**MALWARE DETECTION AND FAMILY CLASSIFICATION WITH DYNAMIC FEATURES USING ML**

**A SOCIALLY RELEVANT MINI PROJECT REPORT**

***Submitted by***

**HARIKARAN K M 211423104199**

**IMMANUVEL V 211423104228**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

### COMPUTER SCIENCE AND ENGINEERING

****

**PANIMALAR ENGINEERING COLLEGE**

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**

**OCTOBER 2025**

**PANIMALAR ENGINEERING COLLEGE**

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**

**BONAFIDE CERTIFICATE**

Certified that this project report **“MALWARE DETECTION AND FAMILY CLASSIFICATION WITH DYNAMIC FEATURES USING ML”** is the Bonafide work of **HARIKARAN K M (211423104199), IMMANUVEL V (211420104228)** who carried out the project work under my supervision.

|  |  |
| --- | --- |
| **Signature of the HOD with date** | **Signature of the Supervisor with date** |
| **Dr L.JABASHEELA M.E.,Ph.D.,** | **P.ALWIN INFANT M.E.,Ph.D.,** |
| **Professor and Head,**  **Department of Computer Science and Engineering,**  **Panimalar Engineering College, Chennai- 123** | **Associate Professor,**  **Department of Computer Science and Engineering,**  **Panimalar Engineering College, Chennai- 123** |

Submitted for the 23CS1512 – Socially relevant mini-Project Viva– Voce

Examination held on ………………………….

**INTERNAL EXAMINER EXTERNAL EXAMINER**

## DECLARATION BY THE STUDENT

We **HARIKARAN K M [211423104199] and IMMANUVEL V [211420104228]** hereby declare that this project report titled **“Malware Detection and Family Classification with Dynamic Features using ML”,** under the guidance of **Alwin Infant M.E.,Ph.D.,** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

**HARIKARAN K M [211423104199]**

**IMMANUVEL V [211420104228]**

**ACKNOWLEDGEMENT**

We express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.,** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to extend our heartfelt and sincere thanks to our Directors **Tmt.C.VIJAYARAJESWARI, Dr.C.SAKTHIKUMAR, M.E., Ph.D., and Dr. SARANYASREE SAKTHIKUMAR B.E.,M.B.A.,Ph.D.,** for providing us

with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.,** for his timelyconcern and encouragement provided to us throughout the course.

We thank the HOD of CSE Department, **Dr.L.JABASHEELA, M.E., Ph.D.,** for the support extended throughout the project.

We would like to thank our Project Coordinators **Dr.P.DEEPA, M.E., Ph.D., Mrs.S .LINCY JEMINA, M.E., (Ph.D)** and our Project Guide **Dr.A.ALWIN INFANT, M.E., Ph.D.,** and all the faculty members of the Department of CSE for their advice and suggestions for the successful completion of the project.

Our sincere thanks to **Mrs.V.A.PRABHA, SCIENTIST-E, C-DAC, CHENNAI**, for this kind support. Finally, I would like to thank all those who were directly or indirectly helpful in carrying out this research.

**HARIKARAN K M (211423104199)**

**IMMANUVEL V (211420104228)**

**ABSTRACT**

The increasing complexity and sophistication of malware pose a significant threat to cybersecurity. Traditional detection approaches, such as static signature-based methods and heuristic disassembly, often fail against modern malware because attackers exploit techniques like obfuscation, encryption, and polymorphism to evade detection. This creates a pressing need for more adaptive and intelligent detection frameworks.

To address these challenges, this project introduces **MalwareClassifier**, a two-stage ensemble machine learning framework. In the first stage, a stacked ensemble of Logistic Regression, Random Forest, and XGBoost is employed to perform binary classification, distinguishing benign samples from malicious ones. This layered approach ensures greater robustness and accuracy by leveraging the complementary strengths of different classifiers.

In the second stage, malware samples identified in Stage 1 undergo fine-grained analysis through a family classification model. Here, ensemble methods, including Logistic Regression, Random Forest, and XGBoost, are combined to categorize malware into families such as Trojans, spyware, and ransomware. This hierarchical approach not only enhances detection performance but also generates actionable threat intelligence, which is crucial in cybersecurity operations.

Extensive preprocessing techniques such as label encoding, feature scaling, and missing value imputation are integrated into the pipeline, making it highly suitable for structured datasets like **CIC-MalMem-2022**. Experimental results demonstrate that the proposed system outperforms traditional static-analysis-based methods, achieving higher accuracy, better generalization, and resilience against evasion strategies. This highlights its potential for real-world malware detection and defense systems.

**TABLE OF CONTENTS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CHAPTER**  **NO** | | **TITLE** | | **PAGE NO.** | |
|  | | **ABSTRACT** | | vii | |
|  | | **LIST OF FIGURES** | | x | |
| **1.** | | **INTRODUCTION** | | **1** | |
|  | | 1.1 Overview | | 1 | |
|  | | 1.2 Problem Definition | | 2 | |
|  | | 1.3 Literature Survey | | 3 | |
| **2.** | | **SYSTEM ANALYSIS** | | **5** | |
|  | | 2.1 Existing System | | 5 | |
|  | | 2.2 Proposed System | | 5 | |
|  | | 2.3 Implementation Environment | | 6 | |
| **3.** | | **SYSTEM DESIGN** | | **7** | |
|  | | 3.1 UML Diagrams | | 7 | |
| **4.** | | **SYSTEM ARCHITECTURE** | | **10** | |
|  | | 4.1 Architecture Diagram | | 10 | |
|  | | 4.2 Module Design Specification | | 13 | |
| **5.** | | **SYSTEM IMPLEMENTATION** | | **19** | |
|  | | 5.1 Preprocessing Implementation | | 19 | |
|  | | 5.2 Stage 1: Binary Classification | |  | |
|  | | 5.3 Stage 2: Family Classification | |  | |
|  | | 5.4 Prediction Function | |  | |
|  | | 5.5 Backend cuckoo Sandbox Integration | |  | |
| **6.** | | PERFORMANCE ANALYSIS | | 27 | |
|  | | 6.1 Performance Metrics | | 27 | |
|  | | 6.2 Manual Test Cases | | 28 | |
|  | | 6.2 Result and Discussion | | 30 | |
| **7.** | | CONCLUSION | | 31 | |
|  | | 7.1 Conclusion | | 31 | |
|  | | 7.2 Future enhancement | | 32 | |
| **8.** | | APPENDICES | | 27 | |
|  | | A1 SDG goals | | 27 | |
|  | | A2 Sample Screenshots | | 34 | |
|  | | A3 Paper Publication | | 37 | |
|  | | A4 Plagiarism report | | 38 | |
| **9.** | | REFERENCES | | 33 | |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIG NO.** | **FIGURE DESCRIPTION** | **PAGE NO** |
| 3.1.1 | Activity Diagram for Malware classification | 7 |
| 3.1.2. | Deployment Diagram for Malware classification | 9 |
| 4.1.1 | Architecture Diagram for Malware classification | 10 |
| 6.1.1. | Flow Chart | 20 |
| 6.1.2. | Confusion Matrix | 22 |
| 6.1.3. | Confusion Matrix Heatmap | 23 |
| 8.1 | Screenshot of basic page | 34 |
| 8.2. | Screenshot of Result | 34 |
| 8.3. | Screenshot of Generated report | 35 |

# CHAPTER 1

**INTRODUCTION**

* 1. **OVERVIEW**

The cybersecurity landscape has become increasingly hostile with the rapid evolution of malware. Unlike early computer viruses that relied on simple replication and disruption, modern malware families are far more complex. They are engineered to achieve diverse objectives such as financial fraud, identity theft, ransomware extortion, and persistent access to compromised systems. This sophistication places enormous pressure on traditional detection mechanisms that cannot keep pace with the creativity and adaptability of attackers.

Traditional detection systems use signature-based methods to identify malware, but they struggle against new or obfuscated variants that exploit weaknesses through modifications or encryption. As static approaches become obsolete, machine learning (ML) offers resilience by learning from both benign and malicious samples. By leveraging static and dynamic features with ensemble methods, ML enhances detection, robustness, and scalability.

However, ML systems face challenges tied to dataset quality, preprocessing, and feature selection, which may cause bias, noise, or overfitting. Models that work well in controlled environments often fail on diverse real-world datasets. This highlights the need for scalable, modular, and interpretable solutions for effective malware detection.

* 1. **PROBLEM DEFINITION**

One of the most pressing issues in malware detection is the inability of traditional approaches to adapt to new and evolving threats. Static analysis, although fast and lightweight, is easily bypassed through code obfuscation and encryption. As a result, malware that exhibits dynamic behavior during runtime often goes undetected.

Early machine learning solutions attempted to address this issue by extracting static features from disassembled binaries. While this showed promise, it introduced limitations in scalability and dataset diversity. Models trained on small or controlled datasets failed to generalize effectively when tested against large-scale, real-world samples.

Furthermore, most traditional methods are limited to binary classification — separating malware from benign programs. They fail to provide family-levelclassification, which is crucial in cybersecurity for understanding attack strategies, attributing threats, and developing targeted defenses. Hence, there is a clear demand for an approach that performs both binary detection and family classification, while maintaining resilience against evasion strategies.

* 1. **LITERATURE REVIEW**

Ravin Thenmozhi M. [1] presented *“Malware Classification Using Machine Learning and Deep Learning: A Comprehensive Approach.”* To improve the accuracy over malware categorization, the study integrates deep learning structure with traditional machine learning technique and algorithms. Despite that deep learning models performed better, they have suggested that combining them with machine learning techniques produced a more balanced system in terms of precision and flexibility

Niveditha S (2024) [2] defined a predicting malware classification and family modal using ML They applied dynamic analysis using Cuckoo sandbox which extract dynamic features, and threshold value for feature selection, and employed KNN as the classifier. And the system uses automation in feature selection and cross-validation for better F1-score and recall values.

Savino Dambra et al. (2023) [3] have proposed a brand-new approach of Measuring Learning in Malware Classification of Windows based systems or as Machine Learning in Windows Malware Classification, published earlier it arXiv preprint. They accumulated the largest dataset of malware available so far, 118, 111 covering four datasets. The static and dynamic features are analysed using different Machine Learning algorithms with AVClass for family labelling.

Zhenshuo, and Tomas (2021) [4] proposed malware classification model which uses the static features of an samples to classify using ML. They developed a modal which uses static features disassemble and employed autoML with Random Forest to achieving up to 99% accuracy using just 40 features.

Ömer and Refik (2020) [5] presented *A Comprehensive Review on Malware Detection Approaches* This survey covered a broad range of detection techniques, including ML, DL, heuristic, and behavior-based methods. They have concluded that no single method is sufficient for making hybrid approaches are essential for classification. They also highlighted that malware detection is an NP-complete problem, meaning that no universal detection system would exists.

Torres, C and García-Nieto (2019) [6] explored *Machine Learning Techniques Applied to Cybersecurity* in *Cybernetics*. Their study surveyed ML applications such as SVM and ANN across various cybersecurity areas which concludes that ML is highly adaptable for different types of threats.

Aslan and Yilmaz (2021 [7] developed A classification framework using on Deep Learning Algorithms. Their method utilized convolutional neural networks (CNNs) on raw binary data and visual malware representations. The system performed with high classification accuracy but used huge computational resources, making its deployment challenging in large scales.

Al-Kasassbeh et al. [8] explored feature selection methods for malware classification within a machine learning. Their study demonstrated that careful feature engineering significantly improves classification accuracy, but also highlighted the challenge of selecting feature subsets from high-dimensional data.

Fernando, Komninos, and Chen [9] provided an in-depth survey about ransomware detection and its using machine and deep learning (DL). They traced the evolution of detection methods, identifying DL-based models as promising, but also noting scalability and adaptability issues against evolving ransomware.

Rieck et al. [10] presented an automatic analysis framework for malware behavior using machine learning. Their system extracted behavioral features from dynamic analysis and trained classifiers for detection, achieving strong results but requiring significant preprocessing and controlled execution environments.

Anderson et al. [11] investigated graph-based malware detection through dynamic analysis. By modeling program behavior as graphs, they enabled detection of structural similarities among malware families. While effective in capturing relationships, graph construction was computationally expensive.

Hasan and Rahman [12] introduced RansHunt, an SVM-based framework for ransomware analysis. By integrating multiple feature sets, the system enhanced classification accuracy against ransomware samples, though it relied on handcrafted features and required continuous updates to remain effective.

Shreyanth and Niveditha [13] proposed a cluster grid model for wireless network data transmission which incorporates routing protocol analysis with the help of deep learning. Although not exclusively focused on malware, their approach highlighted scalability benefits in handling large datasets relevant to security analytics.

**CHAPTER 2 SYSTEM ANALYSIS**

* 1. **EXISTING SYSTEM**

The existing malware detection systems primarily rely on static analysis methods such as signature matching and disassembly-based feature extraction. These approaches are straightforward, lightweight, and efficient for detecting known malware variants. However, their dependence on predefined patterns makes them vulnerable to code obfuscation, encryption, and polymorphism techniques. Furthermore, static systems often achieve high accuracy only in controlled environments but fail to scale when exposed to large and diverse real-world datasets. As attackers continuously modify code structures, static methods become less effective, highlighting the urgent need for more robust detection strategies that can adapt to new threats.

* 1. **PROPOSED SYSTEM**

The proposed system, **MalwareClassifier**, introduces a two-stage ensemble learning framework to enhance malware detection and classification. In the first stage, binary classification is performed using a stacked ensemble of Logistic Regression, Random Forest, and XGBoost to separate benign programs from malicious samples. This design improves accuracy and robustness by exploiting the complementary strengths of the algorithms, ensuring better generalization across unseen threats.

In the second stage, the system classifies detected malware into families like trojans, ransomware, and spyware using an ensemble of classifiers—Random Forest, XGBoost, and neural networks—combined with a logistic regression meta-learner. This hierarchical design enables efficient detection while offering fine-grained threat intelligence, enhancing real-world cybersecurity defense.

**2.3 IMPLEMENTATION ENVIROMENT**

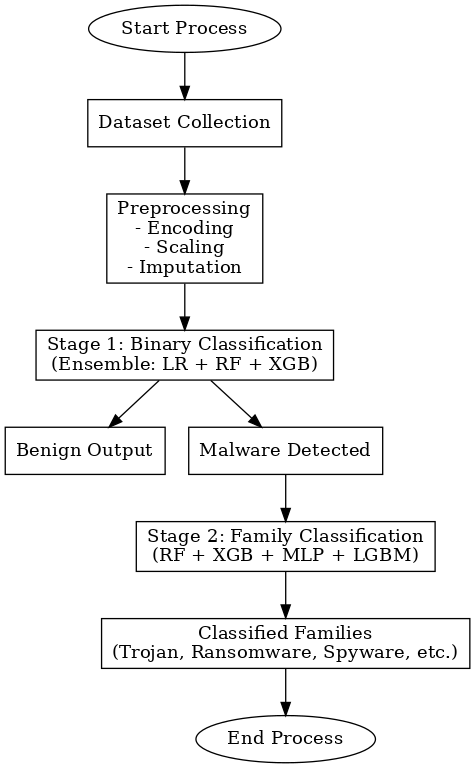
* + 1. **SOFTWARE REQUIREMENT**
* Python.
  + - * Windows 10 or 11
* Scikit-learn.
* XGBoost.
* LightGBM.
  + - * Anaconda Environment
    1. **HARDWARE REQUIREMENT**
* Processor: Intel i5 or above.
* Memory (RAM): 16 GB
  + - * Internet Connection
* GPU (RTX 4050 for acceleration).

**CHAPTER 3**

**SYSTEM DESIGN**

**3.1 UML DIAGRAMS**

**ACTIVITY DIAGRAM**

****

**Fig: 3.1.1. Activity diagram for Betty**

**User / Analyst**: The process begins with the analyst providing a dataset of executables or memory dumps for analysis. These inputs may contain both benign and malicious samples.

**Dataset Collection**: The system collects and organizes these inputs, preparing them for further processing. This ensures that both normal and suspicious files are included in the workflow.

**Preprocessing Stage**: The dataset undergoes preprocessing operations such as missing value handling, feature scaling, and label encoding. This step guarantees that the raw data is transformed into a standardized format suitable for ML models.

**Stage 1 – Binary Classification**: The preprocessed samples are passed to a stacked ensemble consisting of Logistic Regression, Random Forest, and XGBoost. This model decides whether a given sample is benign or malicious.

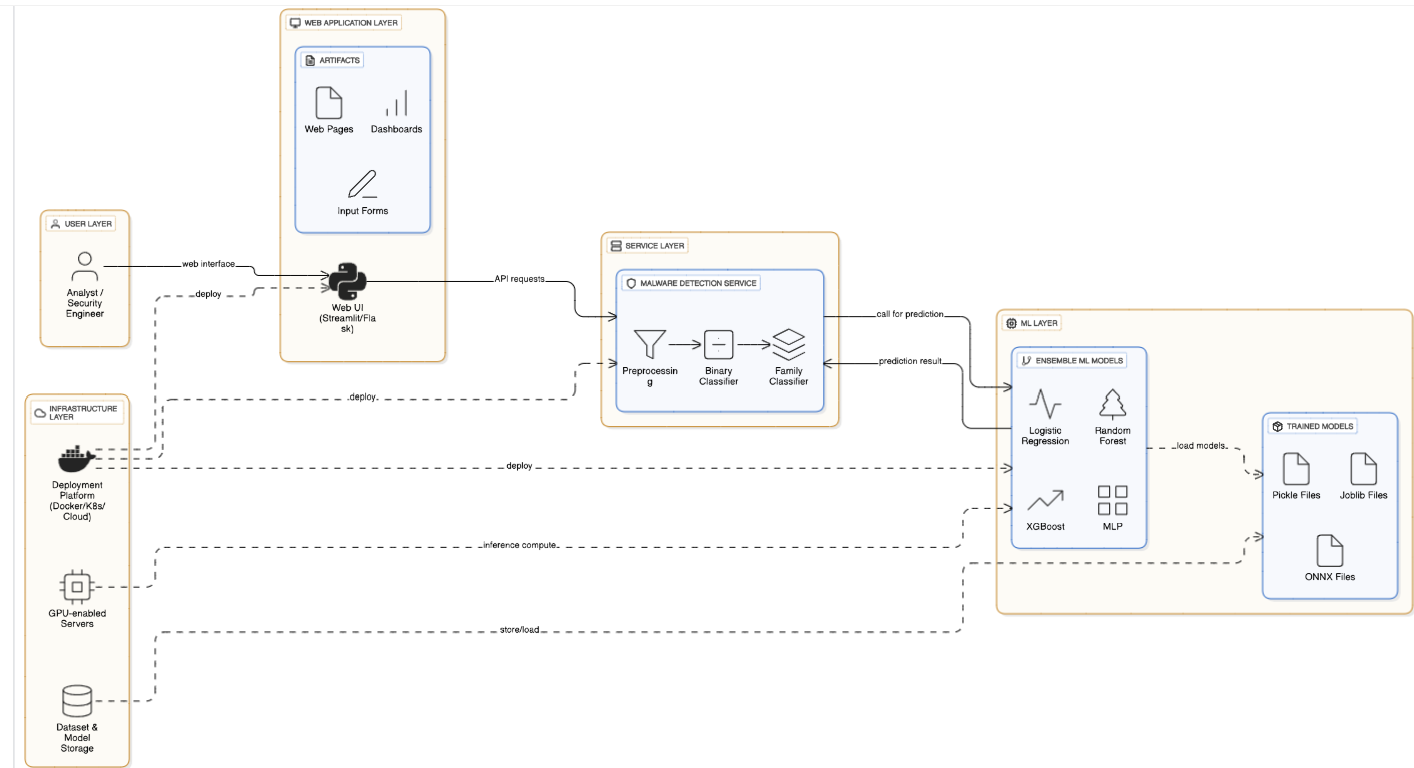
**Benign Output & Malware Detected**: If a sample is classified as benign, the system reports it as safe and no further analysis is required. If the classifier detects a malicious sample, the workflow continues to the second stage for deeper analysis.

**Stage 2 – Family Classification**: The malware samples are further classified into families such as trojans, ransomware, or spyware using another ensemble that combines Random Forest, XGBoost, MLP, and LightGBM.

**Classified Families**: The system provides detailed results, assigning malware to its respective family, which enables cybersecurity professionals to understand its behavior and potential threat impact.

 **End Process**: Finally, the process completes, and the results are presented in a structured form, including both binary detection outcomes and family-level insights.

**DEPLOYMENT DIAGRAM**

****

**Fig: 3.1.2 Deployment diagram for Betty**

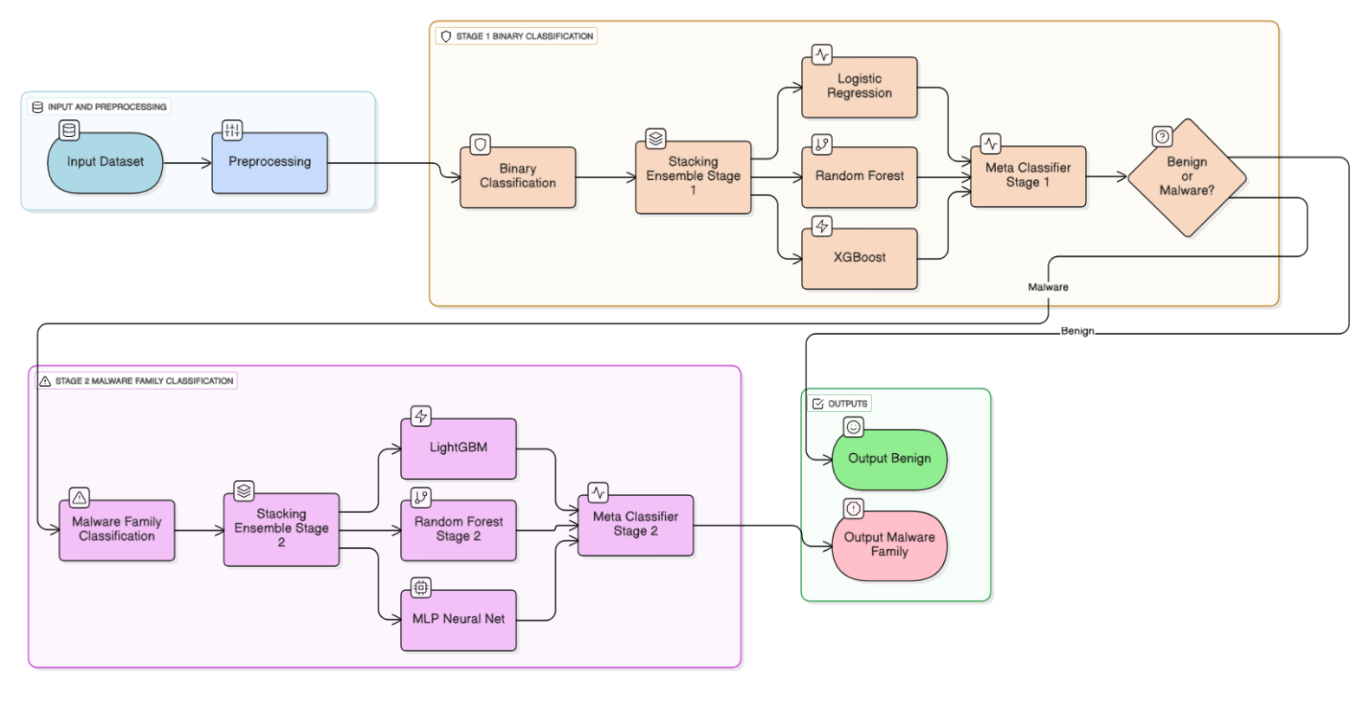
This diagram illustrates a multi-layered architecture for a malware detection system. The **User Layer** consists of analysts or security engineers who interact with the system via a **Web UI** built using Streamlit or Flask. The **Web Application Layer** provides dashboards, and input forms to collect and display insights. The **Service Layer** hosts the **Malware Detection Service**, which preprocesses inputs, runs a binary classifier, and further categorizes malware families. The **ML Layer** contains ensemble models such as Logistic Regression, Random Forest, XGBoost, and MLP. The **Infrastructure Layer** provides deployment platforms (Docker/K8s/Cloud), GPU-enabled servers, and dataset/model storage for computation and scalability. This design ensures seamless deployment, prediction, and analysis in a structured workflow.

**CHAPTER 4**

**SYSTEM ARCHITECTURE**

* 1. **ARCHITECTURE OVERVIEW**

The architecture diagram for a malware detection and family classification system with dynamic features using machine learning outlines the high-level structure and interactions of the system components. Below is a description of each component in the architecture diagram:



**Fig: 4.1.1. System Architecture for Neuro- orchestrator**

**Preprocessing:**

Handles the transformation of raw data into a structured and standardized format suitable for training and classification. Preprocessing removes noise, extracts relevant features, and normalizes inputs. This ensures the classifiers can operate efficiently and accurately on the prepared dataset.

**Stage 1: Binary Classification**

**Binary Classification:**

This stage focuses on distinguishing whether a given sample is benign or malware. It is the first line of defense in the system and ensures only suspicious samples proceed to deeper analysis.

**Stacking Ensemble Stage 1:** Combines multiple machine learning models to improve accuracy and robustness in binary classification. By aggregating outputs from different learners, it reduces the risk of overfitting and enhances generalization.

Logistic Regression, RandomForest, and XGBoost. These are the base classifiers in the ensemble for Stage 1. Each model contributes different strengths: Logistic Regression provides linear separation, Random Forest adds decision-tree-based learning, and XGBoost delivers gradient boosting for high predictive performance.

**Stage 2: Malware Family Classification**

**Malware Family Classification:**

Once a sample is identified as malware, this stage categorizes it into its respective malware family. This step provides deeper insight into the nature of the threat and supports targeted defensive measures.

**Stacking Ensemble Stage 2:**  
Similar to Stage 1, this ensemble combines multiple classifiers to achieve higher accuracy in family classification. It integrates diverse models to capture complex patterns across malware families.

**LightGBM, Random Forest, and MLP Neural Net:**

These are the key models used for family classification. LightGBM provides efficient gradient boosting, Random Forest ensures robust decision-making, and MLP Neural Net captures complex nonlinear relationships in the data.

**Outputs**

**Benign:**When the binary classifier determines that a sample is benign, it is labeled accordingly. This ensures that harmless data does not undergo unnecessary further analysis.

**Malware:**For samples identified as malware, the system outputs the predicted family classification. This provides actionable intelligence for cybersecurity teams to respond more effectively to threat.

* 1. **MODULE DESIGN SPECIFICATION**

### Dataset Preparation and Preprocessing

The quality of any machine learning system depends heavily on the quality of data provided. In malware detection, datasets often contain inconsistencies such as missing feature values, class imbalance, or redundant information. The dataset preparation module handles these issues systematically before model training. Missing data is addressed using **mean imputation**, ensuring that no sample is discarded due to incomplete values. Features are scaled using **StandardScaler** so that high-magnitude attributes (such as file size) do not dominate learning over subtle but significant ones (like API usage counts).

Additionally, categorical labels are encoded to numerical form using **LabelEncoder**, which is crucial for classification algorithms. The class label (Benign or Malware) is used in the binary stage, while the malware families (Trojan, Ransomware, etc.) are encoded separately for Stage 2. Stratified sampling ensures that training and test sets maintain proportional representation of both benign and malware samples.

This design not only prevents bias but also enables the classifiers to learn meaningful boundaries. By enforcing uniform scaling and encoding, the preprocessing pipeline creates a consistent feature space. This guarantees that the same transformation logic is applied during both training and prediction, preventing mismatches that could cause inaccurate classifications.

Finally, preprocessing outputs two sets of labels: one for **binary detection** and one for **multi-class family classification**. This dual-label design allows the system to be modular, supporting hierarchical classification while keeping the workflow clear and adaptable.

### Feature Extraction

The feature extraction module focuses on representing malware behavior and structure in a way that machine learning models can understand. Instead of analyzing raw binaries directly, which is complex and resource-intensive, features are derived from observable attributes. **Static features** include file size, section headers, imported DLLs, and entropy values. These provide quick and lightweight insights into file structure**. Dynamic features** such as API call sequences, registry modifications, and process creation patterns capture runtime behavior that is more resistant to obfuscation.

Combining static and dynamic features creates a hybrid representation that increases detection resilience. For example, a ransomware file may look benign statically but reveal itself through suspicious API calls during execution. To handle such variation, extracted features are stored in structured tabular form suitable for ML pipelines.

To prevent high dimensionality from reducing performance, redundant or non-informative features are filtered. Features contributing little to classification accuracy are pruned using statistical analysis and feature importance metrics from tree-based models. This reduces training time while retaining predictive power.

In the end, this module ensures that each malware sample is represented as a dense, informative feature vector. This becomes the input for Stage 1 (binary classification) and Stage 2 (family classification), enabling accurate detection at multiple levels.

### Binary Malware Detection (Stage 1)

The binary classification module serves as the first gate in the system, tasked with identifying whether a sample is benign or malicious. It employs a **soft voting ensemble classifier** that combines Logistic Regression, Random Forest, and XGBoost. Each model provides distinct advantages. **Logistic Regression** delivers fast linear separation and serves as a baseline. **Random Forest** reduces variance by aggregating multiple decision trees, making it robust to noise. **XGBoost** provides high accuracy through gradient boosting and handles complex feature interactions.

During training, the ensemble learns patterns from labeled datasets, producing a probability score for each class. By applying a configurable threshold, the system can reduce false positives (safe files flagged as malicious) or false negatives (malware passing as benign), depending on the security context.

Cross-validation ensures the model generalizes across unseen data, while metrics such as **ROC-AUC, F1-score, and precision-recall curves** measure performance. The ensemble design is modular, meaning additional classifiers can be integrated if required.

Overall, Stage 1 ensures efficient filtering. Benign files are classified and reported immediately, while suspicious samples are passed to Stage 2 for family classification.

### Malware Family Classification (Stage 2)

The second stage performs **multi-class classification** on samples identified as malware. It uses an ensemble of **Random Forest, XGBoost, LightGBM, and Multi-Layer Perceptron (MLP)**, combined through a voting strategy. Tree-based models capture structured relationships in the dataset, while the neural network identifies complex, non-linear interactions.

This module is specialized for malware-only samples, ensuring that benign data does not interfere with learning. Each family, such as **Ransomware, Spyware, Trojan, Worms, or Adware**, is treated as a separate class. The ensemble outputs a probability distribution across all families, with the highest probability determining the predicted label.

To prevent misclassification, a confidence threshold is introduced. If no family exceeds the threshold, the system labels the sample as “Malware – Unknown”. This avoids false confidence in unseen or emerging malware families.

Performance evaluation includes confusion matrices to visualize misclassifications and F1-scores to balance precision and recall. This ensures the system not only detects malware but also provides detailed intelligence about its type, which is critical for cybersecurity defense and forensic analysis.

**Flow Chart:**

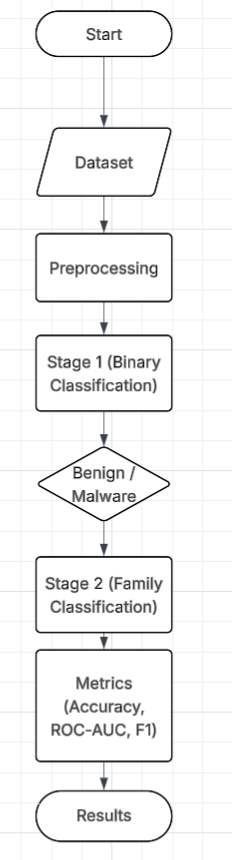
****

Fig 6.1.1 Flow chart of the system

The flowchart begins with the Dataset module, where malware and benign samples are collected. This dataset contains structured features that represent both static and dynamic properties of executables. To ensure consistency and usability, the dataset passes through a Preprocessing stage. Here, missing values are imputed, features are scaled, and labels are encoded to make the data suitable for classification models.

After preprocessing, the data enters Stage 1 – Binary Classification. This stage applies an ensemble of Logistic Regression, Random Forest, and XGBoost to separate benign files from malware. Samples identified as benign are output as safe and no further analysis is performed. Malware samples, however, are directed into the next level for deeper inspection, ensuring computational efficiency and higher reliability in detection.

The Stage 2 – Family Classification step is triggered only for malware-labeled samples. Here, advanced classifiers like Random Forest, LightGBM, XGBoost, and MLP analyze the malicious data to identify the exact malware family. Families such as Trojans, Spyware, and Ransomware are recognized based on their feature patterns. If the confidence level is low, the system assigns the label *“Malware – Unknown”*, ensuring cautious predictions.

**CHAPTER 5**

**SYSTEM IMPLEMENTATION**

**5.1 Preprocessing Implementation**

The preprocessing pipeline ensures consistency across samples. Missing values are handled with mean imputation, features are scaled, and labels are encoded for classification.

Python

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.impute import SimpleImputer

# Initialize preprocessors

scaler = StandardScaler()

le\_family = LabelEncoder()

le\_malware = LabelEncoder()

imputer = SimpleImputer(strategy='mean')

# Apply preprocessing

X\_imputed = imputer.fit\_transform(X)

X\_scaled = scaler.fit\_transform(X\_imputed)

y\_bin = (y\_class == "Malware").astype(int)

y\_multi = le\_family.fit\_transform(y\_family)

This stage ensures that all samples are represented consistently, improving the performance of both binary and family classifiers.

**5.2 Stage 1: Binary Classification**

The first stage uses a soft voting ensemble with Logistic Regression, Random Forest, and XGBoost.

Python

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from xgboost import XGBClassifier

stage1 = VotingClassifier(estimators=[

("LR", LogisticRegression(max\_iter=5000, solver="saga")),

("RF", RandomForestClassifier(n\_estimators=300, max\_depth=12,

n\_jobs=-1, random\_state=42)),

("XGB", XGBClassifier(n\_estimators=300, max\_depth=8, learning\_rate=0.1,subsample=0.8, colsample\_bytree=0.8,

eval\_metric="logloss", n\_jobs=-1, random\_state=42))

], voting="soft", n\_jobs=-1)

stage1.fit(X\_train, y\_train\_bin)

The ensemble achieves high accuracy and reduces false positives.

**5.3 Stage 2: Family Classification**

Malware samples identified in Stage 1 are passed into Stage 2 for family-level classification.

Python

from sklearn.neural\_network import MLPClassifier

from lightgbm import LGBMClassifier

stage2 = VotingClassifier(estimators=[

("LGBM", LGBMClassifier(n\_estimators=500, max\_depth=12,

learning\_rate=0.05, subsample=0.8,

colsample\_bytree=0.8, random\_state=42)),

("RF", RandomForestClassifier(n\_estimators=400, max\_depth=14,

n\_jobs=-1, random\_state=42)),

("MLP", MLPClassifier(hidden\_layer\_sizes=(256,128),

max\_iter=300, random\_state=42))

], voting="soft", n\_jobs=-1)

stage2.fit(X\_train\_malware, y\_train\_family\_enc)

This ensures reliable classification of families such as trojans, ransomware, and spyware.

**5.4 Prediction Function**

The system predicts benign/malware in Stage 1 and family classification in Stage 2 with configurable thresholds.

Python

def predict(X, stage1\_threshold=0.7, stage2\_threshold=0.5):

X\_scaled = scaler.transform(X)

results = []

for i in range(len(X\_scaled)):

proba\_stage1 = stage1.predict\_proba(X\_scaled[i].reshape(1, -1))[0]

if proba\_stage1[1] < stage1\_threshold:

results.append("Benign")

continue

proba\_stage2 = stage2.predict\_proba(X\_scaled[i].reshape(1, -1))[0]

fam\_idx = np.argmax(proba\_stage2)

fam\_prob = proba\_stage2[fam\_idx]

if fam\_prob < stage2\_threshold:

results.append("Malware - Unknown")

**5.5 Backend cuckoo Sandbox Integration**

# Base API URL (adjust if your Cuckoo server runs elsewhere)

CUCKOO\_API = "http://127.0.0.1:8090"

# ===== Submit a file for analysis =====

def submit\_file(file\_path):

url = f"{CUCKOO\_API}/tasks/create/file"

with open(file\_path, "rb") as f:

files = {"file": (file\_path, f)}

response = requests.post(url, files=files)

if response.status\_code == 200:

task\_id = response.json()["task\_id"]

print(f"[+] File submitted successfully! Task ID: {task\_id}")

return task\_id

# ===== Submit a URL for analysis =====

def submit\_url(target\_url):

url = f"{CUCKOO\_API}/tasks/create/url"

data = {"url": target\_url}

response = requests.post(url, data=data)

if response.status\_code == 200:

task\_id = response.json()["task\_id"]

print(f"[+] URL submitted successfully! Task ID: {task\_id}")

return task\_id

else:

print(f"[-] Error submitting URL: {response.text}")

return None

# ===== Check task status =====

def check\_status(task\_id):

url = f"{CUCKOO\_API}/tasks/view/{task\_id}"

response = requests.get(url)

if response.status\_code == 200:

task\_info = response.json()["task"]

print(f"[\*] Task {task\_id} status: {task\_info['status']}")

return task\_info["status"]

else:

print(f"[-] Error checking status: {response.text}")

return None

# ===== Download analysis report =====

def get\_report(task\_id, report\_format="json"):

url = f"{CUCKOO\_API}/tasks/report/{task\_id}/{report\_format}"

response = requests.get(url)

if response.status\_code == 200:

report\_data = response.json()

output\_file = f"report\_{task\_id}.{report\_format}"

with open(output\_file, "w") as f:

json.dump(report\_data, f, indent=4)

print(f"[+] Report saved as {output\_file}")

else:

print(f"[-] Error retrieving report: {response.text}")

# ===== Example workflow =====

if \_\_name\_\_ == "\_\_main\_\_":

# Submit a file or URL

task\_id = submit\_file("sample.exe")

# task\_id = submit\_url("http://malicious-example.com")

# Poll until the analysis is complete

if task\_id:

while True:

status = check\_status(task\_id)

if status == "reported":

print("[+] Analysis complete!")

break

elif status in ["failed\_analysis", "completed"]:

print("[!] Task finished with status:", status)

break

else:

print("[\*] Waiting for completion...")

time.sleep(10)

get\_report(task\_id)

**CHAPTER 6**

* 1. **PERFORMANCE EVALUATION**

**User Engagement Formula:**

* **Accuracy =**

***Measures the ratio of correct classifications to total predictions.***

* **Precision =**

**fraction of true malware among detected malware.**

* **Recall =**

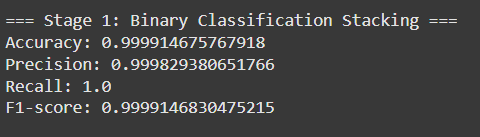
**fraction of actual malware correctly detected.**

* **F1-Score =**

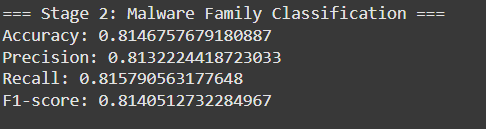
**balances precision and recall.**

**ROC-AUC:**Evaluates the area under the Receiver Operating Curve, highlighting the trade-off between true positives and false positives.

**Attained Performance Score:  
  
Stage 1:**



**Stage 2:**



Finally, the system enters the Performance Metrics and Results stage. Accuracy, Precision, Recall, F1-score, and ROC-AUC are calculated to assess the overall effectiveness of both stages. Confusion matrices further visualize family-level misclassifications. This performance evaluation provides a clear understanding of how well the framework generalizes, highlighting its ability to balance accuracy with robustness in real-world malware detection.

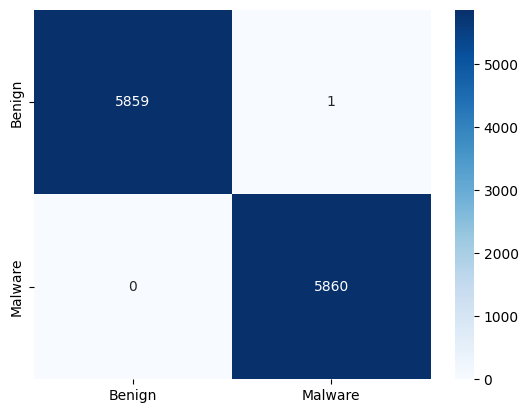


Fig 6.1.2: confusion matrix

The confusion matrix illustrates the performance of the classification model by comparing predicted outputs with actual labels. The diagonal values represent correctly classified samples, while off-diagonal values indicate misclassifications. In this project, the confusion matrix provides insight into how well benign and malware samples are distinguished. It highlights the model’s ability to minimize both false positives and false negatives.

#### By analyzing the confusion matrix, we can identify which malware families are most prone to misclassification. This helps in understanding weaknesses of the current feature set or model design. High values along the diagonal demonstrate the reliability of the ensemble framework in classifying both benign and malicious samples. Overall, the confusion matrix validates the effectiveness of the system and guides areas for improvement.

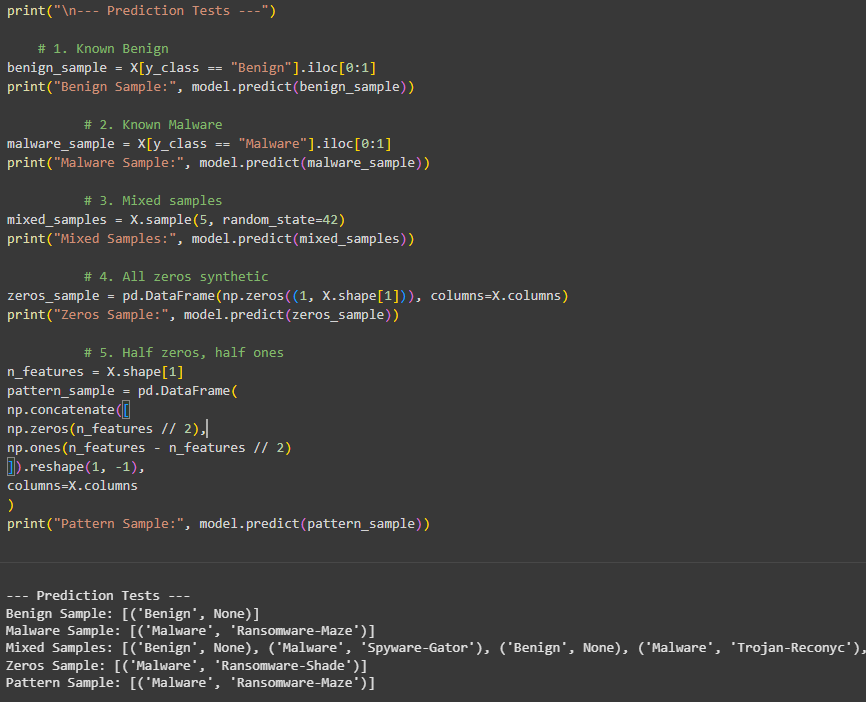
#### 

Figure 6.1.3: Confusion matrix heatmap

The heat matrix provides a visual representation of the correlation between different features within the dataset. Darker shades indicate stronger relationships, while lighter shades show weaker or no correlation. This figure helps in identifying redundant or highly dependent attributes that may affect classification accuracy. By analyzing these relationships, unnecessary features can be pruned to optimize the model.

Feature correlation observed in the heat matrix also highlights which attributes contribute most to distinguishing malware families. Strongly correlated features often represent consistent behavioral patterns such as API usage or file entropy. This visualization supports effective feature engineering by guiding the selection of relevant inputs. Overall, the heat matrix ensures that the system uses meaningful features for robust malware detection.

* 1. **MANUAL TEST CASES**

****

* 1. **RESULTS AND DISCUSSION**

The Malware Classifier was evaluated on the CIC-MalMem-2022 dataset, which includes benign programs and malware families such as trojans, spyware, and ransomware. Preprocessing with label encoding, feature scaling, and missing value imputation produced a clean, normalized dataset for effective training. In binary classification, a stacked ensemble of Logistic Regression, Random Forest, and XGBoost accurately separated benign and malicious programs, reducing false positives and improving generalization. The multi-class family classification stage leveraged tree-based algorithms and neural networks to classify malware samples into families, with accuracy, F1-score, and confusion matrices showing consistent improvements. GPU acceleration for XGBoost and MLP models reduced computational overhead, enabling large-scale deployment. By using dynamic behavioral features, the system remained resilient against obfuscation and polymorphism and efficiently handled over 58,000 samples, demonstrating scalability. Overall, MalwareClassifier enhances detection accuracy while providing detailed family-level insights, making it a practical solution for real-world malware detection and threat intelligence generation.

**CHAPTER 7**

**CONCLUSION AND FUTURE WORK**

* 1. **CONCLUSION**

This project introduced a two-stage ensemble learning framework, MalwareClassifier, for malware detection and family classification. By combining Logistic Regression, Random Forest, XGBoost, LightGBM, and MLP, the system achieved high accuracy and robustness in detecting both benign and malicious samples. Preprocessing methods, including imputation, scaling, and label encoding, enhanced generalization across diverse datasets. Stage 1 effectively filtered benign programs, while Stage 2 provided fine-grained family classification, offering valuable threat intelligence. Experimental evaluations confirmed improvements in precision, recall, and ROC-AUC compared to static analysis methods. The system’s resilience against obfuscation and polymorphism highlights its practical applicability. GPU acceleration ensured efficiency, making the solution scalable for real-world deployment. Overall, this work demonstrates how ensemble-based ML frameworks can bridge the gap between academic research and operational cybersecurity systems.

* 1. **FUTURE ENHANCEMENT**

Future enhancements may include integration of advanced deep learning techniques, such as transformer-based architectures or graph neural networks, to capture more complex relationships in malware behavior. This would further improve accuracy in detecting novel threats.

Another direction is to incorporate real-time monitoring and adaptive learning mechanisms. By continuously updating the models with new data, the system can remain resilient against rapidly evolving malware families.

Additionally, explainable AI (XAI) methods could be embedded to provide transparency in predictions. This would allow security analysts to interpret results more effectively and build trust in automated malware detection frameworks.

**CHAPTER 8**

**APPENDICES**

* 1. **SDG GOALS**

**Industry, Innovation, and Infrastructure (SDG 9):**

The project contributes to secure digital infrastructure by introducing scalable AI-driven malware detection systems that strengthen industrial cybersecurity practices.

**Sustainable Cities and Communities (SDG 11):**

By protecting smart city infrastructures and IoT devices against malware attacks, this system supports the safe development of connected and resilient communities.

**Peace, Justice, and Strong Institutions (SDG 16):**

Strengthening defenses against cybercrime builds trust in governance, e-services, and financial institutions, ensuring justice and protection of sensitive data.

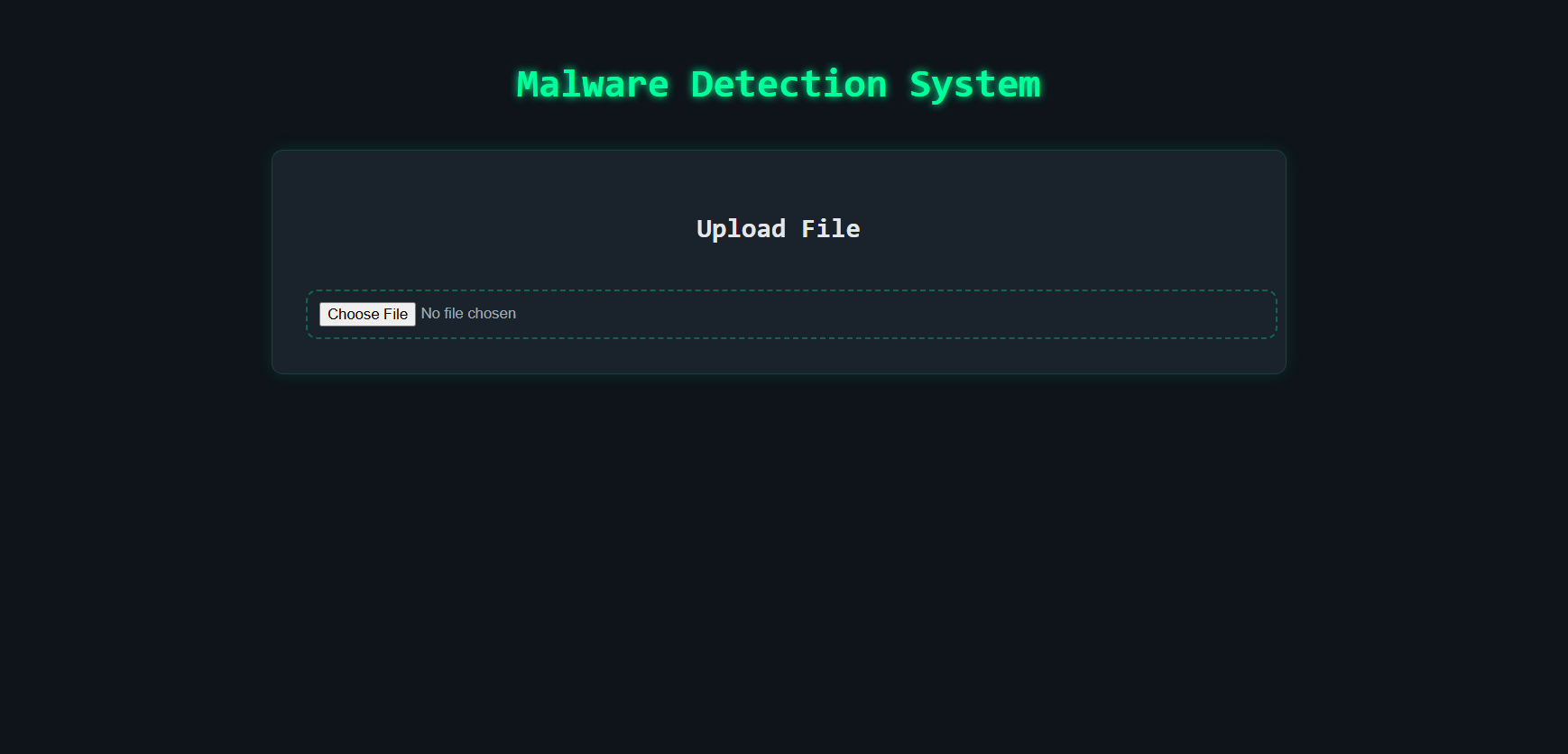
**Quality Education (SDG 4):**

The project also contributes to knowledge-sharing by serving as a reference for machine learning applications in cybersecurity, thereby improving education and research outcomes.

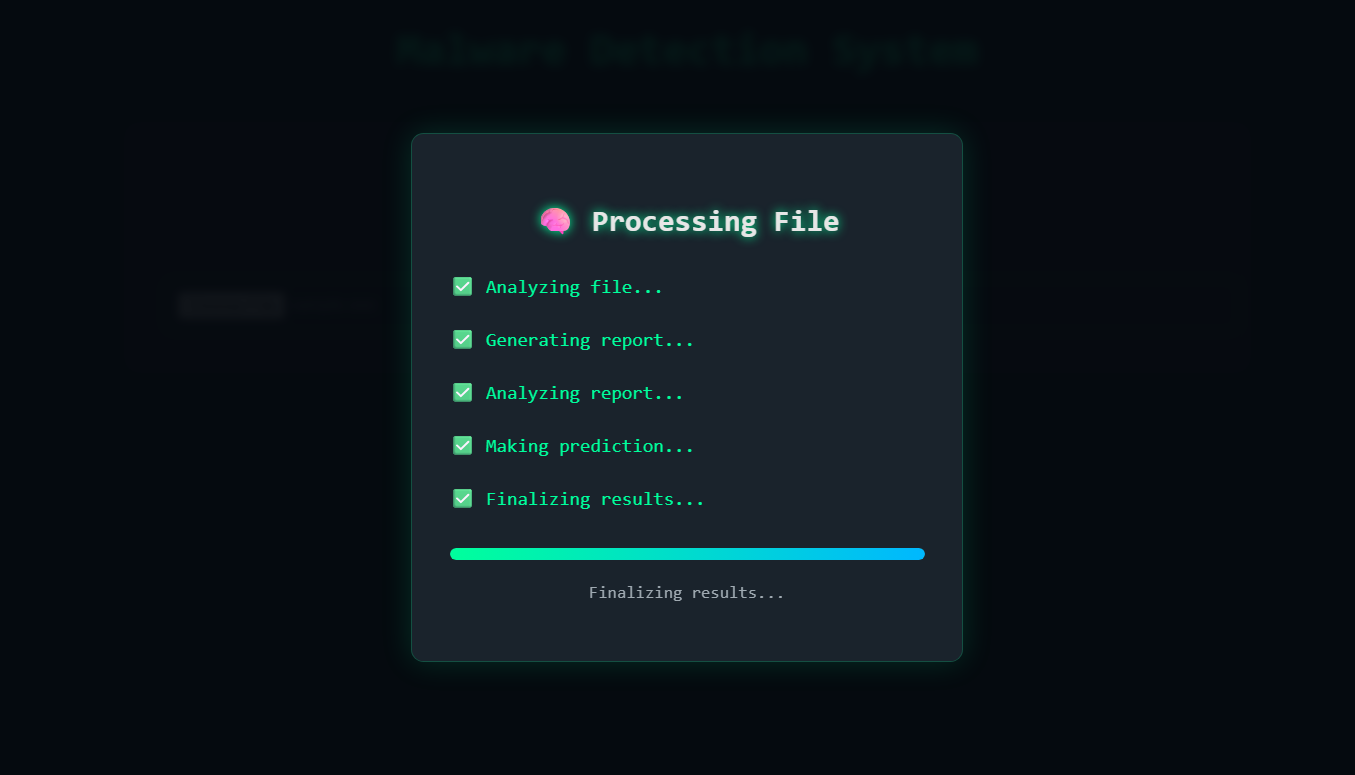
**Decent Work and Economic Growth (SDG 8):**

By reducing losses from cyberattacks and protecting digital businesses, the system ensures secure growth in technology-driven industries.

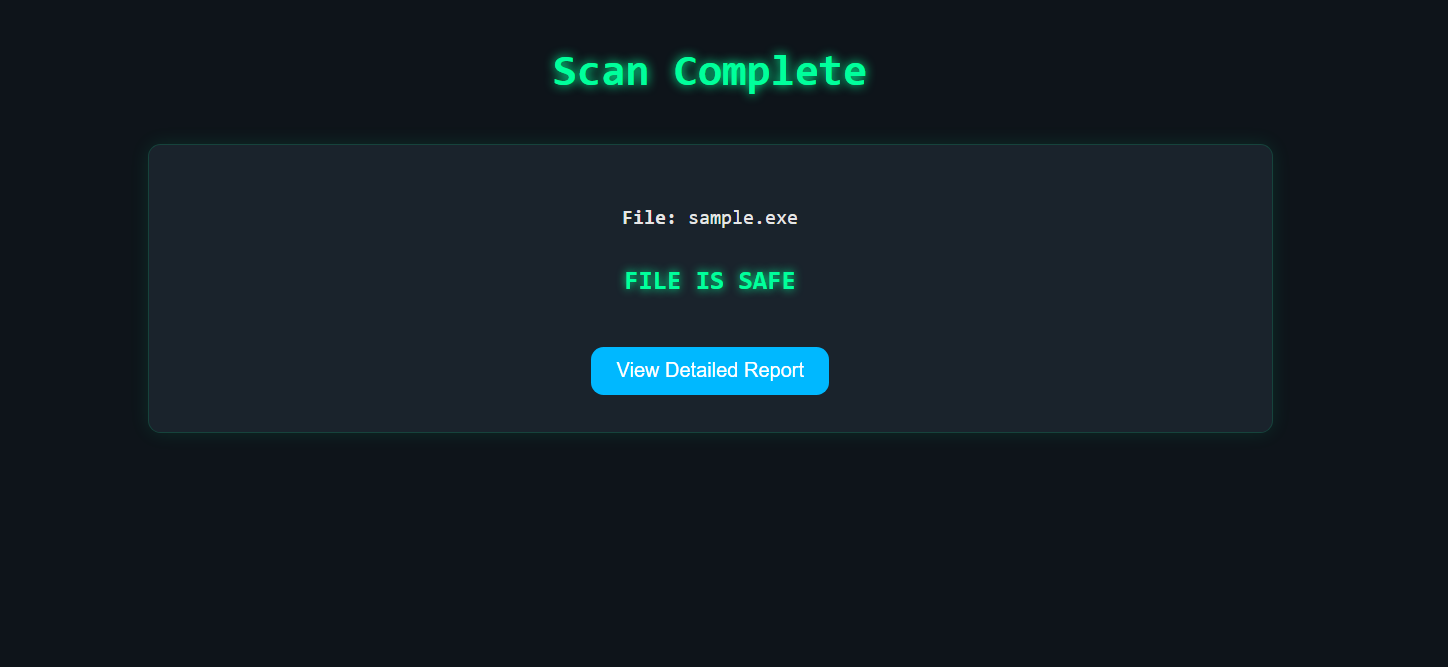
# SCREENSHOTS:

1. ****

**Fig: A.8.1. Screenshot of basic Home Page**

****

**Fig: A.8.2. Screenshot of Processing**



**Fig: A.8.3. Screenshot of Result page**

****

**Fig A.8.4. Screenshot of Report**

# A3. PAPER PUBLICATION

****

**A4. PLAGIARISM REPORT**

****

**CHAPTER 9**

# REFERENCES

1. D R, T A, M T (July 04, 2025) Malware Classification Using Machine Learning and Deep Learning: A Comprehensive Approach. Cureus J Comput Sci 2 : es44389-025-05024-y. <https://doi.org/10.7759/s44389-025-05024-y>
2. S, Niveditha & Rr, Prianka & K, Sathya & S, Shreyanth & Subramani, Nandhagopal & Deivasigamani, Balakrishnan & S, Karthikeyan. (2024). Predicting Malware Classification and Family using Machine Learning.   
   235. 2434-2451. 10.1016/j.procs.2024.04.230.
3. Savino Dambra et al., arXiv (2023). Decoding the Secrets of Machine Learning in Windows Malware Classification: A Deep Dive into Datasets, Features, and Model Performance. <https://doi.org/10.48550/arXiv.2307.14657>.
4. Zhenshuo Chen, Eoin Brophy, Tomas Ward (2021).Malware Classification Using Static Disassembly and Machine Learning. <https://doi.org/10.48550/arXiv.2201.07649>.
5. Aslan, Ömer & Sam Refik. (2020). A Comprehensive Review over Malware Detection Approach. IEEE Access. 8. 1-1. 10.1109/ACCESS.2019.2963724.
6. Martínez Torres, J., Iglesias Comesaña, C. & García-Nieto, P.J. Review: machine learning techniques applied to cybersecurity. *Int. J. Mach. Learn. & Cyber.* **10**, 2823–2836 (2019). <https://doi.org/10.1007/s13042-018-00906-1>
7. Aslan, Ö. and Yilmaz, A.A. (2021) A New Malware Classification Framework Based on Deep Learning Algorithms. *IEEE Access*, 9, 87936-87951.  
   <https://doi.org/10.1109/ACCESS.2021.3089586>
8. Al-Kasassbeh, M., Mohammed, S., Alauthman, M., Almomani, A. (2020). Feature Selection Using a Machine Learning to Classify a Malware.
9. D. W. Fernando, N. Komninos, and T. Chen, “A Study on the Evolution of Ransomware Detection Using Machine Learning and Deep Learning Techniques,” IoT, vol. 1, no. 2, pp. 551–604, Dec. 2020. <https://doi.org/10.3390/iot1020030>
10. Rieck, Konrad et al. "Automatic Analysis of Malware Behavior Using Machine Learning", Journal of Computer Security, vol. 19, no. 4, pp. 639-668, 2011. <https://doi.org/10.3233/JCS-2010-0410>
11. Anderson, B., Quist, D., Neil, J. et al. Graph-based malware detection using dynamic analysis. J Comput Virol 7, 247–258 (2011). <https://doi.org/10.1007/s11416-011-0152-x>
12. M. M. Hasan and M. M. Rahman, "RansHunt: A support vector machines-based ransomware analysis framework with integrated feature set," 2017 20th International Conference of Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 2017, pp. 1-7. <https://doi.org/10.1109/ICCITECHN.2017.8281835>
13. Shreyanth S. and Niveditha S. (2023); Cluster-Based Grid Computing on Wireless Network Data Transmission with Routing Analysis Protocol and Deep Learning, Int. J. of Adv. Res. 11 (Jun). 517-534. <http://dx.doi.org/10.21474/IJAR01/1709618>
14. Yuki Takeuchi, Kazuya Sakai, and Satoshi Fukumoto. 2018. Detecting Ransomware using Support Vector Machines. In Workshop Proceedings of the 47th International Conference on Parallel Processing (ICPP Workshops '18). Association for Computing Machinery, New York, NY, USA, Article 1, 1–6. <https://doi.org/10.1145/3229710.3229726>
15. A. O. Almashhadani, M. Kaiiali, S. Sezer and P. O’Kane, "A Multi-Classifier Network-Based Crypto Ransomware Detection System: A Case Study of Locky Ransomware," in IEEE Access, vol. 7, pp. 47053-47067, 2019. <https://doi.org/10.1109/ACCESS.2019.2907485>
16. S. Poudyal, K. P. Subedi and D. Dasgupta, "A Framework for Analyzing Ransomware using Machine Learning," 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Bangalore, India, 2018, pp. 1692-1699. <https://doi.org/10.1109/SSCI.2018.8628743>