots for SHAP. This output serves as input to the LLM explainability layer, where it will be converted into natural language. Algorithm 3 presents the pseudocode for the XAI analysis procedure.

Algorithm 3 XAI Feature Importance Analysis

Require: Trained model m, feature matrix $\mathbf{X}_{\text{test}}$, label vector $\mathbf{y}_{\text{test}}$, XAI method $\in$ {LIME, SHAP}
Ensure: Feature importance scores Φ for each test sample

if XAI method is LIME **then**
  explainer $\leftarrow$ LimeTabularExplainer($\mathbf{X}_{\text{train}}$, feature_names)
  for each sample $\mathbf{x}_i$ in $\mathbf{X}_{\text{test}}$ **do**
    exp $\leftarrow$ explainer.explain_instance($\mathbf{x}_i$, m.predict_proba, num_features = 15)
    $\Phi_i \leftarrow$ exp.as_list() ▷ List of (feature, weight) pairs
    Save bar plot visualization for Φ_i
  end for
else if XAI method is SHAP **then**
  if m is tree-based **then**
    explainer $\leftarrow$ TreeExplainer(m)
  else
    explainer $\leftarrow$ GradientExplainer(m, $\mathbf{X}_{\text{background}}$)
  end if
  $\Phi \leftarrow$ explainer.shap_values($\mathbf{X}_{\text{test}}$)
  Save summary plot (global importance) and force plots (local)
end if
for each sample i **do**
  top_features$_i \leftarrow$ sort(Φ_i, by = $|\phi|$, descending)[: 15]
  Save {sample_id, $\hat{y}_i$, $\hat{p}_i$, top_features$_i$} to JSON
end for

4.2.3 LLM Explainability Layer

The LLM layer is the most innovative part of this framework. While the XAI layer provides feature importance scores, these scores are often too technical for

non-expert users. For example, an output like "section_entropy_mean: +0.35" is not very meaningful to someone without cybersecurity expertise. The LLM layer solves this problem by translating these technical outputs into clear natural language explanations that are useful for different audiences.

Feature Knowledge Base Before the LLM can generate useful explanations, it needs contextual information about what each feature means. The system includes a curated Feature Knowledge Base that maps technical feature names to human-readable descriptions. This approach is inspired by domain-specific knowledge integration in XAI systems [6]. Each entry in the knowledge base contains a technical feature name (e.g., `section_entropy_mean`), a human-readable name (e.g., Section Entropy Average), a plain-language description of what the feature measures (e.g., "measures randomness in file sections"), a malicious indicator explaining when the feature suggests malware (e.g., "high entropy > 6.5 suggests encryption or packing"), and a benign indicator explaining when the feature suggests legitimate software (e.g., "low to medium entropy < 5.0 indicates normal code"). The knowledge base covers approximately 50 of the most important features commonly identified by XAI analysis, and this domain knowledge is essential for generating accurate and meaningful explanations.

LLM Integration (Ollama + LangChain) The system uses locally hosted Large Language Models through Ollama, which provides privacy and eliminates the need for external API calls. LLMs are based on the transformer architecture [17], which is highly effective for processing language. And these models can be successfully applied to cybersecurity tasks, including malware analysis [4]. Two models are used for comparison: **deepseek-r1:1.5b**, a lightweight model (1.1 GB) that provides fast inference (approximately 1–2 seconds per explanation), and **llama3.2:3b**, a larger model (2.0 GB) that provides higher quality explanations (approximately

2–3 seconds per explanation). The LangChain framework is used to interact with these models, providing robust prompt management, retry logic, and structured output handling. Prompt engineering is critical for guiding LLM outputs in security contexts [18].

Multi-Level Prompt Templates The system uses three different prompt templates, each designed for a specific stakeholder group. The **Level 1 (Security Analyst)** template targets cybersecurity professionals, SOC analysts, and forensic investigators. It requests a detailed technical analysis with specific feature values, behavioral indicators, and investigation recommendations, using technical terminology appropriate for expert audiences. The **Level 2 (Manager)** template is designed for security managers, team leads, and decision-makers. It requests a high-level threat assessment with a risk level (HIGH, MEDIUM, or LOW) and an actionable recommendation (BLOCK, INVESTIGATE, or ALLOW), using business-friendly language with minimal technical jargon. The **Level 3 (End User)** template addresses general computer users and non-technical staff. It requests a simple explanation and clear action items in plain everyday language, explicitly avoiding technical terms such as "entropy," "DLL," or "malware signatures."

Context Building For each explanation request, the system builds a structured context that is inserted into the appropriate prompt template. This context includes the sample's classification result (label, model name, and confidence score), the top 10–15 influential features identified by XAI analysis, and for each feature its human-readable name, current value, importance score, and the relevant malicious or benign indicator from the knowledge base. The context also specifies the XAI method used (LIME or SHAP). Once assembled, this context is inserted into the stakeholder-appropriate prompt template and the complete prompt is sent to the LLM for processing.

LLM Configuration The LLM is configured with a temperature of 0.3, which increases determinism and reduces creativity to ensure consistent explanations across runs. The maximum token limit is set to 500 to keep responses concise, and top-p nucleus sampling is set to 0.9 for coherent output generation.

Explanation Generation Pipeline For each sample to be explained, the system loads the XAI explanation (LIME or SHAP results), extracts the top influential features, and builds the context using the feature knowledge base. It then generates three prompts — one for each stakeholder level — and queries both LLM models (deepseek-r1 and llama3.2). The natural language explanations received from the LLMs are formatted and saved to individual text files organized by stakeholder level. Algorithm 4 presents the pseudocode for the LLM explanation generation pipeline.

Output Format The LLM layer generates three types of output. Individual text files provide separate explanations for each combination of sample, model, XAI method, LLM, and stakeholder level, organized in directories named `analyst/`, `manager/`, and `user/`. A comprehensive JSON summary file contains all explanations together with metadata including timestamps, model names, confidence scores, and top features. An interactive HTML report allows side-by-side comparison of explanations across different levels and models. For a typical run with 4 samples, 2 classification models (XGBoost, MLP), 2 XAI methods (LIME, SHAP), 2 LLMs, and 3 stakeholder levels, the system generates 48 unique explanations.

4.3 Data and Information Flow Design

This section describes how data flows through the entire system, from the raw PE file input to the final human-readable explanations. Understanding this flow is important because it shows how each layer adds value and transforms the data into

Algorithm 4 LLM Explanation Generation Pipeline

Require: XAI outputs Φ, feature knowledge base $\mathcal{K}$, LLM models $\mathcal{L}$, prompt templates $\mathcal{T}$
Ensure: Natural language explanations $\mathcal{E}$ for each stakeholder level
  Verify Ollama service is running (HTTP ping to `localhost:11434`)
  Verify required models are available in $\mathcal{L}$
  for each sample i in selected test samples **do**
    $\hat{y}_i,\ \hat{p}_i \leftarrow$ classification label and confidence
    top_features$_i \leftarrow \Phi_i[:15]$ ▷ Top 15 features from XAI
    Build context:
    for each feature (f_j, v_j, ϕ_j) in top_features$_i$ **do**
      if $f_j \in \mathcal{K}$ **then**
        Enrich with $\mathcal{K}[f_j]$: human name, description, indicator
      else
        Use cleaned feature name as fallback
      end if
    end for
    for each stakeholder level $\ell \in$ {analyst, manager, user} **do**
      prompt $\leftarrow \mathcal{T}[\ell]$.format($\hat{y}_i,\ \hat{p}_i,$ enriched context)
      for each LLM $L \in \mathcal{L}$ **do**
        retries $\leftarrow 0$
        repeat
          $e \leftarrow L$.invoke(prompt)
          retries $\leftarrow$ retries $+ 1$
        until e is valid **or** retries ≥ 3
        $\mathcal{E}[i][\ell][L] \leftarrow e$
        Save e to `llm_explanations/`*level*`/` directory
      end for
    end for
  end for
  Save comprehensive JSON summary and HTML comparison report
  return $\mathcal{E}$

a more interpretable format.

Stage 1: Feature Extraction The feature extraction stage utilizes the EMBER 2018 dataset, which provides raw PE file information in JSONL format. While EMBER's standard methodology can produce 2,381-dimensional features through embedding functions, those features lack semantic meaning and are unsuitable for explainability. Therefore, this research employs a custom feature extraction ap-

proach that creates 618 interpretable features with explicit semantic meaning from the raw EMBER data. These custom features describe various characteristics of Windows PE binary files, including headers, sections, imports, exports, strings, and byte statistics, and are specifically designed to enable XAI interpretation. The processed features are stored in `balanced_dataset.pkl`.

Stage 2: Classification In the classification stage, the 618-dimensional feature vector is processed through feature scaling. The system utilizes various models, including Random Forest, XGBoost, Logistic Regression, MLP, 1D-CNN, and Autoencoder, all stored in the directory (e.g., `xgboost.pkl`, `model.h5`). The output consists of a class label (0=benign, 1=malicious) along with a confidence score ranging from 0.0 to 1.0.

Stage 3: XAI Analysis The XAI analysis stage receives both the feature vector and the model prediction as input, then applies LIME or SHAP techniques to identify feature contributions. This process generates feature importance scores (e.g., section_entropy_mean: +0.35, has_digital_signature: -0.28), which are visualized through bar plots (LIME), force plots (SHAP), and summary plots. The results images are stored in the `xai_explanation` directory.

Stage 4: Context Building The context-building stage takes the top 10-15 features identified from the XAI analysis and enriches them with semantic information. By looking up each feature in the knowledge base, the system extracts human-readable names, descriptions, and malicious/benign indicators, producing an enriched context with comprehensive semantic information about each feature.

Stage 5: Prompt Generation During prompt generation, the system combines the enriched context with the specified stakeholder level (analyst/manager/user)

and inserts this information into the appropriate prompt template. The output is a complete, structured prompt that is ready for processing by the LLM.

Stage 6: LLM Explanation The final stage takes the generated prompt text as input and feeds it to the LLM, which produces a natural language explanation based on the provided context. The LLM generates human-readable text explanations consisting of 3-5 sentences, which are stored in the `llm_explanations/` directory organized by stakeholder level.

Information Transformation Summary The data transformation can be summarized as:

$$\text{Raw EMBER Data} \rightarrow \text{Custom Semantic Features} \rightarrow \text{Prediction} \rightarrow \text{Feature Importance} \rightarrow \text{Context} \rightarrow \text{Prompt} \rightarrow \text{Human Explanation}$$

Each transformation step reduces technical complexity and increases human understandability. Raw EMBER data in JSON format is converted into 618 custom interpretable features designed with semantic meaning for XAI. These semantic features are then passed through the classifier to produce a simple label and confidence score, though without any explanation at this stage. The prediction is subsequently analyzed by LIME or SHAP to yield technical numerical attributions (feature importance scores). These importance scores are then enriched with domain knowledge from the knowledge base to produce contextualized features. Finally, the contextualized features are processed by the LLM to generate fully human-understandable, stakeholder-appropriate natural language explanations. This progressive transformation ensures that the final output is both technically accurate (faithful to the model's decision) and practically useful (understandable by the intended audience).

4.4 Prompt Design and LLM Integration Process

Prompt engineering is a critical component of the LLM explainability layer. The quality of the generated explanations depends heavily on how well the prompts are designed [13, 18]. This section describes the prompt design principles and provides detailed examples of the templates used in the system.

The following principles guide the design of prompts for this system:

1. Clarity and Specificity Each prompt clearly states the LLM's role, the input information available, and the expected output format. Vague instructions lead to inconsistent or irrelevant responses [18].

2. Context Provision Prompts include all necessary context, such as the file's classification result, the model used, the XAI method applied, and the list of top features with their values and importance scores. Without this context, the LLM cannot generate accurate explanations [13].

3. Audience-Appropriate Language Each prompt explicitly instructs the LLM about the target audience and the appropriate language level. For example, the analyst prompt allows technical jargon, while the user prompt prohibits it [13].

4. Structured Output Requirements Prompts specify the desired output structure, such as "provide an executive summary with risk level and recommended action" for managers or "explain in 3-4 technical sentences" for analysts.

5. Domain Knowledge Integration Prompts include domain-specific information from the feature knowledge base, such as what high entropy means (code packing) or why missing signatures are suspicious (lack of authentication).

6. Conciseness While prompts must be detailed enough to guide the LLM, they should also be concise to avoid confusion and reduce processing time [18].

4.4.1 Detailed Prompt Templates

To support different stakeholders (analysts, managers, and end users), separate LLM prompt templates were designed for each group. The complete prompt templates used by the LLM layer are provided in the Appendix B.

4.4.2 LLM Integration Implementation

The integration of LLMs is implemented using three main components. The first is the **Ollama Server**, a local LLM deployment tool that allows running models like Llama, Mistral, and Deepseek on a local machine without sending data to external APIs. The Ollama server is started using the command `ollama serve`, which listens on `http://localhost:11434`. The second component is the **LangChain Framework**, which serves as the interface between the Python code and the Ollama server. LangChain provides an LLM class that simplifies querying the model, structured prompt template management, built-in retry logic for handling failures and timeouts, and response parsing utilities for extracting and formatting LLM outputs.

The integration workflow begins by checking whether the Ollama service is running via an HTTP ping to `localhost:11434` and verifying that the required models (deepseek-r1:1.5b and llama3.2:3b) are available. For each explanation request, the system builds a prompt from the template and context, sends it to the LLM via LangChain, receives the generated explanation, validates and formats the response, and saves it to the appropriate output file. Once all explanations are generated, a summary JSON file and an HTML comparison report are produced.

The system also includes robust error handling throughout this workflow. If

Ollama is not running, a clear error message with startup instructions is displayed. If a specific model is not available, that model is skipped and processing continues with the remaining models. If LLM generation fails, the system retries up to three times with exponential backoff. If all retries fail, an error message is saved in place of the explanation to preserve the output structure.

4.5 Generalization Considerations (Windows to Android or IoT)

While this research focuses on Windows PE malware classification, the proposed framework is designed to be generalizable to other platforms and file types. This section discusses how the three-layer architecture can be adapted for Android APK files, IoT firmware, or other malware analysis domains.

4.5.1 Framework Adaptability

The three-layer framework (Classification → XAI → LLM) is platform-agnostic by design. The core principles remain the same regardless of the file type: the first layer extracts features and classifies using ML/DL models, the second layer identifies important features using LIME/SHAP, and the third layer translates technical outputs to natural language using the LLM.

4.5.2 Adaptation for Android Malware

To extend the framework to Android APK file analysis, the following modifications would be needed:

Feature Engineering Instead of PE headers and sections, Android-specific features would be extracted. Manifest features capture the permissions requested by

an application, such as `SEND_SMS`, `READ_CONTACTS`, and `ACCESS_FINE_LOCATION`, as well as the declared services, broadcast receivers, and content providers. These elements are widely recognized as strong indicators of potentially malicious behavior in Android applications. API call features represent patterns of Android API usage, with particular attention to sensitive APIs related to location, camera access, microphone usage, and network communication. Package metadata includes general application information such as the app name, version details, and signing certificate data. Finally, resource features cover application components such as layout files, drawable assets, and string resources, which may also provide useful contextual information for analysis [21, 22, 23].

Knowledge Base Updates The feature knowledge base should contain domain-specific information related to Android. For example, the high `_ risk_permissions` of the entry should explain that when an application requests SMS permissions together with network access, it may attempt to send premium-rate messages or steal SMS data from users [23]. The obfuscation entry _detected should state that the use of code obfuscation tools, such as ProGuard or DexGuard, can conceal malicious activities within the application [24]. Similarly, the entry for dynamic_code_loading should clarify that loading code at runtime can help an application evade static analysis techniques and is a strategy frequently used by malware [22].

Prompt Adjustments The LLM prompts would need minor adjustments to use Android-specific terminology. References to "PE file" would be replaced with "APK file," "DLL imports" with "Android API calls," and "section entropy" with "DEX file patterns." Android-specific recommendations would also be included, such as "Check Google Play Protect status" instead of "Isolate host."

4.5.3 Adaptation for IoT Malware

For IoT firmware analysis, the framework can be adapted in a similar manner.

Feature Engineering IoT-specific features would include firmware metadata such as device type, vendor, firmware version, and update mechanism [25]. Embedded executable features would capture the presence of BusyBox, shell scripts, and embedded binaries. Network configuration features would record hardcoded IP addresses, default credentials, and backdoor ports [25]. Cryptographic artifact features would flag weak or missing encryption and hardcoded keys. String pattern features would identify bot commands, IRC channels, and malicious URLs embedded in the firmware.

Knowledge Base Adaptation IoT-specific indicators would be added to the knowledge base. The entry for *default_ credentials* would explain that the presence of default vendor passwords (e.g., admin/admin) enables easy compromise — a technique systematically exploited by the Mirai botnet, which used a hardcoded list of 62 default credential pairs to compromise millions of IoT devices via Telnet [26]. The entry for *telnet_ enabled* would note that an unauthenticated Telnet service is a primary entry point for Mirai-like botnets that scan and compromise devices at internet scale [26]. The entry for *http_ server_ anomalies* would explain that unusual web server configurations embedded in firmware may indicate command-and-control infrastructure, consistent with patterns identified in large-scale IoT firmware security analyses [25].

4.5.4 Cross-Platform Challenges

Adapting the framework to new platforms involves several challenges. Different platforms have different file structures and available metadata, so some features

that work well for Windows PE files (like section entropy) may not have direct equivalents in Android APKs or IoT firmware. While EMBER provides a large labeled dataset for Windows malware [3], similar high-quality datasets may not exist for other platforms, which affects model training and evaluation. Creating a comprehensive feature knowledge base also requires deep domain expertise for each platform, since security indicators that apply to Windows may not apply to Android or IoT. Finally, models trained on one platform do not transfer to another, so new models must be trained for each platform and their performance may vary based on data quality and feature discriminability.

5 Experimentation and Testing

This chapter describes the experimental setup, implementation details, and results of the proposed human-centered explainability framework for malware classification. This experiment has three main parts: the ML/DL classification layer, the XAI interpretation layer that uses LIME and SHAP methods, and the LLM explainability layer that uses local running LLMs. Each layer is evaluated, and the results are discussed in terms of classification accuracy, performance, quality, and usefulness of the generated explanation for different stakeholders.

5.1 Experimental Setup

5.1.1 Dataset and Preprocessing

This experiment uses the EMBER 2018 dataset [3], which is a publicly available benchmark dataset for Windows PE malware classification. The full EMBER dataset contains about 1.1 million PE file records stored in JSONL format. Each record includes raw features that are statically extracted from the binary file.

For this study, a balanced subset of the dataset was created to ensure a fair comparison between malicious and benign files. This subset dataset includes 1,000 malicious samples (label = 1) and 1,000 benign samples (label = 0), making a total of 2,000 samples with equal class distribution. This balance helps to avoid bias towards one class and shows how well the model performs on a balanced dataset.

The feature extraction process is clean and reliable to make sure that there are no missing or infinite values, and all features are in numeric form.

From each PE file record, 618 interpretable features were extracted using the custom pipeline described in Chapter 4. These features are categorized into six groups: general file properties, PE header metadata, import and export table details, section statistics, string-based features, and byte-level histogram data. The dataset was divided into training (70%), validation (15%), and test (15%) sets using stratified random splitting to keep the class balance in each set. Table 5.1 shows the dataset breakdown.

Table 5.1: Dataset composition and split used in the experiments.

Split	**Total Samples**	**Malicious**	**Benign**	**Percentage**
Training	1,400	700	700	70%
Validation	300	150	150	15%
Test	300	150	150	15%
Total	2,000	1,000	1,000	100%

5.1.2 Experimental Environment

All experiments were run on a local development machine without using any cloud services. The setup used Python 3.9.25 with key libraries including scikit-learn for machine learning, XGBoost for gradient boosting, and TensorFlow/Keras for building and training deep learning models. LIME (v0.2.0.1) and SHAP (v0.44) were used for explainability analysis, while LangChain and Ollama handled local LLM integration. The Ollama server ran locally on the same machine, so no external API calls were needed.

5.2 Classification

5.2.1 Machine Learning Model Performance

Three traditional machine learning models were trained and evaluated on the test set: Logistic Regression, Random Forest, and XGBoost. These models were selected because they represent a range of model complexity, from the simple linear classifier (Logistic Regression) to the more powerful ensemble methods (Random Forest and XGBoost). A further practical reason is that both LIME and SHAP have efficient support for tree-based models, which matters for the XAI pipeline. Logistic Regression and Random Forest were applied to the raw (unscaled) features since tree-based models do not require feature scaling, while Logistic Regression was applied to standardized features using the fitted Standard Scaler.

Table 5.2 presents the classification metrics for each ML model on the test set. Performance is reported using four metrics: Accuracy, Precision, Recall, F1-Score, and ROC-AUC, which together give a complete picture of classification.

Table 5.2: Classification performance of ML models on the test set (300 samples, balanced)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**ROC-AUC**
Logistic Regression	0.880	0.870	0.893	0.882	0.928
Random Forest	0.907	0.863	0.967	0.912	0.982
XGBoost	**0.970**	**0.961**	**0.980**	**0.970**	**0.997**

XGBoost clearly outperforms the other two models in all metrics. It achieved an accuracy of 97.0%, a precision of 96.1%, a recall of 98.0%, an F1-Score of 97.0%, and a near-perfect ROC-AUC of 0.997. This strong result is consistent with existing literature on malware detection, where gradient boosting models consistently perform well on tabular feature sets [11]. The high recall (98.0%) is particularly important in a security context because it means XGBoost missed very few actual malware samples (only 1% false negative rate), which is the more dangerous type of error in

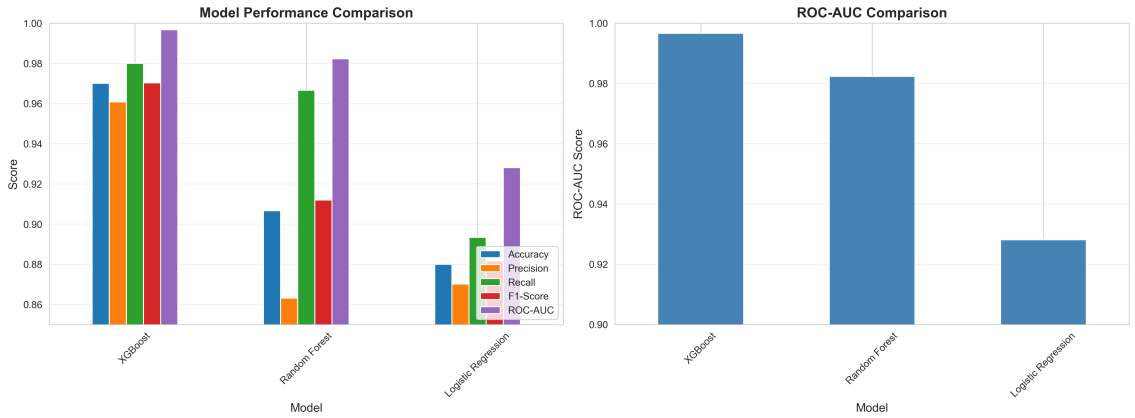

Figure 5.1: Comparative performance of ML models (Logistic Regression, Random Forest, XGBoost) across five classification metrics on the test set

malware detection.

Random Forest comes in second with a 90.7% accuracy and an F1-Score of 91.2%. Notably, its recall of 96.7% is also high, indicating that it does not miss many malware files. However, its precision (86.3%) is somewhat lower than XGBoost, which means it produces more false positives, meaning flagging some benign files as malicious. The ROC-AUC of 0.982 is excellent and indicates that the model has very good overall discriminative ability. Logistic Regression, as expected for a linear model, performs the lowest of the three with 88.0% accuracy and an F1-Score of 88.2%, but its ROC-AUC of 0.928 still shows reasonable separability.

Figure 5.1 shows the comparative performance of all three models across the five evaluation metrics. Figure 5.2 shows the confusion matrices for all three models, and Figure 5.3 shows the ROC curves.

5.2.2 Deep Learning Model Performance

Three deep learning models were also trained and evaluated: a Multi-Layer Perceptron (MLP), a one-dimensional Convolutional Neural Network (1D-CNN), and an Autoencoder. The MLP and 1D-CNN are supervised classification models similar to the ML models above, while the Autoencoder is an anomaly-detection model trained

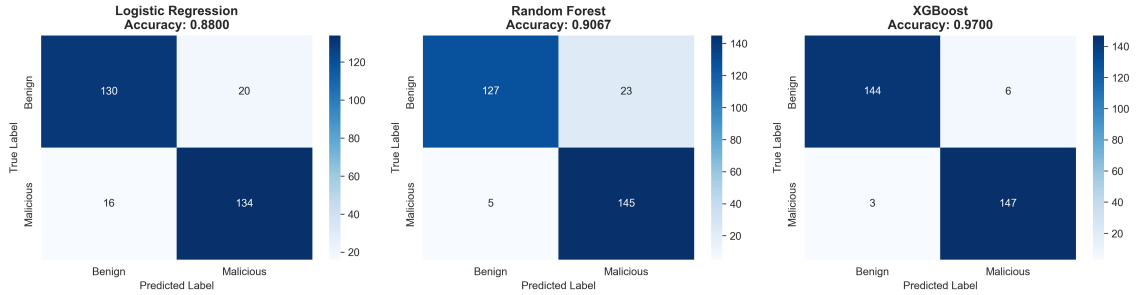

Figure 5.2: Confusion matrices for all three ML models on the test set, showing true positive/negative and false positive/negative counts

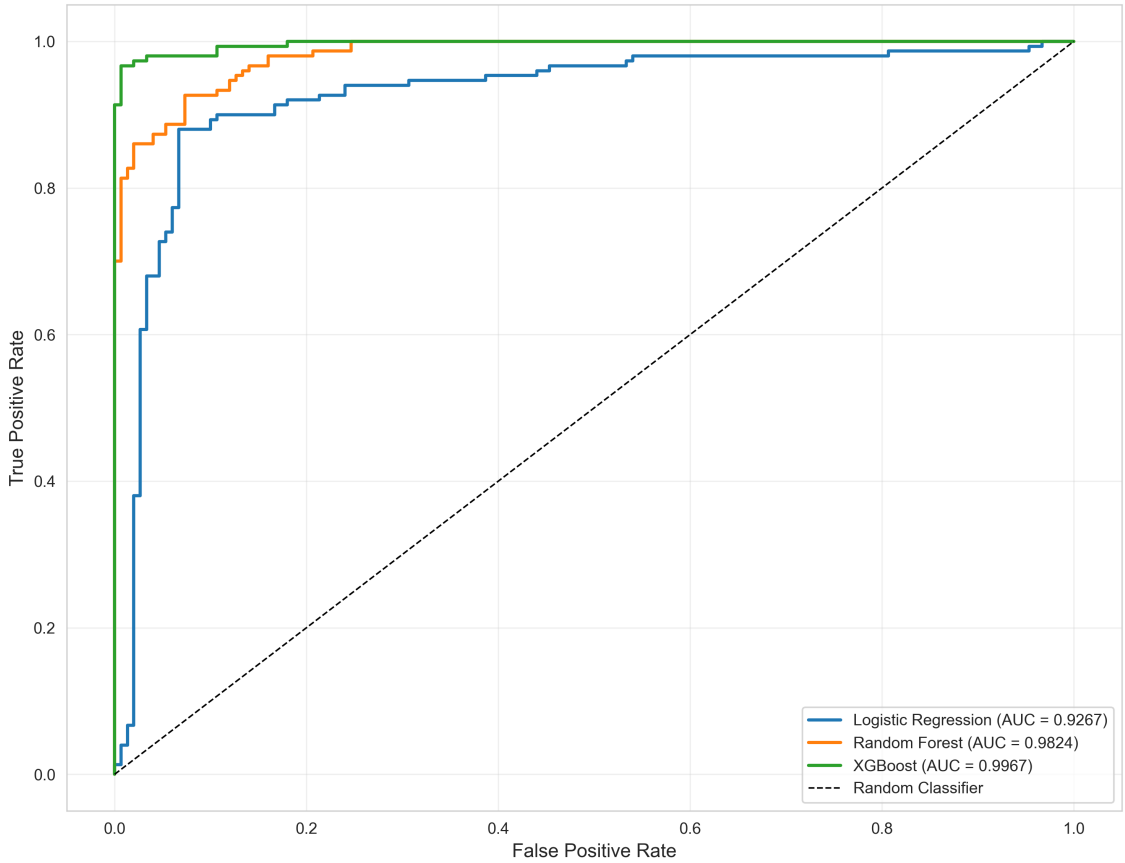

Figure 5.3: ROC curves for all three ML models. XGBoost achieves a near-perfect AUC of 0.997

only on benign samples. All deep learning models were trained using the scaled feature set (standardized using Standard Scaler) and trained with early stopping based on validation loss to prevent overfitting. Table 5.3 shows the results.

Table 5.3: Classification performance of DL models on the test set (300 samples, balanced)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**ROC-AUC**
MLP	**0.910**	**0.930**	0.887	**0.908**	**0.970**
1D-CNN	0.903	0.885	**0.927**	0.906	0.971
Autoencoder	0.483	0.419	0.087	0.144	0.614

Among the deep learning models, the MLP performs the best with 91.0% accuracy, 93.0% precision, and an F1-Score of 90.8%. The 1D-CNN achieves a very similar result with 90.3% accuracy and an F1-Score of 90.6%. The small difference between the MLP and 1D-CNN suggests that neither architecture has a strong advantage over the other for this particular feature set, although the MLP's higher precision means it produces fewer false positives. The 1D-CNN's slightly higher recall (92.7% vs. 88.7%) means it catches more malware but at the cost of more false positives.

The Autoencoder's performance tells a different and important story. With an accuracy of only 48.3%, an F1-Score of 14.4%, and a recall of just 8.7%, the Autoencoder essentially fails as a classifier for this dataset. The fundamental reason is that the Autoencoder, being trained only on benign samples, relies on reconstruction error as a proxy for maliciousness. The assumption is that malware samples will produce high reconstruction errors because they are "different" from benign files. However, with 618 interpretable features (as opposed to raw byte sequences), many malicious samples appear structurally similar to benign files in feature space, which means the Autoencoder cannot distinguish them effectively. Anomaly detection approaches are less effective for malware classification when using semantic, interpretable features, and supervised learning is clearly preferable for this task.

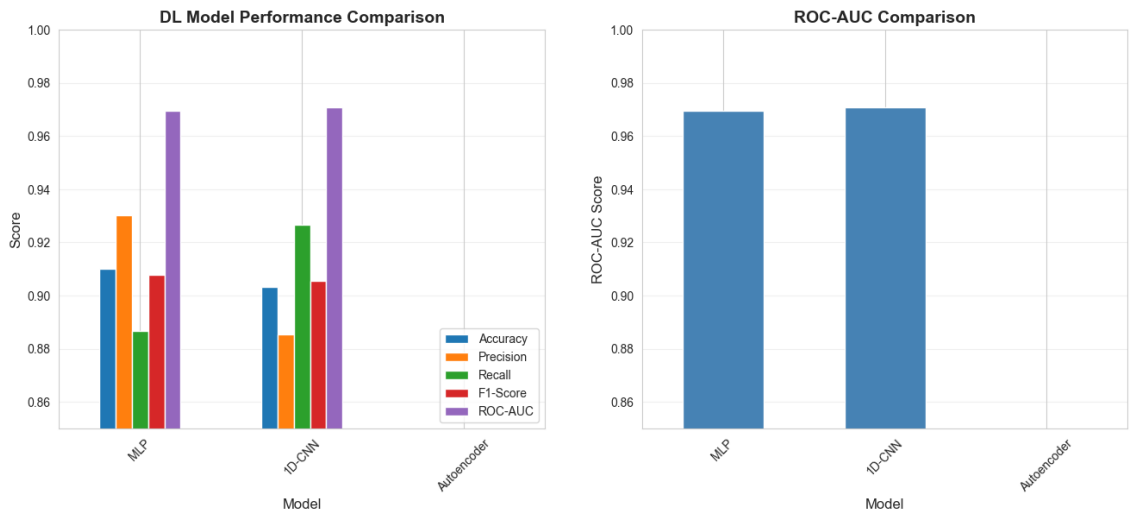

Figure 5.4: Comparative performance of DL models (MLP, 1D-CNN, Autoencoder) across classification metrics. The Autoencoder performs poorly due to the limitations of anomaly detection on interpretable feature sets.

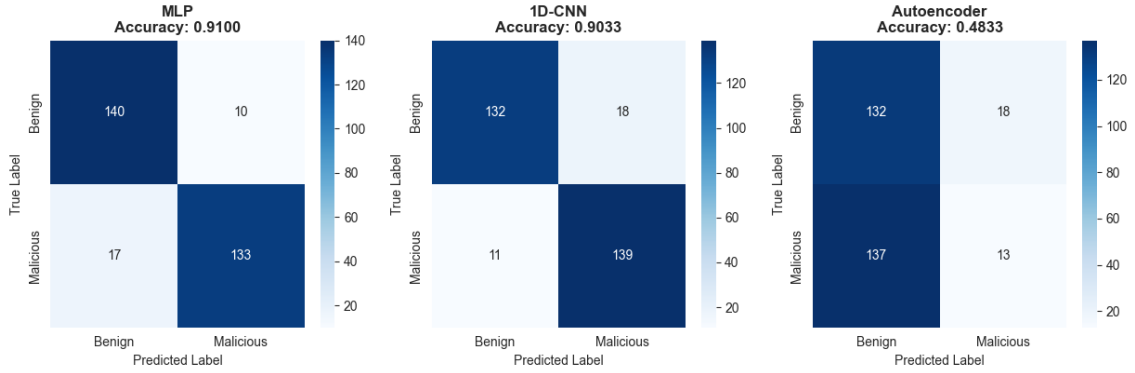

Figure 5.5: Confusion matrices for all three DL models on the test set

Comparing the deep learning results with the ML results, the MLP (91.0%) is slightly better than Random Forest (90.7%) but considerably behind XGBoost (97.0%). This suggests that for this type of structured, tabular feature data, gradient boosting methods are more suitable than neural networks. This observation is consistent with findings in the broader machine learning literature, where tree-based gradient boosting often outperforms neural networks on structured data [20]. Figure 5.4 and Figure 5.5 show the DL model performance and confusion matrices.

Figure 5.6, Figure 5.7, and Figure 5.8 show the training history for mlp, 1d-cnn,

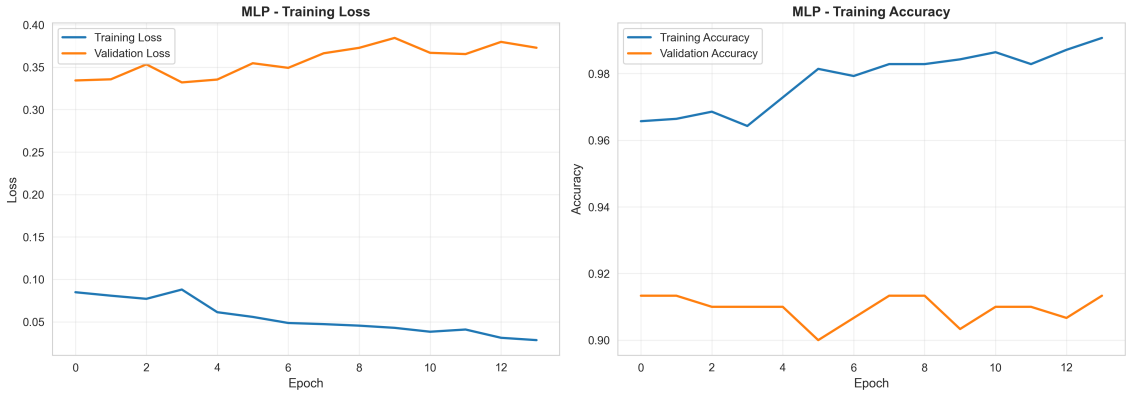

Figure 5.6: Training and validation loss curves for the MLP training history

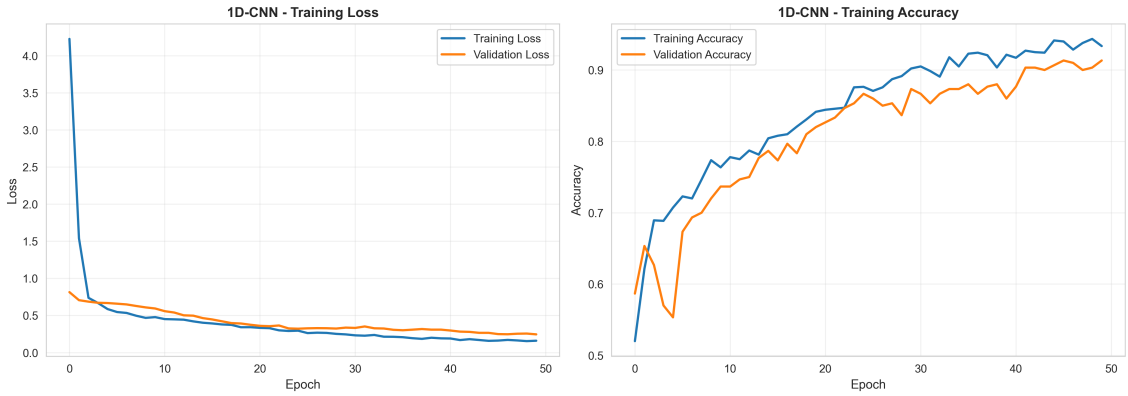

Figure 5.7: Training and validation loss curves for the 1D-CNN training history

and auto encoder.

5.2.3 Overall Model Comparison

Table 5.4 shows a combined comparison of all six models. XGBoost achieved the best performance across most evaluation metrics, so it was chosen as the main model for the later XAI and LLM explainability experiments. The MLP model was the best deep learning model for the same stages. Using these two models also allows a comparison of how LIME and SHAP explanations behave for a tree-based model against a neural network.

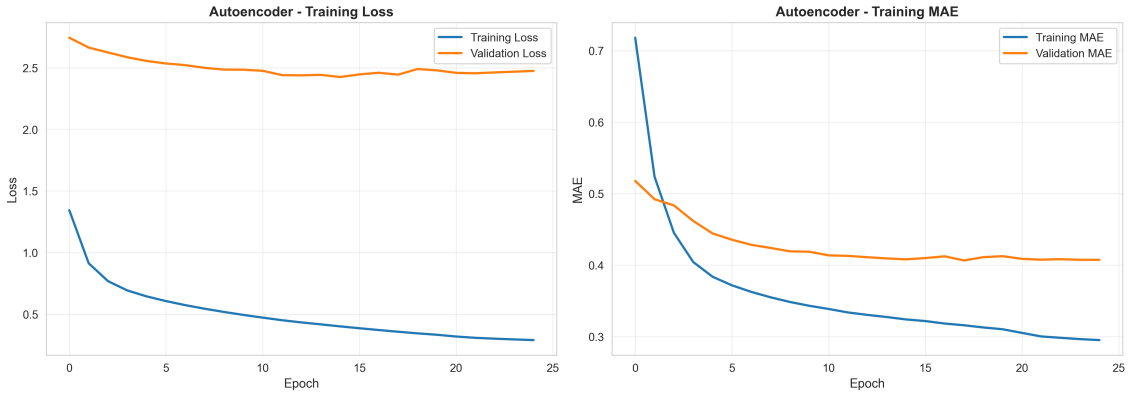

Figure 5.8: Autoencoder training history (reconstruction loss)

Table 5.4: Unified comparison of all six classification models (ML and DL) on the test set

Category	Model	Acc.	Prec.	Rec.	F1	AUC
ML	XGBoost	**0.970**	**0.961**	**0.980**	**0.970**	**0.997**
	Random Forest	0.907	0.863	0.967	0.912	0.982
	Logistic Regression	0.880	0.870	0.893	0.882	0.928
DL	MLP	0.910	0.930	0.887	0.908	0.970
	1D-CNN	0.903	0.885	0.927	0.906	0.971
	Autoencoder	0.483	0.419	0.087	0.144	0.614

5.3 XAI Analysis

5.3.1 LIME Explanations for ML Models

The LIME method was applied to six selected test samples for each of the three ML models: Random Forest, XGBoost, and Logistic Regression. The purpose was to identify which features LIME considers most influential for each prediction. Table 5.5 summarizes the top features highlighted by LIME for each model.

There are several observations that can be made from this analysis. First of all, the `compilation_timestamp` feature appears among the top features for both Random Forest and XGBoost. This is important because the PE compilation timestamp is a metadata field that malware authors often modify to make a file look older or to hide the real development time. Unusual timestamps, such as very old dates or dates far in the future, are commonly treated as indicators during static malware

Table 5.5: Top features identified by LIME for each ML model (aggregated across 6 test samples).

Model	**Top LIME Features**
Random Forest	`compilation_timestamp`, `dd_architecture_va`, `dd_exception_table_size`, `has_section_edata`, `byte_freq_065`, `sizeof_heap_commit`, `vsize_to_size_ratio`, `dd_tls_table_va`
XGBoost	`compilation_timestamp`, `dd_exception_table_size`, `dd_clr_runtime_header_va`, `byte_freq_196`, `dd_architecture_size`, `has_section_edata`, `byte_freq_217`, `byte_entropy_163`
Logistic Regression	`dd_exception_table_va`, `dd_certificate_table_size`, `byte_entropy_184`, `byte_entropy_121`, `byte_entropy_255`, `byte_freq_207`, `byte_freq_093`

analysis.

Second, several data directory features, such as `dd_exception_table_size`, `dd_architecture_va`, and `dd_clr_runtime_header_va`, also appear many times. These fields describe the presence and size of specific PE data directory entries. Unusual or zero values in these fields may indicate stripped or malformed binaries, which is a known technique used to evade detection.

Third, byte frequency and byte entropy features (`byte_freq_*`, `byte_entropy_*`) appear more often in the top features for Logistic Regression compared to the tree-based models. This likely happens because linear models are more sensitive to the statistical distribution of byte values, while tree-based models can capture more complex relationships between features.

Figures 5.9, 5.10, and 5.11 show the LIME bar plots for one representative test sample for each of the three models. Each bar represents a feature's contribution to the classification decision, with positive values (green) pushing the prediction towards malicious and negative values (red) pushing towards benign.

Figure 5.9: LIME explanation for Logistic Regression. The linear model relies more heavily on byte frequency and entropy statistics compared to tree-based models

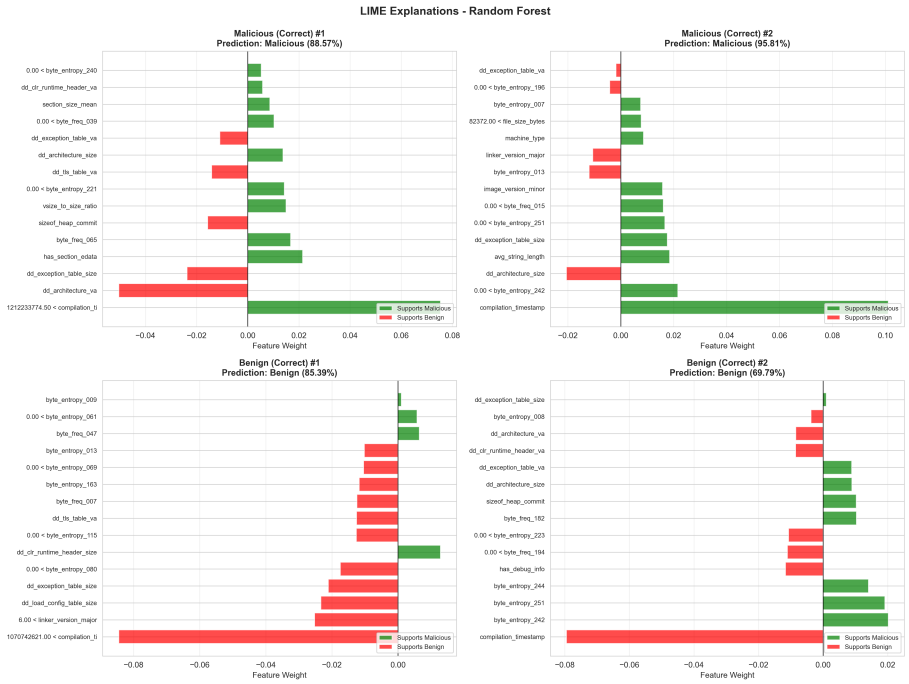

Figure 5.10: LIME explanation for Random Forest on a sample classified as malicious. The bar chart shows feature importance weights; positive values indicate features contributing to the malicious prediction

Figure 5.11: LIME explanation for XGBoost on the same malicious sample, showing the top influential features with their importance scores

5.3.2 SHAP Explanations for ML Models

SHAP was used on 200 test samples for each ML model in this study. From this analysis, both the local and global importance of features were examined. For Random Forest and XGBoost models, TreeExplainer was applied because it is suitable for tree-based methods. In the case of Logistic Regression, LinearExplainer was used. Table 5.6 lists the top 10 features by mean absolute SHAP value for each model.

After the analysis was completed, one common result was seen in all three models. The feature `compilation_timestamp` was identified as the most important global feature. This result is important because even though different types of models were tested, the same feature was selected again and again. It shows that the compilation time of the PE file has a strong role in separating malware from normal files. It was also noticed that byte entropy features such as `byte_entropy_242`, `byte_entropy_249`, and `byte_entropy_255` were repeatedly highlighted. These

Table 5.6: Top 10 features by mean absolute SHAP value for each ML model (200 test samples)

Model	**Top SHAP Features (ranked)**
Random Forest	`compilation_timestamp`, `byte_entropy_242`, `linker_version_major`, `byte_entropy_249`, `byte_entropy_251`, `byte_entropy_243`, `byte_entropy_255`, `byte_entropy_250`, `byte_entropy_246`, `dd_tls_table_va`
XGBoost	`compilation_timestamp`, `byte_entropy_243`, `byte_entropy_255`, `byte_freq_077`, `dd_tls_table_va`, `byte_freq_113`, `byte_entropy_147`, `sizeof_code`, `byte_freq_002`, `byte_entropy_120`
Logistic Regression	`compilation_timestamp`, `num_sections`, `dll_characteristics`, `has_section_rsrc`, `byte_freq_222`, `os_version_major`, `byte_freq_027`, `byte_freq_014`, `byte_freq_158`, `byte_freq_044`

features are related to high-entropy byte areas inside PE files. In many cases, high entropy is linked with compressed or encrypted code sections. Such sections are commonly found in packed malware samples. In addition, the feature `sizeof_code` appeared among the top features in the XGBoost model. The size of the executable code section gives basic information about the structure of the binary file. Even though it is a simple indicator, it can still provide useful support in malware classification.

In the case of Logistic Regression, some structural features were highlighted together with byte frequency features. Features such as `num_sections`, `dll_characteristics`, and `os_version_major` appeared as important in the model. The feature `num_sections` represents the number of sections present in a PE file. It is often seen that malware files contain an unusual number of sections when compared to normal software.

The feature `dll_characteristics` includes different flags and properties related to the security settings of the PE file. If unusual or empty values are present in this

field, it can be a sign that the file was changed or intentionally modified. In some cases, such behavior is linked with malicious executables.

When LIME and SHAP were compared, it was found that both methods highlighted `compilation_timestamp` as important. However, the order of other features was not the same. LIME explains one sample at a time, so the importance depends on that specific file. Because of this, some features may look more important in one case and less important in another. SHAP, on the other hand, calculates importance by looking at many samples together. For this reason, differences in ranking were observed. Such differences are considered normal since both methods work in different ways.

Figure 5.12 shows a direct comparison of the top features identified by LIME and SHAP for the ML models.

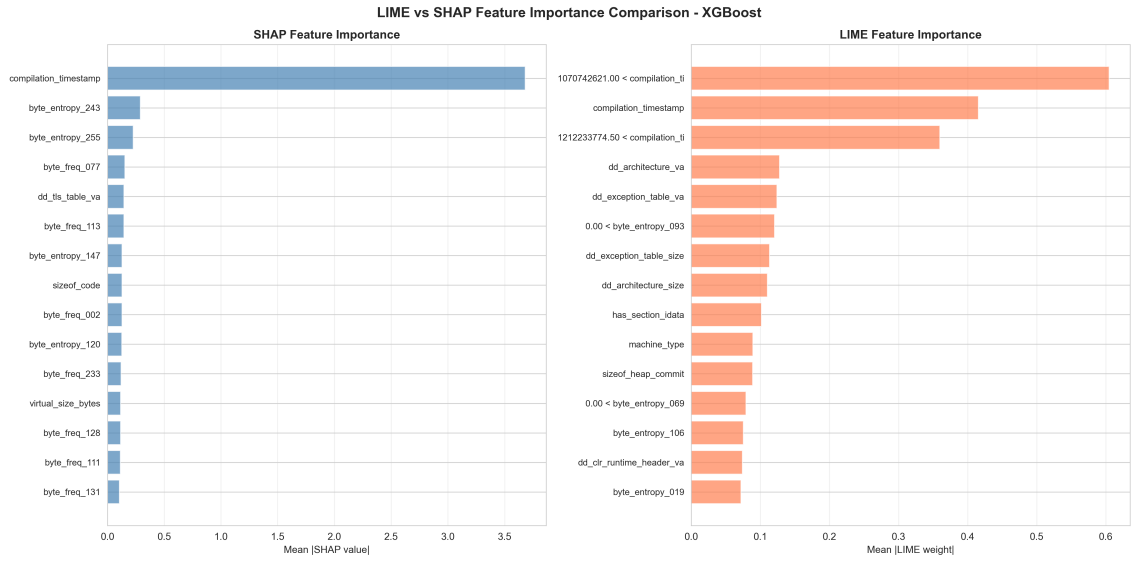

Figure 5.12: Visual comparison of LIME and SHAP top features for ML models. `compilation_timestamp` is the only feature that consistently appears in the top rankings of both methods across all three models

5.3.3 XAI Results for Deep Learning Models

LIME and SHAP were also applied to the deep learning models such as MLP, 1D-CNN, and autoencoders. Since MLP and 1D-CNN are based on gradients, GradientExplainer was used for these models.

Table 5.7: Top 10 features by mean absolute SHAP value for DL models (200 test samples).

Model	Top SHAP Features (ranked)
MLP	`compilation_timestamp`, `dll_characteristics`, `dd_tls_table_size`, `num_sections`, `os_version_major`, `coff_characteristics`, `has_exports`, `subsystem_version_major`, `dd_certificate_table_size`, `has_section_rsrc`
1D-CNN	`compilation_timestamp`, `coff_characteristics`, `dll_characteristics`, `has_exports`, `os_version_major`, `has_relocations`, `has_section_rsrc`, `byte_entropy_137`, `has_resources`, `has_tls`
Autoencoder	`byte_entropy_035`, `byte_entropy_066`, `byte_entropy_137`, `byte_entropy_044`, `byte_freq_173`, `byte_freq_049`, `byte_freq_235`, `byte_entropy_246`, `byte_entropy_051`, `byte_freq_211`

It was found that in both MLP and 1D-CNN, the feature `compilation_timestamp` was identified as the most important one. A similar result was already observed in the ML models. When the same feature is selected by different types of models, it can be understood that this feature plays an important role in malware classification. Some differences were also observed between DL models and ML models. In the DL models, Boolean structural features were given more importance. Features such as `has_exports`, `has_relocations`, `has_section_rsrc`, `has_resources`, and `has_tls` were included among the top features in MLP and 1D-CNN. These features only indicate whether a specific structure exists inside the PE file or not. From the SHAP results, it can be

seen that neural network models relied more on the presence or absence of these structures. On the other hand, tree-based models were more focused on numerical features, especially byte entropy values.

For the Autoencoder model, only byte entropy and byte frequency features were highlighted. This behavior is related to the working mechanism of the Autoencoder. It is trained to reconstruct the normal patterns of files. When unusual byte-level statistics are present, a higher reconstruction error is produced. However, as discussed in the classification results, these statistical differences alone were not sufficient to clearly distinguish malware from benign files when both have similar overall byte distributions.

Figure 5.13 shows a summary comparison of ML versus DL XAI results.

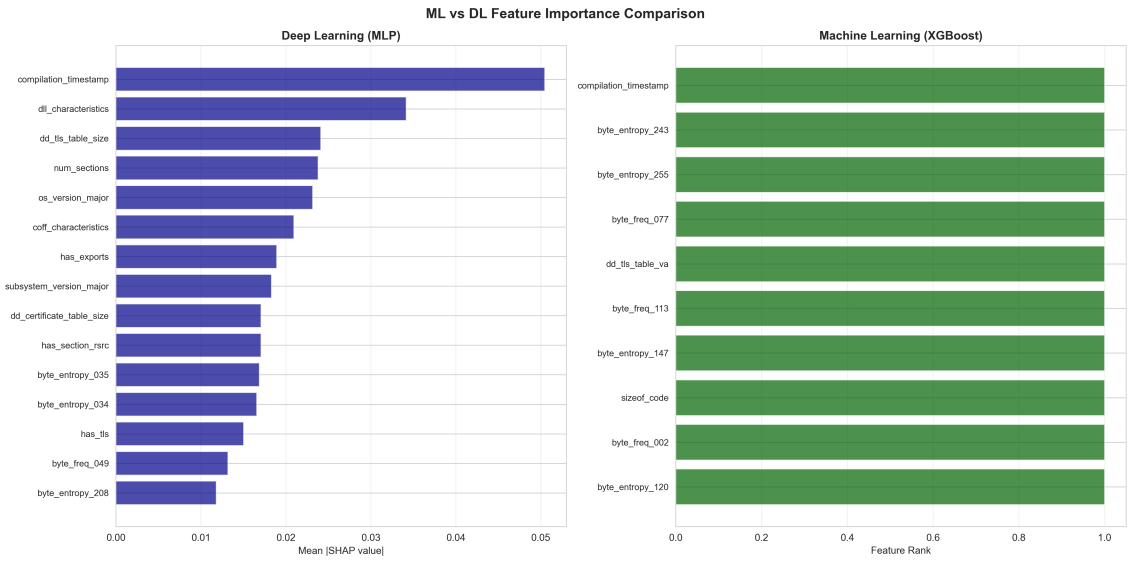

Figure 5.13: Comparison of top features from XAI analysis across ML and DL models. `compilation_timestamp` is the only feature common to both the ML and DL top-feature sets, confirming its universal importance

5.4 Integration of LLM Module for Explainability

5.4.1 Explanation Generation Setup

For the LLM explainability, four test samples were selected. Two classification models were used: XGBoost and MLP. Both XAI methods were used: LIME and SHAP, and both LLM models, **deepseek-r1:1.5b** (1.1 GB) and **llama3.2:3b** (2.0 GB), were used to generate explanations. Explanations were generated for all three stakeholders, such as the analyst (technical), the manager (executive), and the user (non-technical). In total, there will be 96 ($4 \times 2 \times 2 \times 2 \times 3$) unique explanations generated. It was made sure that the Ollama server is running before generating explanations and retried up to 3 times. All explanations were saved in a text file, organized by stakeholders.

5.4.2 Analyst-Level Explanations

To generate explanations for analysts, the LLM is instructed to produce detailed technical explanations tailored to cybersecurity professionals. Here, suspicious indicators (identified by LIME or SHAP) are provided, linked to malware behavior patterns, and used with a knowledge base to generate explanations and recommend investigative steps. Table 5.8 shows a sample analyst-level explanation, which is generated by llama3.2:3b for a malicious sample classified by XGBoost, and LIME was used.

From this example, it can be observed that the analyst-level explanation converts the raw LIME feature importance values into understandable cybersecurity findings. The important features are described, and reasons are given for why they can be considered suspicious in the case of Windows PE malware.

Table 5.8: Example analyst-level explanation generated by llama3.2:3b for a malicious XGBoost/LIME prediction

Field	**Content**
Sample	Index 0 (True label: MALICIOUS)
Model / XAI	XGBoost / LIME
Confidence	87.0%
Top Features	linker_version_minor: 50.00; has_relocations: 0.00; linker_version_major: 2.00; dll_characteristics: 0.00; has_debug_info: 0.00
LLM Explanation	"The Windows PE file was classified as MALICIOUS due to several significant technical indicators, including a linker version minor of 50.00, which is an unusual value typically associated with malicious code. The presence of relocations (Has Relocations: 0.00) and debug information (Has Debug Info: 0.00) also raised suspicions, as these features are often used to evade detection by legitimate applications. Furthermore, the low code-to-file ratio and high size of headers suggest a potential attempt to hide or obfuscate malicious code. These indicators warrant further investigation using techniques such as memory analysis or API hooking to determine the file's actual functionality."

5.4.3 Manager-Level Explanations

The manager-level explanation was designed in such a way as to provide a short and concise summary for senior managers. At this level, managers do not require technical details; instead, they need a summary to make quick decisions. This explanation is organized into three main parts: the first is a brief description of the threat, the second is the risk level (HIGH, MEDIUM, or LOW), and the third is the recommended action (BLOCK, INVESTIGATE, or ALLOW).

In this explanation, rather than using technical terms such as "linker version" or "PE header", specify the level of risk and the action to be taken. Since the output is in a structured format, it can easily be used in daily security operations.

Table 5.9: Example manager-level explanation for the same malicious sample

Field	Content
LLM Output	"Our security systems have detected a malicious file on the network, which poses a significant risk to our system's stability and data integrity. We recommend immediate action to prevent potential harm."
Risk Level	**HIGH**
Recommended Action	**BLOCK**
Confidence	87.0%

Table 5.10: Example user-level explanation for the same malicious sample

Field	Content
LLM Output	"This file has been flagged as suspicious, which means your computer might be at risk if you open or run it. It's like having a warning light on your dashboard — it's telling you to be careful and take action. Don't open or run this file under any circumstances. Instead, delete it immediately from your computer."

5.4.4 User-Level Explanations

The user-level explanations are designed for end users without technical knowledge, such as home users and non-technical office staff. They do not require a technical explanation or summary; instead, they would like to know what the security alert means for them and what action they should take. Table 5.10 shows the user-level output for the same sample.

In this explanation, technical terms are avoided completely, such as "entropy", "DLL characteristics", or "linker version". Instead, suggest a direction of action to take in a simple way that can be understandable for all users, even if they don't know much about malware.

5.4.5 Comparison of LLM Models

In this experiment, two LLM models were used, named deepseek-r1:1.5b and llama3.2:3b. Both models are able to generate explanations for all three stakeholders. However, there are some noticeable differences, for example, deepseek-r1:1.5b model, which is small, but it generates long explanations with the thinking process, which is not required in the final explanation. On the other hand, llama3.2:3b generated a more concise and brief explanation, which is to the point and better for manager-level staff. Table 5.11 outlines the standard qualitative characteristics of the two LLMs across the three explanation levels.

Table 5.11: Qualitative comparison of deepseek-r1:1.5b and llama3.2:3b across explanation levels

Characteristic	**Level**	**deepseek-r1:1.5b**	**llama3.2:3b**
Output length	Analyst	Long (detailed)	Moderate
Technical accuracy	Analyst	Good	Good
Conciseness	Manager	Moderate	High
Structured format	Manager	Variable	Consistent
Plain language	User	Good	Excellent
Response time	All	∼55 s (CPU-only)	∼47 s (CPU-only)
Model size	–	1.1 GB	2.0 GB

Here, slightly better explanations were produced by llama3.2:3b, which is quite useful for managers and end users because this explanation is short and easy to read and understand. But both models are good enough to use as they run locally and don't require API calling.

5.5 Explainability Metrics

Beyond classification accuracy, the system was evaluated for XAI methods using two main metrics: local fidelity and runtime overhead. Both metrics were measured

empirically on individual samples, and time was recorded for each sample separately. The predictions of the LIME surrogate model were compared with the XGBoost model's decisions.

5.5.1 Fidelity

Local fidelity is used to measure how closely the simple surrogate model matches the actual prediction that was made by the complex model. In other words, it checks whether the explanation matches the decision made by the original model, which is highly important in a security context. For example, if the explanation does not reflect the true logic of the model, then later explanations generated by the LLM could mislead the analyst.

LIME's local fidelity was evaluated using 200 held-out test samples, which were taken from the balanced EMBER subset. For each sample, the predicted class from the LIME surrogate model was compared with the predicted class from the XGBoost model. It was observed that the surrogate model agreed with the XGBoost model with `100.0%` accuracy.

This result shows that the feature importance values provided to the LLM truly reflect the local decision of the classifier. It also indicates that the linear approximation used by LIME is consistent with the decisions made by XGBoost for this dataset.

On the other hand, TreeSHAP is designed in such a way that it satisfies the exact additivity for tree-based models [27]. For each sample, the SHAP values add up to the difference between the model prediction and the overall expected value. Because of this, the local fidelity of TreeSHAP is considered theoretically perfect, and it provides reliable attribution for both local and global explanations.

To check this numerically, the additivity residual $|\sum_i \phi_i + \mathbb{E}[f(x)] - f(x)|$ was calculated for all 200 test samples. It was observed that the mean residual was

1.22×10^{-6} and the maximum residual was 4.77×10^{-6}. These very small values confirm that TreeSHAP maintains the exact additivity within the floating-point precision for all samples.

5.5.2 Runtime

Runtime performance was evaluated to assess whether the model can be used in real-world situations. In cybersecurity, speed is crucial as threats can spread quickly. The average values were calculated over 50 individual samples, and the experiment was carried out on a local machine.

On average, classification with XGBoost required 268.5 ms per sample. LIME explanation generation required about 0.65 seconds per sample when 1,000 perturbations were used for each call. TreeSHAP generated explanations in around 0.23 seconds per sample. From these results, it can be seen that TreeSHAP is much faster compared to the LIME method.

For the LLM part, the explanation generation time was checked for all three prompt levels, which include analyst, manager, and end-user. For each call, the time was recorded and then averaged, which is already shown in Table 5.11. The deepseek-r1:1.5b model took around 55 seconds per call on average, and the llama3.2:3b model took around 47 seconds per call. These times are calculated on a local machine that has a CPU without a dedicated GPU, which is why the generation time is higher. In a real production environment where GPU hardware is available, the inference time can be reduced to a few seconds per call. If the LLM time is combined with the XAI processing time, the total time to a readable explanation was about 47.6 seconds per file on the local test machine. In a GPU-based setup, this time would be much lower.

5.6 Human-Centered Evaluation

In addition to the technical metrics, a qualitative evaluation of the generated explanations was conducted. The goal was to understand whether the explanations are useful and trustworthy for the three stakeholder groups. Since a formal user study is outside the scope of this research, the evaluation focuses on the explanations produced by the framework. It also examines how system design helps to meet the information needs of each stakeholder [28].

5.6.1 Trust

Trust refers to the degree of trust that a user has in the output of a system. In cybersecurity, predictions from unexplained "black-box" models are often difficult for analysts to trust. When a model only gives a prediction without explaining the reason, security analysts may hesitate to rely on it [29].

In this framework, trust is improved by using a *feature knowledge base* that is built into script `06_llm_explainability_layer.ipynb`. Before sending the explanation to the LLM, each XAI feature score is linked to a short domain-related description, which helps translate the technical feature values into meaningful cybersecurity information.

For example, a `section_entropy_mean` value is explained as high code randomness, which may indicate encryption or packing. In the same way, an unusual and suspicious `compilation_timestamp` can be described as a possible sign of timestamp spoofing. These descriptions make the explanation easier to understand because they relate the features to common cybersecurity concepts instead of only showing numeric importance values.

As shown in the analyst-level example in Table 5.8, the LLM also identified that `has_debug_info` value of zero and a `dll_characteristics` value of zero could indicate possible evasion behavior. An experienced analyst can quickly compare

these findings with their own knowledge, which helps increase confidence in the system's output.

5.6.2 Quantitative Trust Metrics

In addition to qualitative design analysis, two measurable metrics were calculated across all 96 saved explanation files to support the trust evaluation. These metrics are the Feature Coverage Rate (FCR) and the Flesch–Kincaid Grade Level (FKGL).

Feature Coverage Rate measures how well the LLM explanation aligns with the XAI top features specified in the prompt. It is defined as the proportion of the top-5 feature names from the prompt that also appear in the generated explanation. Checking whether an explanation is based on the actual model evidence is an important evaluation criterion in XAI research [30]. A high FCR means the LLM discusses the features that influenced the prediction instead of generating generic text. In this study, FCR is used as a simple proxy for faithfulness that can be calculated without human evaluation

Flesch–Kincaid Grade Level is a commonly used readability formula [31] that is also widely used as an evaluation metric in NLP text generation research [32]. It measures how difficult a text is to read by assigning it a U.S. school grade level. A lower score indicates simpler language. In this study, FKGL is used to check whether the three-level prompt design produces language suitable for each stakeholder group, such as technical language for analysts, standard professional language for managers, and simpler language for end users.

Both metrics were calculated from 96 explanation files (32 per level) using the TextStat Python library. These explanations were generated by two LLMs (deepseek-r1:1.5b and llama3.2:3b) using two classification models (XGBoost and MLP) and two XAI methods (LIME and SHAP). The results are presented in Table 5.12.

Table 5.12: Quantitative trust metrics computed over all 96 saved explanation files

Level	**LLM**	**FCR (%)**	**FKGL**
Analyst	deepseek-r1:1.5b	72.5	9.38
Analyst	llama3.2:3b	30.0	12.81
Analyst (combined)	—	**51.2**	**11.10**
Manager	deepseek-r1:1.5b	0.0	10.56
Manager	llama3.2:3b	0.0	14.93
Manager (combined)	—	**0.0**	**12.74**
User	deepseek-r1:1.5b	0.0	6.61
User	llama3.2:3b	0.0	9.63
User (combined)	—	**0.0**	**8.12**

Table 5.13: Standard US school grade mappings and corresponding reading levels for Flesch–Kincaid Grade Level scores (derived from the original formula [31])

FKGL Score	**US School Grade**	**Reading Level**
6.0 – 8.0	6th – 8th grade	Middle School (Easy, conversational plain English)
9.0 – 12.0	9th – 12th grade	High School (Standard reading)
13.0 – 16.0	College / University	Undergraduate (Academic, technical, difficult to read)

To interpret these scores, Table 5.13 outlines the standard US school grade mappings corresponding to the formula defined by Kincaid et al. [31]. As shown in Table 5.12, the results show a clear and meaningful pattern in the FKGL scores, as analyst-level explanations had an average grade level of 11.10, which corresponds to secondary school or early university reading. User-level explanations had a grade level of 8.12, which is closer to middle school reading. This difference shows that the three-level prompt design successfully adjusted the language complexity for each intended audience.

For the FCR metric, only analyst-level explanations included feature names, with a score of 51.2%. Manager and user explanations scored 0% because the LLMs

converted technical features into plain language instead of feature names. Together, these two metrics show that the generated explanations are based on the actual XAI evidence and are written in a way that is easy to read for each target audience.

5.6.3 Usefulness

Usefulness means how practical and helpful the generated explanations are for a specific task. In this framework, the three-level prompt design helps achieve this goal. For *analysts*, the prompt keeps the technical feature values and importance scores, which allows the generated explanation to include clear indicators that can help to guide a security investigation.

For *managers*, the prompt produces a more structured output. It includes clear fields such as Risk Level (HIGH, MEDIUM, or LOW) and Recommended Action (BLOCK, INVESTIGATE, or ALLOW). Technical details such as entropy values or data-directory information are removed because they are not needed for quick decision-making.

For *end-users*, the prompt avoids technical terms completely. As shown in the Table 5.10, the explanation uses simple language and easy comparisons, such as describing the warning, and also gives clear action for the user to follow. By using a different explanation format for each group, the framework helps convert technical XAI feature scores into clear and practical guidance that each stakeholder can understand and use.

6 Results and Discussion

In this chapter, a discussion was carried out about the experimental results in Chapter 5 in relation to the research objectives that were introduced in Chapter 1. The main objective is to discuss the result with the research questions. In addition, also discussed the importance of the result for a cybersecurity analyst, the performance, and limitations of the framework.

6.1 Model Performance Evaluation

6.1.1 Interpreting the Classification Results

The result shows the clear dominance of the XGBoost model, which achieved the best accuracy, which is 97% accuracy, and an ROC-AUC of 0.997, which shows perfect separation between malicious and benign files. MLP and 1D-CNN performed in the middle range with an accuracy of around 91%. The Autoencoder performed the worst with 48.3% accuracy, which is close to random guessing. The strong performance of XGBoost is expected as gradient-boosted tree models are known to perform very well on structured tabular data, and PE malware features fall into this category [11].

From a security perspective, recall is a very important metric because it indicates when malware was missed by the system, which could be very dangerous. With XGBoost 0.980 and Random Forest 0.967 recall shows that very few malicious files

were missed. Even Logistic Regression achieved a 0.893 recall value. This shows that features extracted from the EMBER dataset have a clear pattern related to the malware.

6.1.2 Deep Learning Versus Traditional Machine Learning

Deep Learning models are generally considered more powerful, but in this experiment, XGBoost performs better than MLP and 1D-CNN. One reason is that deep learning models work better on unstructured data such as raw text, images, and audio, where models learn patterns automatically. While in this experiment, data is well structured in tabular form, and tree-based models like XGBoost perform well on this type of data because they handle numerical features and nonlinear relationships more effectively [20].

The 1D-CNN model works differently, which treats the feature vector as a one-dimensional sequence and applies convolution filters, and assumes that nearby features may contain related patterns. But the EMBER features are not arranged in a natural sequence, so this does not work. Despite this limitation, the model achieves 90.3% accuracy, which is comparable to the MLP model, indicating that it effectively captures useful patterns. On the other hand, the Autoencoder was trained only on benign samples and used reconstruction errors to detect malware. But this approach performs poorly with only 8.7% recall, which means that malware samples are incorrectly classified as benign. This suggests that anomaly detection methods do not work well with an interpretable features dataset, which makes it harder to distinguish malware from benign.

6.2 Quantitative Results Evaluation

In this experiment, the dataset was balanced with an even split of malicious and benign files, so accuracy is a reliable indicator and straightforward to understand. However, F1-Score and ROC-AUC are more powerful for classification because they consider the balance between precision and recall. ROC-AUC measures how well a model can distinguish between the two classes without relying on a fixed decision threshold. One reason that the model performs well is because of high-quality feature extraction from the EMBER dataset, where 618 interpretable features describing different parts of the PE file provide a comprehensive view of each sample to the classifier.

The XAI analysis shows that important features that are identified by the LIME and SHAP methods, including `compilation_timestamp`, byte entropy values, and PE data directory fields, are the same features that are usually examined by the malware analyst, which shows that the feature extraction pipeline was well designed. In addition, `compilation_timestamp` was ranked as the most important feature by SHAP across all five supervised learning models used in this experiment. High byte entropy features were also identified as important features by both LIME and SHAP, which suggests the presence of compressed or encrypted regions that are commonly found in packed malware [33].

6.3 LLM Enhanced Explanation

6.3.1 Quality of Generated Explanations

The addition of an LLM explainability layer is one of the key contributions of this research. The explanations generated by llama3.2:3b and deepseek-r1:1.5b are clear, understandable, and adapted well to the three different stakeholders. At the analyst

level, the explanations included suspicious PE and connected them to known malware indicators. For example, when LIME identifies linker_version_minor: 50.00 as an important feature, it indicates that the LLM should explain that this value is unusual and that obfuscation could have occurred. The LLM may also suggest further investigation, such as memory analysis or sandbox testing. At the manager level, the explanations are short and focused on the possible impact on the system without using any technical terms. It includes a summary of the threat, the risk level (HIGH, MEDIUM, or LOW), and the recommended action, such as BLOCK, INVESTIGATE, or ALLOW. At the user level, the explanations are simpler by providing easy comparison and clear instructions. Users are clearly instructed not to open the file and delete it immediately, which makes it easy to understand for non-technical users.

6.3.2 LIME Versus SHAP as Input to the LLM

In the LLM-generated explanations pipeline, both LIME and SHAP results are given as input to the LLM, and the LLM produces explanations of similar quality if the top features and their value are clearly provided. However, there are slight differences as both methods identified top features, SHAP, which is based on Shapley values that provide a more stable global reference, and LIME focuses on local values that could vary in a random perturbation process.

In this experiment, there is a practical difference noted between the two XAI methods. SHAP generates a single explanation in about 0.23 seconds, while LIME takes around 0.65 seconds because of its perturbation process, making SHAP roughly three times faster per sample. However, SHAP global analysis took a long time over more samples than LIME, which is usually applied to only a few selected samples. Both of them have their own use case; for example, LIME is suitable for near-real-time explanations when only local explanations are needed, and SHAP is suitable

for offline analysis where global patterns across many files need to be studied.

6.3.3 deepseek-r1:1.5b Versus llama3.2:3b

These two LLMs show slightly different styles, which is important when deciding how to deploy the system. The deepseek-r1:1.5b is a reasoning model, even though it is relatively small. Because of this, it includes visible thinking steps in its output, such as "Let me think about this". This behavior is common in reasoning-based models, especially when they are asked to follow strict output formats. The llama3.2:3b model is a general-purpose instruction model. It follows structured templates more consistently and produces much cleaner outputs, especially for the manager-level and user-level explanations.

For analyst-level explanations, both models provide similar technical information. However, for manager and user explanations, llama3.2:3b works better because the results are clearer and more structured. For a production deployment, it is recommended to use llama3.2:3b as the primary model, with deepseek-r1:1.5b as the optional model for step-by-step reasoning. Both models run locally without external APIs, which helps to meet strict privacy and data protection requirements in security environments.

6.4 Comparison of Results

Table 6.1 shows how this study compares with previous work on malware detection. Overall, the proposed approach performs competitively, reaching an accuracy of 97.0% using XGBoost on a subset of the EMBER dataset.

Some earlier studies report slightly higher accuracy. For example, Anderson and Roth (2018) achieved 99.3% accuracy with LightGBM on the full EMBER dataset, but their approach does not include explainability features. Similarly, Raff

et al. (2017) obtained 94% accuracy using a CNN-based model, but without any interpretability support.

In contrast, this work not only focuses on classification performance but also adds explainability through LIME, SHAP, and LLM-based explanations. So, even though the accuracy is a bit lower than the best reported results, the main advantage of this approach is that it provides both strong performance and clear interpretability, which is important for real-world cybersecurity use.

Table 6.1: Comparison of this work's classification results with selected related works on PE malware detection.

Reference	**Dataset**	**Best Model**	**Accuracy**	**XAI / LLM**
Anderson & Roth (2018) [3]	EMBER	LightGBM	99.3%	None
Raff et al. (2017) [33]	Proprietary (2M)	MalConv (CNN)	94.0%	None
This work	EMBER (subset)	XGBoost	97.0%	LIME + SHAP + LLM

6.4.1 Alignment with Research Objectives

There were four primary research objectives stated in Chapter 1: (1) evaluating the performance of ML and DL models for malware classification; (2) improving interpretability using XAI techniques; (3) investigating how LLMs can change raw XAI outputs into human-centered explanations; and (4) evaluating the benefits of this combination for trust, usability, and decision making. The results from Chapters 5 and 6 clearly show that all four goals and their corresponding research questions have been achieved successfully.

Objective 1 was achieved by training and evaluating multiple models, with XGBoost performing best at 97.0% accuracy and 0.997 ROC-AUC. Objective 2 and

Research Question 1 (RQ1) were addressed using LIME and SHAP, which highlighted `compilation_timestamp` and byte entropy as the most important features. Objective 3 and Research Question 2 (RQ2) were fulfilled by the three-level LLM explanation pipeline, that provide clear and understandable explanation for analysts, managers, and end-users. Finally, Objective 4 and Research Question 3 (RQ3) were successfully addressed by evaluating the readability of the LLM outputs using FKGL scores, which shows that the explanations are easy to understand and thus increase trust and usability in real-world decision-making.

6.5 Discussion of findings and implications

One important finding from this research is that explainability for malware classification is practically feasible. The whole system, from classification, XAI analysis, to LLM explanation, runs locally using open-source tools, without needing cloud services or API keys. These small models used in this experiment are available online and run on a local machine with the help of Ollama without needing a GPU. Because of this, the framework can also be used by small and medium-sized organizations.

Another important benefit of this framework is the three-level explanations. In most of the security systems, alerts usually show only the rule ID or a confidence score of the model. However, this framework improved the communication gap by producing three different explanations for different stakeholders, which are tailored to their requirements. Security analysts receive technical details, managers receive short summaries with suggested actions, and end users see a simple warning message with recommended action.

From the explainability perspective, it is evident that LIME and SHAP produced meaningful and strong results. Both methods identified that `compilation_timestamp` is an important and top feature. Because of this, it can be concluded that the feature extraction process and XAI methods are reliable enough

to support security analysis and deployment.

Finally, the use of an LLM layer raises another research concern: how should the quality of generated explanations from LLMs be evaluated? In this study, the evaluation was mainly qualitative, and better metrics are still needed for evaluation. These metrics should check whether the generated explanations reflect the important features correctly and do not hallucinate the information. Developing better evaluation methods for LLM-generated explanations in cybersecurity remains an important area of research.

7 Conclusion and Future Work

This thesis aims to address the gap in how AI is used in cybersecurity. The malware detection system operates like a "black box"; it produces results but does not explain why a file is flagged as malicious or benign. For that reason, a security analyst cannot trust the system, and the result is useless to non-technical users. To address this, a three-layer framework was proposed that combines a classification layer, an XAI layer using LIME and SHAP, and an LLM layer to generate natural-language explanations. The explanations are generated for three stakeholders: analysts, managers, and end users.

7.1 Addressing the Research Questions

This section revisits the three research questions established at the beginning of the study and provides answers based on the experimental findings.

RQ1: Which XAI technique provides the most interpretable and useful explanations for malware classification? Both LIME and SHAP were evaluated to determine which explanation method was more suitable for this problem. The results showed that SHAP provided a more stable and consistent understanding of the behavior of the model. For example, it repeatedly identified `compilation_timestamp` as one of the most important features in different models. Although LIME was faster and useful for quick local explanations, SHAP's

stronger mathematical foundation made its feature importance scores more reliable and trustworthy for analysts. As a result, SHAP was shown to be the most effective approach to detailed technical analysis in this study.

RQ2: How can LLMs enhance the interpretability of XAI outputs, making them useful for security analysts and understandable for managers? One of the biggest issues with raw XAI outputs, such as SHAP values, is that they are mostly just numbers and technical details, which can be difficult for many people to understand. LLMs help by turning these outputs into clear and readable explanations. For example, when explaining results to a security analyst, the LLM can connect the values to possible malware behavior, such as explaining that an unusual linker version may indicate that the file has been obfuscated. However, for managers or non-technical users, the LLM avoids complex technical terms and instead provides a simple summary of the risk along with a recommended action, such as investigating or blocking the file. This makes the explanations much more practical and easier to understand for users with different levels of technical knowledge.

RQ3: Does the integration of XAI and LLMs improve trust and usability in AI-driven malware detection? The findings strongly suggest that it does. One of the main challenges in using AI for malware detection is the "black box" problem, where users do not know why the system flagged a file as malicious. By combining XAI and LLMs, the system can explain its decisions in a clear and easy-to-understand way. When users, whether security analysts or regular users, can read a simple explanation of why a file was flagged, they are more likely to trust the system's decision. In addition, the structured explanations help users understand the situation more quickly and decide what action to take next, which improves the overall usability of the system.

7.2 Contribution of Research

Several contributions were made to the fields of cybersecurity and explainable AI through this research.

A complete three-layer framework A complete explainability pipeline was developed and implemented for Windows PE malware classification. For this research, the EMBER 2018 dataset was used, with data available in JSONL format per file. The framework takes this file as input and goes through from classification to natural language explanation generation. These explanations are for three audience levels that include the technical analyst, security manager, and general user. The complete system runs locally without the requirement of the internet or an API call.

Evaluation of six classification models Six models were used in this experiment that trained on 618 static features extracted from the EMBER dataset. These models were Logistic Regression, Random Forests, XGBoost, MLP, 1D-CNN, and an Autoencoder. From these models, XGBoost was found to be the best model with an accuracy of 97% and an ROC-AUC of 0.997. The Autoencoder does not perform well as it was trained only on benign samples. These results are useful as a reference for researchers to choose between model types for static PE malware analysis.

Cross-method agreement on feature importance Two explainability models, LIME and SHAP, were used in this experiment, and found that `compilation_timestamp` is the most important feature across all models and both XAI methods. The agreement between these two XAI methods gives confidence that this feature is extremely important in distinguishing. Another important feature found is byte entropy in high-value bins, which shows a sign of packed or encrypted malware.

Local LLM for security explainability In this experiment, small and locally hosted language models were used, which are llama3.2:3b and deepseek-r1:1.5b, that produce audience-appropriate explanations. Since both of these models are not that big and run successfully on CPU machines, without the requirement of a GPU. It means that the commercial LLM API is not required and the system can be implemented in privacy-sensitive environments at very low cost.

A modular and generalizable design The framework was designed so that each layer can be updated or replaced independently. As discussed in Chapter 4, the feature extraction component can be replaced to support Android APK malware or IoT firmware malware, without minor changes to the XAI or LLM layers. This makes the framework broadly applicable beyond just Windows PE files.

7.3 Practical Implications for Cybersecurity

Several practical implications are drawn from this research for organizations that use automated malware detection.

Bridging the communication gap in SOCs One of the major problems in security operations centers (SOCs) is that automated alerts are often impossible to understand for non-technical users. Analysts frequently receive alerts with little context, which leads to alert fatigue. The three-level explanation system in this framework solves this problem as it produces explanations for each type of user, without requiring a technical person to interpret the alerts.

Supporting analyst trust and decision-making Explainability is not only about communication, but also about trust, which helps decision-making. When the file is flagged as malicious, a security analyst needs to know the reason, and for this purpose, LIME and SHAP provide detailed feature-level explanations. For

example, if the top features are highlighted by explainability methods that match known malware patterns, then the analyst can trust the alert and make decisions. If the features seem unusual, the analyst must further investigate before taking any action.

Privacy and low-cost deployment Because the whole system runs locally, sensitive file data is never sent to external service providers. This is particularly important for organizations that operate under GDPR or other strict data protection laws. There is also no licensing cost involved because only open-source tools are used, which makes the system affordable and easy to scale.

7.4 Limitations of the Study

Several limitations have been identified in the study:

Small dataset and time limitation This experiment was performed on a balanced subset of 2,000 samples from the EMBER 2018 dataset. The reason for choosing a small subset is that the experiment can be performed on a local system and within time limits. However, it is a small subset of the dataset, but the results for a balanced dataset look better. In real deployment, where benign files outnumbered malicious files, the result could be different. Also, the EMBER 2018 dataset is several years old, which may not represent well for new evasion techniques. For that purpose, models need to train on newer versions of the dataset.

Static analysis only Another limitation of this study is that the framework is completely working on static malware analysis, where static features are extracted. However, static analysis is fast and safe, but it could be bypassed by packing, encryption, and polymorphic code. In modern malware, there is some form of obfuscation that can reduce the effectiveness of this static analysis.

Qualitative LLM evaluation The quality of explanations generated by the LLM was evaluated by the researcher. In this investigation, no formal study or quantitative evaluation metric was used. However, the explanations appear clear and well aligned with the model outputs, but there could be a possibility of inaccuracies that a formal user evaluation could identify. In addition, hallucination is one of the common limitations of large language models that also needs to be addressed.

No fine-tuning and runtime limitations Both LLMs were used with zero-shot learning, without fine-tuning with domain-specific knowledge. These general-purpose LLMs with a small number of parameters might not contain a deep understanding of the cybersecurity domain. Because of this, some explanations might not provide highly specialized explanations at the analyst level. In addition, the average generation time for explanation was about 55 seconds per request on a CPU-only machine, which could be reduced greatly with GPU-enabled hardware. The current sequential pipeline would require further optimization to support large-scale or high-volume production environments.

Table 7.1 summarizes each limitation alongside its corresponding future research direction.

Table 7.1: Summary of study limitations and corresponding future research directions

Limitation	**Proposed Future Direction**
Small, time-bounded dataset (EMBER 2018, 2,000 samples)	Train on the full EMBER dataset, and also train on the latest EMBER dataset, which is EMBER-2024
Static analysis only; vulnerable to packing and obfuscation	Add dynamic features from a sandboxed environment (e.g., Cuckoo Sandbox)
LLM evaluation was qualitative only; no user study	Conduct a formal user study and develop automated explanation quality metrics
No domain-specific LLM fine-tuning	Fine-tune on cybersecurity text or apply Retrieval-Augmented Generation (RAG)

7.5 Future Research and Directions

Based on the limitations identified, several directions are proposed for future work.

Larger and more diverse datasets The most immediate future extension of this research is to train the system on the full EMBER-2018 dataset and evaluate it on new samples from different sources. In addition, the system also implements the newer version of the EMBER dataset, which is 2024. Cross-dataset evaluation also provides a more realistic understanding of how well the system generalizes to new data.

Hybrid static and dynamic analysis Another future extension of this research is to implement a hybrid approach. Dynamic analysis features collected from a sandbox environment could be added to expand the current feature set. The only changes that need to be made are in the feature extraction layer, while other layers could remain the same or need minor changes. However, there is an interesting question of whether features like `compilation_timestamp` still play a major role, or dynamic features become more important.

LLM fine-tuning and RAG Fine-tuning the large language models on a large collection of cybersecurity corpus such as threat intelligence reports, MITR ATT&CK descriptions, and CISA advisories, could improve the accuracy of technical-level explanations. Another approach that could be used is the RAG (Retrieval-Augmented Generation), which can avoid the full retraining of the model. With RAG, relevant documentation is retrieved at inference time and provided to the LLM as additional context. This allows the system to stay updated with the current threat intelligence without requiring retraining of the model.

Formal user study and automated metrics Another improvement would be to conduct a structured user study involving all three stakeholder groups (analysts, managers, and general users) to provide evidence that these explanations improve understanding, accelerate decision-making, and build trust compared with traditional alerts. In addition, there is a need to develop automated metrics to assess the quality of the generated explanations. General NLP metrics such as BLEU or ROUGE are not suitable here because there is no single correct explanation.

Extension to Android and IoT malware As discussed in Chapter 4, the framework could be extended to support malware in other platforms, such as Android and IoT. However, these extensions still need to be implemented and tested to ensure that the framework generalizes well compared to other platforms. For this purpose, datasets such as MaMaDroid and DREBIN can be used for Android malware analysis, and for IoT malware analysis, the large-scale feature extraction approach can be used that is proposed by Costin et al. [25].

References

[1] Antti Hakkala and Seppo Virtanen. "Refining Engineering MSc Theses with a Focus Enhancing Structure Model". In: *Proceedings of the 15th International CDIO Conference*. June 25–27. Aarhus University, Aarhus, Denmark, June 2019, pp. 438–446.

[2] Mohd Saqib, Samaneh Mahdavifar, Benjamin C. M. Fung, and Philippe Charland. "A Comprehensive Analysis of Explainable AI for Malware Hunting". In: *ACM Computing Surveys* 56.12 (Dec. 2024), pp. 1–40. DOI: 10.1145/3677374.

[3] Hyrum S. Anderson and Phil Roth. "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models". In: *arXiv:1804.04637 [cs.CR]* (Apr. 2018). preprint, pp. 1–8. DOI: 10.48550/arXiv.1804.04637.

[4] Rebet Keith Jones. "Malware Analysis and Classification Using Deep Learning". In: *Revolutionizing Cybersecurity With Deep Learning and Large Language Models*. Ed. by Hewa Zangana, Jamal Al-Karaki, and Marwan Omar. Hershey, PA, USA: IGI Global, 2025, pp. 165–200. DOI: 10.4018/979-8-3373-3296-3.ch006.

[5] Ismayil Hasanov, Seppo Virtanen, Antti Hakkala, and Jouni Isoaho. "Application of Large Language Models in Cybersecurity: A Systematic Literature Review". In: *IEEE Access* 12 (2024), pp. 176751–176778. DOI: 10.1109/ACCESS.2024.3505983.

[6] Zhibo Zhang, Hussam Al Hamadi, Ernesto Damiani, Chan Yeob Yeun, and Fatma Taher. "Explainable Artificial Intelligence Applications in Cyber Security: State-of-the-Art in Research". In: *IEEE Access* 10 (2022), pp. 93104–93139. DOI: 10.1109/ACCESS.2022.3204051.

[7] Nicola Capuano, Giuseppe Fenza, Vincenzo Loia, and Claudio Stanzione. "Explainable Artificial Intelligence in CyberSecurity: A Survey". In: *IEEE Access* 10 (2022), pp. 93575–93600. DOI: 10.1109/ACCESS.2022.3204171.

[8] Richa Dasila, Vatsala Upadhyay, Samo Bobek, and Abhishek Vaish. "A Novel Study on Intelligent Methods and Explainable AI for Dynamic Malware Analysis". In: *arXiv:2508.10652 [cs.CR]* (Aug. 2025). preprint, pp. 1–34. DOI: 10.48550/arXiv.2508.10652.

[9] Harikha Manthena, Shaghayegh Shajarian, Jeffrey C. Kimmell, Mahmoud Abdelsalam, Sajad Khorsandroo, and Maanak Gupta. "Explainable Artificial Intelligence (XAI) for Malware Analysis: A Survey of Techniques, Applications, and Open Challenges". In: *IEEE Access* 13 (2025), pp. 61611–61640. DOI: 10.1109/ACCESS.2025.3555926.

[10] E. Baghirov. "A Comprehensive Investigation into Robust Malware Detection with Explainable AI". In: *Cyber Security and Applications* 3 (Dec. 2025), p. 100072. DOI: 10.1016/j.csa.2024.100072.

[11] Aaron B. Daniel and N. D. Patel. "Detection of Malware on Portable Executable Files Using Machine Learning". In: *Proceedings of the 2025 International Conference on Networks & Advances in Computational Technologies (NetACT)*. Thiruvananthapuram, India: IEEE, Aug. 2025, pp. 1–6. DOI: 10.1109/NetACT65906.2025.11187442.

[12] Farida Siddiqi Prity, Md. Shahidul Islam, Emran Hossain Fahim, Md. Maruf Hossain, Sazzad Hossain Bhuiyan, Md. Ariful Islam, and Mirza Raquib. "Ma-

chine Learning-Based Cyber Threat Detection: An Approach to Malware Detection and Security with Explainable AI Insights". In: *Human-Intelligent Systems Integration* 6.1 (Dec. 2024), pp. 61–90. DOI: 10.1007/s42454-024-00055-7.

[13] Ahsan Bilal, David Ebert, and Beiyu Lin. "LLMs for Explainable AI: A Comprehensive Survey". In: *arXiv preprint arXiv:2504.00125* (Mar. 2025). DOI: 10.48550/arXiv.2504.00125.

[14] Ahmed M. Salih, Zahra Raisi-Estabragh, Ilaria Boscolo Galazzo, Petia Radeva, Steffen E. Petersen, Karim Lekadir, and Gloria Menegaz. "A Perspective on Explainable Artificial Intelligence Methods: SHAP and LIME". In: *Advanced Intelligent Systems* 7.1 (2025), p. 2400304. DOI: 10.1002/aisy.202400304.

[15] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?" Explaining the Predictions of Any Classifier". In: *Proceedings of the Demonstrations Session of NAACL-HLT 2016 - 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, CA, USA, 2016, pp. 97–101. DOI: 10.18653/v1/n16-3020.

[16] Hung T. T. Nguyen, Quoc C. Hung, Khanh V. T. Nguyen, and D. K. P. Nguyen. "Evaluation of Explainable Artificial Intelligence: SHAP, LIME, and CAM". In: *Proceedings of the FPT AI Conference (FAIC 2021)*. Ha Noi, Vietnam, May 2021, pp. 1–6.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008. DOI: 10.48550/arXiv.1706.03762.

[18] Hamed Jelodar, Samita Bai, Parisa Hamedi, Hesamodin Mohammadian, Roozbeh Razavi-Far, and Ali Ghorbani. "Large Language Model (LLM) for Software Security: Code Analysis, Malware Analysis, Reverse Engineering". In: *arXiv:2504.07137 [cs.CR]* (Apr. 2025). preprint, pp. 1–15. DOI: 10.48550/arXiv.2504.07137.

[19] Dorcas Esther. *Explainable AI in Malware Analysis and Detection*. May 2023. URL: https://www.researchgate.net/publication/387225235_Explainable_AI_in_Malware_Analysis_and_Detection.

[20] Aditya Choudhary, Sarthak Pawar, and Yashodhara Haribhakta. "Efficient Malware Detection with Optimized Learning on High-Dimensional Features". In: *arXiv:2506.17309 [cs.CR]* (June 2025). preprint, pp. 1–8. DOI: 10.48550/arXiv.2506.17309.

[21] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. "MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models". In: *Proceedings of the 24th Annual Network and Distributed System Security Symposium (NDSS 2017)*. San Diego, CA, USA: Internet Society, Feb. 2017, pp. 1–15. DOI: 10.14722/ndss.2017.23353.

[22] Yue Pan, Xin Ge, Chunrong Fang, and Yutian Fan. "A Systematic Literature Review of Android Malware Detection Using Static Analysis". In: *IEEE Access* 8 (2020), pp. 116363–116379. DOI: 10.1109/ACCESS.2020.3002842.

[23] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket". In: *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS 2014)*. San Diego, CA, USA: Internet Society, Feb. 2014, pp. 23–37. DOI: 10.14722/ndss.2014.23247.

[24] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. "DroidChameleon: Evaluating Android Anti-malware against Transformation Attacks". In: *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '13)*. Hangzhou, China: ACM, May 2013, pp. 329–334. DOI: 10.1145/2484313.2484355.

[25] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. "A Large-Scale Analysis of the Security of Embedded Firmwares". In: *Proceedings of the 23rd USENIX Security Symposium (USENIX Security '14)*. San Diego, CA, USA: USENIX Association, Aug. 2014, pp. 95–110.

[26] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. "Understanding the Mirai Botnet". In: *Proceedings of the 26th USENIX Security Symposium (USENIX Security '17)*. Vancouver, BC, Canada: USENIX Association, Aug. 2017, pp. 1093–1110.

[27] Scott M. Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems*. Vol. 30. Long Beach, CA, USA: Curran Associates, Inc., 2017, pp. 4765–4774.

[28] Zachary C. Lipton. "The Mythos of Model Interpretability". In: *Queue* 16.3 (2018), pp. 31–57. DOI: 10.1145/3236386.3241340.

[29] Cynthia Rudin. "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead". In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215. DOI: 10.1038/s42256-019-0048-x.

[30] Meike Nauta, Jan Trienes, Shreyasi Pathak, Elisa Nguyen, Michelle Peters, Yasmin Schmitt, Jörg Schlötterer, Maurice van Keulen, and Christin Seifert. "From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI". In: *ACM Computing Surveys* 55.13s (2023), pp. 1–42. DOI: 10.1145/3583558.

[31] J. Peter Kincaid, Robert Paul Fishburne, Richard L. Rogers, and Brad S. Chissom. *Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel*. Tech. rep. RBR 8-75. Millington, TN, USA: Chief of Naval Technical Training, 1975.

[32] Fernando Alva-Manchego, Louis Martin, Antoine Bordes, Carolina Scarton, Benoît Sagot, and Lucia Specia. "Data-Driven Sentence Simplification: Survey and Benchmark". In: *Computational Linguistics* 46.1 (2020), pp. 135–187. DOI: 10.1162/coli_a_00370.

[33] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. *Malware Detection by Eating a Whole EXE*. preprint. Oct. 2017. DOI: 10.48550/arXiv.1710.09435.

Appendix A Feature Knowledge Base Mapping

The following table shows the subset Feature Knowledge Base that maps technical features to human-readable knowledge.

Table A.1: Subset Feature Knowledge Base mapping technical features to human-readable knowledge

Technical Feature	**Human-Readable Name**	**Knowledge Base (Malicious Indicator)**
`file_size_bytes`	File Size	Extremely large or very small files can be suspicious
`section_entropy_mean`	Section Entropy (Avg)	High entropy (>6.5) suggests encryption, packing, or obfuscation
`has_digital_signature`	Digital Signature Present	Missing signature (0) is a major red flag
`num_imported_dlls`	Number of Imported DLLs	Very few (<3) or excessive (>50) imports are suspicious
`imports_ws2_32`	Imports WS2_32.dll	Network communication capability - common in malware
`imports_advapi32`	Imports ADVAPI32.dll	Often used for registry manipulation or privilege escalation
`timestamp`	Compilation Timestamp	Future dates or very old dates can be spoofed

Appendix B Complete LLM Prompt Templates

This appendix provides the exact, complete prompt templates used in Python scripts to generate the three levels of explanations. These templates demonstrate how context from the XAI layer was injected into the LLM.

Analyst-Level Prompt Template (Technical)

```
You are an expert cybersecurity analyst.
A machine learning model has classified a Windows PE file.

**Classification Result:**
- File Label: {label}
- Model: {model_name}
- Prediction Confidence: {confidence:.1%}
- XAI Method: {xai_method}

**Top Influential Features:**
{feature_list}

**Task:**
```

```
Provide a technical analysis explaining WHY this
file was classified as {label}.

Focus on:
1. The most significant technical indicators
2. Specific suspicious or benign behaviors identified
3. How these features relate to known malware/benign patterns
4. Recommended next steps for investigation

Keep your response concise (3-4 sentences) and technical.
```

Manager-Level Prompt Template (Executive Summary)

```
You are briefing a security operations manager about
a malware detection result.

**Detection Summary:**
- File Classification: {label}
- Confidence Level: {confidence:.1%}
- Analysis Model: {model_name}

**Key Risk Indicators:**
{simplified_features}

**Task:**
Provide an executive summary suitable for a
```

```
manager (non-technical).

Include:
1. High-level summary of the threat/safety assessment (1-2 sentences)
2. Risk level: HIGH, MEDIUM, or LOW
3. Recommended action: BLOCK, INVESTIGATE, or ALLOW

Use business-friendly language, not deep technical jargon.
```

User-Level Prompt Template (Non-Technical)

```
You are explaining a security alert to a non-technical computer user.

**File Status:** This file has been flagged as {label}

**Why this happened:**
{user_friendly_summary}

**Task:**
Explain in simple, non-technical language:
1. What this means for the user (1-2 sentences)
2. What the user should do RIGHT NOW

Use everyday language. Avoid technical terms
like "entropy", "DLL", or "malware signatures".
Keep it very simple and actionable.
```

Appendix C Source Code Repository

To ensure complete reproducibility and transparency of the research methodology, the entire codebase developed for this thesis has been made publicly available. The repository contains all Python scripts, Jupyter Notebooks, model training configurations, XAI extraction code, and LLM explanation generation pipelines utilized in this study.

The complete human-centered explainability framework and associated documentation can be accessed on GitHub at the following URL:

`https://github.com/gak92/LLM-XAI-Malware-Framework`